Miika Mehtälä

HTML5 tablet uutissovelluksen suunnittelu ja toteutus

Tietotekniikan
pro gradu -tutkielma
31. lokakuuta 2013

Jyväskylän yliopisto

Tietotekniikan laitos

Kokkolan yliopistokeskus Chydenius

Tekijä: Miika Mehtälä
Yhteystiedot: miika.mehtala@mbnet.fi
Puhelinnumero: 040 6722200
Ohjaaja: Risto T. Honkanen
Työn nimi: HTML5 tablet uutissovelluksen suunnittelu ja toteutus
Title in English: Design and implementation of HTML5 tablet news application
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 104
Tiivistelmä: Keskipohjanmaa-lehdestä haluttiin toteuttaa web-pohjainen, HTML5:tä hyödyntävä tablet-uutissovellus Keski-Pohjanmaan Kirjapaino Oyj:lle. Työssä tutkittiin uusimpia web-teknologioita, kannettavia laitteita, web-sovelluksen tekemistä ja tablet-sovelluksen luomista sanomalehdestä. Työn aikana toteutettiin kolme prototyyppiä ja kaksi virallista tablet-sovellusversiota. Ensimmäisessä virallisessa versiossa rakenne ja data olivat yhdessä sekä osa sovelluslogiikasta oli palvelimen koodissa. Toisessa versiossa data ja rakenne olivat erillään ja sovelluslogiikka oli tehty Javascriptillä. Tehdessä web-sovellusta tablet-laitteille selvisi, että on huomioitava suorituskyky ja Internet-selainten eroavaisuudet. Sovellusta on hyvä testata kohdelaitteilla. Tutkimuksen perusteella voidaan suositella tablet-uutissovelluksen toteuttamista nimenomaan web-sovelluksena.
Avainsanat: HTML5, Web, Tablet, iPad, sanomalehti
Abstract: The need was to create a web-based tablet news application that utilizes HTML5 for Keski-Pohjanmaan Kirjapaino Oyj. New web technologies, portable devices, the development of web-applications and the transformation from printed newspaper to tablet application were researched. Three prototypes and two official tablet news application versions were created. In the first official version, the application structure and data were embedded together and some of the logic were on the server-side code. On the other version, the data and structure were separated and the logic was made on with client-side Javascript. While making web-application for tablet devices it became clear that performance and the differences in web-browsers should be noted. Best practice would be to test on the target device. Based on this study, it is recommended to implement the newspaper application as a web application.
Keywords: HTML5, Web, Tablet, iPad, newspaper

Sanasto

Android	Mobiililaitteille suunniteltu käyttöjärjestelmä.
API	Application Programming Interface. Lähdekoodimääritelmä.
C#	Microsoftin ohjelmointikieli.
CSS	Cascading Style Sheet. Web-sivuissa käytettävä tyylimäärittelykieli.
DHTML	Dynamic HTML. Yhdistelmä HTML:ää, CSS:ää, Javascriptiä ja DOM:ia.
DOM	Document Object Model. HTML/XML -dokumenttien objektien esitystapa.
Flash	Kehitysympäristö, jolla voidaan esittää multimediasisältöä web-sivussa.
HTML	Hypertext Markup Language. Www-sivujen kuvauskoodikieli.
Java	Oliopohjainen ohjelmointikieli.
Javascript	Web-sivustoissa käytettävä koodikieli.
LMS	Learning Management System. Verkossa toimiva oppimisympäristö.
MIME	Multipurpose Internet Mail Extensions. Määrittelee merkkivalikoimia ja Internetin mediatyyppejä.
RIA	Rich Internet Application. Selaimen lisäosan avulla web-sivussa toimiva ohjelma.
SDK	Software Development Kit. Ohjelmistokehityspaketti.
XML	Extensible Markup Language. Merkintäkieli.
.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto.
RSS	Really Simple Syndication. XML:n perustuva sisältöprotokolla.
IIS	Internet Information Services. Palvelinohjelmistokokonaisuus.
ViewState	Ominaisuus, jolla voidaan tallentaa web-sivun tila sivupäivitysten yli.

Sisältö

1 Johdanto

Internet on kehittynyt dokumenttien jako -verkosta maailmanlaajuiseksi sovellusalustaksi. Internet-selaimet näyttävät HTML-määritysten mukaan web-sivun ulkoasun ja käyttöliittymän. Selaimessa toimiva Javascript-moottori suorittaa koodia käyttäjän selaimessa ja mahdollistaa näin reaaliaikaisten sovellusten tekemisen. Työskentelen Keski-Pohjanmaan Kirjapaino Oyj:n (`http://konserni.kpk.fi`) kuuluvassa Kosila Digimediassa (`http://www.kosila.fi`), jossa mm. toteutetaan websivustoja ja web-pohjaisia sovelluksia. Keskipohjanmaa-lehdestä haluttiin toteuttaa tabletissa toimiva web-pohjainen, Javascriptiä käyttävä ja HTML5:tä (Hypertext Markup Language versio 5) hyödyntävä web-sovellus.

Työn alkuosa koostuu kirjallisuuskatsauksesta. Työtä varten tutkittiin webin kehitystä nykymuotoonsa. Ensin tutkittiin uusimpia web-teknologioita ja HTML:n uusinta 5:s versiota ja sen tuomia ominaisuuksia. Haettiin tietoja ja tutkittiin myös web-sovelluksia ja niiden eroja natiiveihin sovelluksiin. Tietoa haettiin myös kannettavista lukulaitteista, joihin kuuluvat e-kirjalaitteet ja tablet-tietokoneet. Lisäksi tutkittiin näiden laitteiden käytettävyyttä. Tutkittiin myös, miten sanomalehdestä voitaisiin muodostaa tablet-uutissovellus ja selvitettiin käyttöliittymän suunnittelussa huomioon otettavia asioita. Tutkimusta tehtiin hakemalla tietoa artikkeleista, kirjallisuudesta ja Internet-lähteistä.

Työn loppuosa koostuu konstruktiivisen tutkimuksen raportoinnista. Tablet-uutissovellusta varten toteutettiin kolme prototyyppiä ja hahmoteltiin käyttöliittymää kuvankäsittelyohjelmalla. Prototyyppien jälkeen uutisaineistosta tehtiin RSS-syöte (Really Simple Syndication) ja sitä testattiin erilaisissa RSS-syötettä tukevissa lukusovelluksissa. Sitten toteutettiin kaksi virallista sovellusversiota. Sovelluskehitys toteutettiin käyttäen ketteriä menetelmiä ja scrumia. Kaksi eri versiota oli rakenteeltaan erilaisia. Käyttöliittymä oli molemmissa samankaltainen. Ensimmäisessä versiossa rakenne ja data olivat yhdessä. Sovelluksessa oli paljon sovelluslogiikkaa palvelimen koodissa. Toisessa versiossa data ja rakenne eroteltiin toisistaan. Sovellus oli kokonaan toteutettu Javascriptilla ja sen koodi ajettiin siis täysin käyttäjän laitteessa. Palvelimelta ladattiin vain uutismateriaali. Lopulta saatiin aikaiseksi kaksi hyvin toimivaa HTML5:tä hyödyntävää web-sovellusta, jotka otettiin käyttöön.

Tehdessä HTML5 web-sovellusta tablet-laitteille selvisi, että tablet-laitteissa pitää ottaa huomioon suorituskyky ja laitteissa olevien Internet-selainten eroavaisuudet. Selaimet tukevat eri HTML5 ominaisuuksia ja renderöivät jotkin HTML-sisällöt vaihtelevilla tavoilla. Ei riitä, että testaa sovellusta perinteisellä tietokoneella, vaan sovellusta on testattava oikeasti kohdelaitteilla. Sovelluksen ominaisuuksissa on huomioitava se, mihin laitteiden selaimet pystyvät. Tutkimuksen antaman tietämyksen perusteella voidaan todeta, että on erittäin suositeltavaa toteuttaa tablet-uutissovellus nimenomaan web-sovelluksena. Web-sovelluksista saa hyvin toimivia ja koodi tarvitsee kirjoittaa vain kerran ja se toimii kaikilla laitteilla.

Luvussa 2 esitellään webin kehitystä dokumenttien jako -verkosta sovellusympäristöksi. Seuraavassa luvussa 3 kerrotaan HTML5:stä ja uusista web-teknologioista. Lisäksi luvussa verrataan natiivia koodia Javascriptiin ja HTML5:n. Luvussa 4 käydään läpi kannettavia laitteita ja niiden käytettävyyttä. Luvussa kerrotaan e-kirjalaitteista ja tablet-tietokoneista ja käydään läpi niiden käytettävyyttä. Luvussa 5 tutkitaan tablet-uutissovelluksen muodostamista sanomalehdestä. Luvussa selvitetään myös uutissovelluksen suunnittelua, mainontaa, ideointia sekä käyttöliittymän suunnittelua. Samalla esitellään myös olemassa olevia HTML5-uutissovelluksia. Luvussa 6 esitellään Keski-Pohjanmaan Kirjapaino Oyj:lle toteutettu tablet-uutissovellus. Ensin käydään läpi suunnittelua ja prototyyppejä. Sitten käydään läpi ensimmäisen virallisen version toteutus. Lopuksi käydään läpi vielä toinen virallinen versio ja verrataan sitä ensimmäiseen versioon. Luvussa 7 on yhteenveto läpikäydyistä asioista.

2 Webin kehitys

HTML (Hypertext Markup Language) on yksinkertainen merkkauskieli, jolla kuvataan HTML-dokumentin rakenne [57, s. 9]. Dokumentissa voidaan näyttää tai siihen voidaan linkittää mediatiedostoja. Tällaisia voivat olla esimerkiksi kuva- ja äänitiedostot. HTML-sivuun voidaan määritellä tyylejä ulkoasuun. Tyylejä voivat olla esimerkiksi tekstin vahvistus, tekstin alleviivaus, taustaväri ja niin edelleen. HTML-dokumentteihin voidaan luoda linkkejä toisiin HTML-dokumentteihin. Itse dokumentit voivat olla fyysisesti eri puolilla maailmaa. HTML-dokumentit ovat tarkoitettu käytettäväksi www:ssä (World Wide Web). Dokumentit ovat määritelty niin, että Internetselaimet osaavat näyttää niitä ja käyttäjät voivat selata dokumentteja ja liikkua niiden välillä linkkejä klikkaamalla. [49]

Nykyisen webin käyttö ja suosio ovat alkuaikoihin verrattuna räjähdysmäisesti kasvaneet ja web onkin muuntunut ympäristön mukana. Nykyinen web ei ole enää pelkkä dokumenttien jakojärjestelmä, vaan sitä voidaan käyttää sovellusalustana ja siellä voidaan jakaa kaikenlaista sisältöä. Tässä luvussa käydään läpi webin kehitystä ja sitä, kuinka web on kehittynyt nykyiseen muotoonsa. Antero Taivalsaari ja Tommi Mikkonen kirjoittavat artikkelissaan *The Web as an Application Platform: The Saga Continues* webin käyttämisestä sovellusalustana [59]. Web ei ole pelkkä sivustokirjasto, vaan sinne voidaan rakentaa interaktiivisia työpöytämäisiä alustariippumatonta 3D-grafiikkaa käyttäviä sovelluksia. Nykyään ollaan kaukana alkuaikojen dokumenttien jakojärjestelmästä uusien teknologioiden myötä.

Mikkonen ja Taivalsaari [59] mainitsevat kolme erikseen eroteltavaa aikakautta webin kehityksessä: Ensimmäisenä aikakautena web oli dokumenttiympäristö, jossa ohjelmointimahdollisuudet olivat rajatut. Toisena kautena sovelluskehitysmahdollisuudet webissä alkoivat nousta, jossa eri teknologiat kilpailivat keskenään. Kolmantena kautena, joka on nyt menossa, webin toimiminen sovellusalustana vaikuttaa ohjelmistoteollisuuteen ja siihen miten sovelluksia tehdään.

2.1 Ensimmäinen aikakausi: klassinen- ja hybridiweb (hybrid web)

Www luotiin tiedonjakamista varten. HTML luotiin määräämään standardi dokumenteille ja tiedon esittämiselle Internetissä [15]. Alkuperäinen www tai web sisälsi pelkästään tekstidokumentteja [15]. Klassisen webin alkuaikoina 1990-luvun alussa sivustot olivat dokumentteja, joissa oli tekstiä, lomakkeita ja kuvia [59]. Sivut sisälsivät linkkejä toisiin sivuihin, eikä asynkronista liikennettä ollut [59]. Sivut sisälsivät aina kaiken sisällön, mitä sivustolla oli tarjota. Jos sama graafinen ulkoasu haluttiin säilyttää toisella sivulla, niin sama sisältö täytyi kopioida jokaiseen tiedostoon [15]. Käytännön esimerkkinä, jos etusivulla oli linkki yhteystiedot-sivulle, täytyi tehdä etu- sekä yhteystiedot-sivu erikseen ja luoda molempiin sama sisältö. Yhteystiedot-sivun sisältöosuuteen vain lisättiin halutut yhteystiedot.

1990-luvun loppupuolella tuli hybridiweb ja sen myötä DHTML [83], joka on yhdistelmä HTML:ää [49], CSS:ää (Cascading Style Sheet) [50], Javascriptiä (koodikieli) [32, s. 1–2] ja DOM:ia (Document Object Model). Mahdollisuudet lisääntyivät sen myötä ja voitiin luoda sivustoja, joissa oli mahdollista käyttää grafiikkaa ja animaatioita laajemmin [59]. CSS-tyyleillä pystyi määritellä sivuston ulkoasua laajasti ja tehdä yksilöllisempiä sivustoja. Sivustojen käytettävyyttä pystyttiin suunnittelemaan myös grafiikan ja Javascriptin avulla.

DHTML:n myötä webin kaupallisuus lähti liikkeelle 1990-luvun lopussa, kun sivustoilla alettiin näyttää mainontaa. Samaan aikaan Java-koodikieli ja Java appletit nostivat suosiotaan [59]. Java applet -elementtiä käytettiin lisäämään Java-sovellus web-sivuun [15]. Java appleteilla HTML-sivuun voitiin lisätä animaatioita, interaktiivisuutta ja mitä tahansa Java-ohjelmointikielellä pystyi tekemään [15]. 1990-luvun lopulla näytti siltä, että appletit olisivat pääsovelluskehitysalustana [59]. Näin ei kuitenkaan käynyt. Yritykset ymmärsivät, että webissä oli mahdollista tehdä rahaa mainonnalla ja myymällä tavaraa [59]. Sivustoille voitiin myydä mainoksia ja jos kävijämäärät olivat suuret, niin ylläpitäjä tienasi rahaa, vaikka itse sivun sisältö oli ilmaista. Webiin kehitettiin selainlisäosia kuten Flash, Realplayer, Quicktime ja Shockwave. Näillä lisäosilla webistä voitiin tehdä interaktiivisempi, lisätä visuaalista näyttävyyttä ja videoita [59].

2.2 Toinen aikakausi: Web mahdollisena sovellusympäristönä

Webin toinen aikakausi alkoi 2000-luvun alusta, kun Ajax (Asynchronous Javascript and XML) otettiin käyttöön. Ajaxin myötä oli mahdollista tehdä asynkronisia kyselyitä palvelimen ja selaimen välillä. Asiakkaan ja palvelimen välisen asynkronisen liikenteen erottelu itse web-liikenteestä mahdollisti käyttökokemuksen, joka oli samantyylistä kuin työpöytäohjelmissa. Käytännössä koko sivua ei enää tarvinnut päivittää, jos halusi tehdä jonkin muutoksen sivulle. Selainten rajoitteet alkoivat tulla vastaan, kun Ajaxia alettiin käyttämään [59]. Selaimilla ei ollut pääsyä kaikkiin tietokoneen resursseihin, kuten esimerkiksi koneen kiintolevylle, tai koneeseen liitettyihin lisälaitteisiin.

Navigointi ei enää ollut pelkkiä linkkejä sivulla. Sivuille pystyi rakentamaan työpöytämäisiä toimintoja; esimerkiksi valikoita ja painikkeita toimintoineen. Voitiin ladata jotain haluttua sisältöä tiettyyn sivun osaan läpinäkyvästi niin, että taustalla lähetettiin kysely palvelimelle. Palvelin palautti kyselyä vastaavan sisällön ja sitten saatu sisältö voitiin käsitellä. Luvussa 2.1 esiteltiin käytännön esimerkki navigoinnista www-sivulla. Esimerkissä siirryttiin etusivulta yhteystiedot sivulle linkkiä klikkaamalla. Web oli kehittynyt niin, että etusivulta ei ollut enää pakko tehdä esimerkin mukaista linkkiä yhteystiedot sivulle, vaan sen sijaan oli mahdollista esimerkiksi asettaa etusivulle painike *näytä yhteystiedot*. Painiketta klikkaamalla haettiin Javascriptiä käyttäen palvelimelta XMLHttpRequest-pyynnöllä yhteystiedot. Yhteystietojen saavuttua, ne asetettiin näkyviin sivulle ilman erillistä sivupäivitystä.

Ajaxin ja työpöytäohjelmien kaltaisten web-sovellusten suosion myötä tulivat esille RIA:t (Rich Internet Applications). RIA-alustat toimivat niin, että ne tuovat työpöytäohjelmien teknologiaa suoraan selaimeen eri koodikielillä ja ympäristöillä. Alustoja ovat esimerkiksi Adobe AIR [2], Google Web Toolkit [34], Microsoft Silverlight [58] ja JavaFX [66]. Alustat toimivat siten, että sivustoihin upotetaan sisältöä, jotka selainten lisäosilla toimivat asiakkaan selaimessa [59]. RIA-alusta ei ollut sisäänrakennettu selaimeen ja se vaati lisäosan asentamisen sekä päivittämisen erikseen. Tämä karkotti käyttäjiä, koska lisäosan asentaminen saattoi tuntua turhauttavalta tai hankalalta ja joissakin tapauksissa lisäosia ei sallita asentaa tietoturvaan vedoten.

Lisäosien saatavuutta ei voida varmistaa jokaiselle käyttöjärjestelmälle ja tällöin joudutaan tekemään kyseinen RIA-alusta -lisäosa jokaiselle käyttöjärjestelmälle erikseen. RIA-tekniikat eivät ole varsinaisia web-tekniikoita, koska ne korvaavat itse

web-tekniikat, kuten Javascript-kielen, omilla ohjelmointikielillä ja API-rajapinnoilla. RIA-järjestelmät tarjoavat paljon hyötyä sovelluskehittäjälle juuri API (Application Programming Interface) -rajapintojen myötä [59]. HTML5 on ratkaisu tähän ongelmaan, koska se on tuleva standardi ja se toteutetaan jokaisessa selaimessa kiinteästi käyttöjärjestelmästä riippumatta. Sovelluksia varten HTML5:n toteutetaan määrätyt API-rajapinnat. Tietoturvan kannalta HTML5 on myös parempi, koska pääsy esimerkiksi käyttäjän tietokoneelle tiedostotasolle on tarkkaan rajattu ja määritelty.

2010 -luvun puolivälissä mobiilisovelluskehittäjät alkoivat kiinnostua webistä. Mobiililaitteissa oli paljon eroavaisuuksia: näytön koko, suorituskyky ja Internet-yhteyden nopeus [59]. Mobiililaitteet eroavat toisistaan myös nykypäivänä, mutta yhteisiä ominaisuuksia on aikaisempaa enemmän. Internet-yhteydet toimivat 3G:llä ja uusimmat 4G:llä, jotka mahdollistavat nopeat ja kiinteät verkkoyhteydet edulliseen hintaan. Laitteiden eroavaisuuksista johtuen mobiiliwebin alkuvaiheessa tehtiin epäonnistuneita tuotoksia, kuten WAP (Wireless Application Protocol). Yksi mobiili-web-teknologia oli web widget, joka tuli 2010 -luvun lopulla. Web widget on pieneksi pakattu sovellus, jonka voi asentaa mobiililaitteeseen. Widgetin pakkaus ja asennusformaatit ovat ennalta määrättyjä. Web widget eroaa HTML5-sovelluksesta siinä, että se pitää erikseen asentaa [59].

2.3 Kolmas aikakausi: Web sovellusympäristönä

Mobiililaitteisiin tehdään natiiveja sovelluksia ja niitä myyvät valmistajat sovelluskaupoissaan. HTML5 mahdollistaa web-sovellusten tekemisen ilman valmistajien rajoituksia tai valvontaa. 2010-luvulla natiivien mobiilisovellusten sekä HTML5-web-sovellusten kilpailu kasvaa ja sovelluskehityksen tulevaisuus määräytyy kilpailun voittajan mukaan [59]. Webissä on kuitenkin puutteita, kuten verkon viiveet ja suorituskyky raskaammissa ohjelmissa ja peleissä. Javascript oli kuitenkin vielä vuonna 2012 suorituskyvyltään paljon heikompi kuin vastaava C++/C -koodi [90]. Web ei välttämättä tule lähitulevaisuudessa kokonaan korvaamaan natiiveja sovelluksia. Tietokoneissa ja mobiililaitteissa on paljon lisälaitteita ja sensoreita, joihin ei vielä ole rajapintoja suoraan HTML5:ssä. Tarvitaan myös natiiveja sovelluksia, jotka tarjoavat toimintoja juuri näihin tarpeisiin.

Mikkonen ja Taivalsaari [59] pitävät todennäköisenä, että web-sovellukset voittavat kilpailun pääsovellusalustana natiiveista sovelluksista. Tähän on monia syitä. Ensinnäkin web-sovellusta ei tarvitse tehdä jokaiselle alustalle uudestaan. Lisäksi

web-sovellukset eivät tarvitse erillistä asennusta sekä web-sovellusten päivitys tapahtuu välittömästi maailmanlaajuisesti ja kuka tahansa voi julkaista sovelluksen omalla www-sivullansa fyysisestä sijainnista välittämättä [59]. Mikkosen ja Taivalsaaren mukaan webistä tulee myös pääsovellusalusta muillekin kuin mobiililaitteille. HTML5 on kuitenkin kannettaville laitteille erittäin hyvä sovellusalusta, koska se tarjoaa saman sovelluksen alustariippumattomana [59]. Uskon, että kannettaville laitteille, kuten tablet-tietokoneille ja matkapuhelimille, web on erinomainen alusta ja tulee todennäköisesti olemaan pääalusta sovelluksille.

3 Web-teknologiat ja HTML5

Web-teknologioihin kuuluvat kaikki webissä käytetyt tekniikat. Niihin kuuluvat mm: Javascript, HTML-määritykset ja niiden kärjessä HTML5-määritykset. HTML-sivun merkintä- ja koodirakenne muuttuvat HTML5:ssä [52, s. 24–28]. HTML5 tuo tekniikoita, joilla saadaan aikaiseksi monenlaisia uusia toimintoja.

Uusia HTML5-tekniikoita ovat mm. *WebGL*, jolla voidaan rakentaa 3D-grafiikkaa ja esimerkiksi pelejä toimimaan suoraan selaimessa. Kanvaasi-elementti (canvas) on myös yksi uusi lisäominaisuus HTML5:ssä. Kanvaasi-elementti on HTML-elementti, johon voidaan piirtää, tuoda kuvia ja luoda grafiikkaa. Piirtäminen tehdään Javascriptia käyttäen. Web-sivulle voidaan siis piirtää sisältöä suoraan koodilla, eikä sitä tarvitse ladata erillisestä kuvatiedostosta. [52, s. 24–28]

Käyttäjän koneelle voidaan lisäksi tallentaa hallitusti tietoa käyttäen HTML5:n yhteydettömän tilan määrityksiä. Tietoa voidaan tallentaa suoraan selaimen tukemiin web-varastoihin. Web-varastoja on erilaisia ja selaintuki ei ole standardi. Käytännössä suurinta osaa HTML5-ominaisuuksista hallitaan Javascriptilla. [52, s. 268–270]

CSS3 on uusi tyylistandardi, jonka ominaisuuksia on toteutettu uudemmissa selaimissa. Vaikka CSS määrittelee lähinnä sivun ulkoasua ja tyylejä, CSS3 luetaan joskus HTML5-käsitteen alle. CSS3:sta voidaan käyttää myös vanhemmissa HTML-sivuissa. CSS3 tuo lisäksi mm. muunnokset, joita käyttämällä voidaan toteuttaa animointeja web-sivulla. HTML5 on kuitenkin vielä kesken ja selainvalmistajat ovat toteuttaneet itse valitsemiaan ominaisuuksiaan. Tässä luvussa käydään läpi web-sovelluskehityksessä käytettäviä tekniikoita. [52, s. 285]

3.1 Web-sivu

Yksinkertaisimmillaan web-sivu koostuu HTML-määritysten mukaisesta sisällöstä ja elementeistä, johon kuuluvat yläosa (head) ja sisältö (body) -osiot. Yläosaan määritellään sivun otsikko (title), joka näkyy selaimen yläpalkissa. Yläosaan asetetaan myös CSS-tyylit ja Javascript-koodit tai viittaukset tiedostoihin, joista tyylit ja Javascript haetaan ja käytetään sivulla. Sisältöosuuteen asetetaan HTML-määritysten

mukaisia elementtejä XML-muotoilua käyttäen. Esimerkiksi, jos sivulle halutaan lisätä kuva näkyviin, sille luodaan kuva (img) elementti, johon asetetaan määrite lähde (src), jossa on tekstimuodossa osoite, mistä kuva löytyy ja sieltä se voidaan esittää sivulla. Käytännössä HTML-koodi kuvan lisäämiselle olisi **. [57, s. 16–19]

Web-sovelluksissa voidaan käyttää näitä elementtejä luodessa käyttöliittymää sovellukselle. Rakenne ja ulkoasu määritellään CSS-tyylitiedostossa, joka on linkitetty sivun yläosaan tai suoraan sivun yläosassa tyylielementin (style element) sisälle. Lisättäville elementeille voidaan asettaa jokin tyyliluokka, jonka mukaan ulkoasu luodaan. Tämä tyyliluokka on kirjoitettu CSS-tiedostoon ja siihen on lisätty jotain CSS-tyylimääritysten mukaisia tyylejä. [57, s. 27–33]

HTML5 tuo mukanaan lisää elementtejä, jotka mahdollistavat monenlaista sisältöä sivulle. Selainvalmistajat ohjelmoivat selaimet, kuten esimerkiksi Internet Explorer, Google Chrome, Opera ja Mozilla Firefox toimimaan HTML-määritysten mukaisesti. Web-sivuja tehdessä koodaajan täytyy tehdä HTML-määritysten mukaisia sivuja, joita Internet-selaimet näyttävät. Web-sovelluksia tehdessä koodaaja voi käyttää Javascript-koodikieltä tuodakseen sovellusmaisia ominaisuuksia web-sivustoihin. [57, s. 12–13]

3.2 Javascript

Web-sovelluksissa käytettävä koodikieli on nimeltään Javascript. Flanagan määrittelee Javascriptin kirjassaan *Javascript: the definitive guide* [32] näin: tulkittu ohjelmointikieli, jossa on olio-ohjelmointiominaisuuksia. Javascriptiä käytetään web-sivustoissa ja web-sovelluksissa [32, s. 1–2]. Internetselain tulkitsee Javascriptin yhdessä DOM:n kanssa [32, s. 4]. Javascript suoritetaan käyttäjän selaimessa ja se siirtää prosessorikuormaa palvelimelta käyttäjälle. Nykyään lähes kaikki Internetselaimet tukevat Javascriptia ja sitä käytetäänkin lähes kaikissa web-sivustoissa [90]. Webin käyttäjistä yli 95% käyttää Javascriptia selatessaan Internetiä [90]. HTML5:n monet uudet ominaisuudet käyttävät Javascriptiä.

Käytännössä Javascriptiä käytettäessä web-sivun koodiin lisätään skriptielementti (*<script>*), jonka sisään voidaan kirjoittaa Javascript-koodi [32, s. 4]. Koodi voidaan myös kirjoittaa erilliseen tiedostoon ja sivulle voidaan sen jälkeen lisätä viittaus kyseiseen tiedostoon. Työpöytäohjelmissa on tyypillisesti *Tiedosto* ja *Muokkaa* -valikko. Työpöytäohjelmassa käyttäjä klikkaa *Tiedosto*, josta aukeaa valikko, jossa on lisäva-

lintoja kuten *Avaa*, *Tulosta* ja *Sulje*. Valintaa klikkaamalla tapahtuu siihen ohjelmoitu toiminto. Tällaisen valikon voi tehdä web-sivulle käyttäen esimerkiksi Javascriptiä. Tämä tapahtuu esimerkiksi siten, että sivulle lisätään jokin HTML-määritysten mukainen rakenne-elementti, jonka sisään kirjoitetaan teksti *Tiedosto* ja lisätään elementille määrite *onclick="JokinJavascriptFunktio()"*. Määrite tarkoittaa sitä, että kun käyttäjä klikkaa hiirellä tekstiä *Tiedosto*, niin suoritetaan sen *onclick*-määritteen mukainen Javascript koodi. Sivustolle lisätään sitten toinen rakenne-elementti, johon kirjoitetaan *Tiedosto*-tekstistä aukeavan valikon sisältö. Tämä toinen elementti piilotetaan käyttäjän näkyvistä jollakin CSS-tyylimääritteellä esimerkiksi *visibility:hidden*. Sen jälkeen ei tarvitse, kuin kirjoittaa Javascript-koodi, joka käyttäjän klikatessa, asettaa valikko-elementin CSS-tyylimääritteen *visibility:visible*-tilaan. [32]

3.3 CSS3

CSS:ää (cascading style sheet) käytetään web-sivuilla määrittelemään sivuston ulkoasua, fontteja ja tyylejä. CSS:n uusin versio CSS3 sisältää lisäyksiä, kuten efektit ja muunnokset (transform). Efektejä ovat esimerkiksi pudotusvarjo (drop-shadow), pyöristetyt kulmat elementeissä ja läpinäkyvyys. Efektien lisäksi CSS3:n esittelee muunnokset, joilla voidaan muuttaa elementin muotoa, sijaintia, kääntää sitä ja muuttaa elementin kokoa. Näitä toimintoja yhdistelemällä web-sivusta saadaan houkuttelevampi ja kauniimpi sekä mahdollisesti mielekkäämpi käyttää. [48, s. 123]

Pelkästään CSS-tyyleillä on mahdollista luoda natiivien sovellusten kaltaisia toimintoja. CSS:llä voidaan määrätä esimerkiksi mitä tapahtuu HTML-elementille, kun hiiri tulee sen päälle web-sivulla [48, s. 123]. Elementille voidaan määrätä esimerkiksi, että sen taustaväri muuttuu punaiseksi ja elementin korkeus kasvaa kaksinkertaiseksi, kun hiiri tulee elementin päälle. Lisäksi voidaan määrätä esimerkiksi, että HTML-elementin kaikki lapsielementit vaihtavat fonttia ja tekstin väri vaihtuu valkoiseksi. Mahdollisuudet ovat laajat käyttöliittymän suunnittelun osalta.

CSS3 tuo lisäksi myös siirtymät (transitions), joilla voidaan luoda animointeja sivulle sekä tehdä erilaisista liikkeistä sulavampia. Siirtymissä voidaan pelkkää CSS-määritettä muuttamalla saada käyttöliittymässä aikaan erilaisia liukumia ja animointeja, joita on yleensä natiivien sovellusten käyttöliittymissä [52, s. 285]. CSS-määritettä muuttamalla selain animoi siirtymän alkutilasta uuteen muutettuun arvoon. Käytännössä tämä voi olla jokin numeerinen määrite, kuten jonkin HTML-elementin pituus, sijainti tai väri [30, s. 1].

CSS3 on HTML5:n tavoin vielä keskeneräinen. Internetselaimien tekijät ovat kuitenkin toteuttaneet CSS3:n ominaisuuksia uusimmissa selainversioissaan ja niitä voidaan käyttää suurelta osin. CSS3 yhdistetään joskus HTML5:n, mutta se on kehittynyt kuitenkin erillään HTML:stä. CSS3:ssa on samanlaisia määritteitä kuin esimerkiksi HTML5:n kanvaasi-elementissä. Yhtenäisiä asioita kanvaasin kanssa on esimerkiksi CSS3:n värimäärittely, joka on samanlainen molemmissa. CSS3:ssa voidaan värimäärittelyssä asettaa läpinäkyvyyttä kuvaava arvo, jolla määritellään värin läpinäkyvyys desimaaliarvolla nollan ja yhden välillä. [52, s. 285]

3.4 WebGL

WebGL on alustariippumaton laitteistokiihdytteinen 3D-grafiikka API, jonka on kehittänyt Mozilla (http://www.mozilla.org) sekä Khronos Group (http://www.khronos.org) ja joukko muita yrityksiä kuten Apple, Google ja Opera. WebGL tuo 3D-grafiikan natiivisti webiin niin, että se toimii suoraan web-selaimilla ilman lisäosien asentamista [59]. 3D-sovellusten puute on ollut viimeinen asia, mihin sovelluskehittäjät ovat voineet vedota, kun verrataan webbiä sovellusalustana natiivien sovellusten korvaajana.

WebGL pohjautuu OpenGL-rajapintaan ja se käyttää GLSL-kieltä (OpenGL Shading Language). WebGL toimii HTML5:n kanvaasielementissä (canvas element) ja siihen pääsee käsiksi DOM:n (Document Object Model) kautta suoraan Javascriptiä käyttämällä. Tämä tarkoittaa sitä, että koko ympäristö on dynaaminen ja sovelluksia pystytään dynaamisesti muuttamaan lennosta. [59] 3D-grafiikkaa on aiemmin pystytty näyttämään vain määrätyissä selaimissa, käyttäen erikseen asennettuja lisäosia. 3D-rajapinta on suuri askel myös peliteollisuudelle.

3.5 HTML5

HTML-kielen uusin versio on HTML5, jota ei ole vielä määritelty lopullisesti ja sen ominaisuudet voivat muuttua. HTML5 tuo HTML-kieleen paljon uusia ominaisuuksia, jotka mahdollistavat näyttävämmän sisällön ja monipuolisemmat mahdollisuudet tehdä web-sovelluksia ja -sivustoja. HTML5 mahdollistaa muun muassa piirtoalustat, videoiden ja äänen esittämisen suoraan selaimessa, paikkatiedon haun, selaimen muistin, lomakkeiden uusia kenttiä ja välimuistin käytön, joka mahdollistaa sivustojen käytön yhteydettömässä tilassa. HTML5 rakentuu vanhemman

HTML4:n päälle ja on siis taaksepäin yhteensopiva. HTML5-web-sovellusten toiminnallisuus on pääosin toteutettu käyttäen Javascript-kieltä. [52, s. 12–15]

Tavallinen Internetin käyttäjä pääsee käyttämään HTML5:n tuomia uusia ominaisuuksia päivittämällä tarvittaessa selaimen uusimpaan versioon ja menemällä jollekin HTML5:tä hyödyntävälle sivustolle. Tämä kaikki näyttää helpolta ja yksinkertaiselta. HTML5:n määritys ei ole vielä valmis, mutta selaimet ovat toteuttaneet silti sen ominaisuuksia ja niitä käytetään osalla web-sivustoista. Muutoksia on tullut myös koodaajan näkökulmasta katsottuna. HTML-koodiin on tullut rakennetta selkeyttäviä muutoksia. HTML5-web-sovelluksia tehdessä halutaan mahdollisesti tavoitella natiivisovelluksen käytettävyyttä ja tähän päästään rakentamalla HTML5-käyttöliittymä sen mukaisesti.

HTML5:ssa kaikki ominaisuudet eivät ole vielä tuettuja kaikissa selaimissa ja esimerkiksi rajapinta tietokoneeseen ja sen lisälaitteisiin tai mobiililaitteen sensoreihin on määrittelyasteella. Standardisointi saattaa kestää sen takia, että sitä määrittelee useampi taho. Mikkosen ja Taivalsaaren mielestä tarvitaan vielä yksi kierros standardisointia 2010-luvun loppupuolella, jotta saataisiin valmiimpi web-sovellusalusta. He käyttävät siitä nimitystä HTML5+, joka tarkoittaa HTML5:n seuraajaa. [59]

3.5.1 Katsaus rakenteeseen ja ominaisuuksiin

HTML-sivun rakenne koostuu yleensä ryhmistä tai osista, joihin sivu jaetaan. Jon Duckett kertoo kirjassaan *HTML & CSS Design and Build Websites* [27] HTML:n ja CSS:n käytöstä sivustojen luomisessa ja perinteisestä HTML-sivusta sekä siitä, minkälainen uusi HTML5:n rakenne on. HTML-sivuissa ennen HTML5:stä käytettiin *div*-elementtejä, joilla sivu rakennettiin. Sivu ryhmiteltiin *div*-elementeillä mm. otsikkoon, navigointiin, sisältöön, sivupalkkeihin ja alatunnisteeseen [27, s. 431]. *Div*-elementille asetettiin *id*, joka kuvasti käyttötarkoitusta, tai jos sivulla oli toistuva elementti, voitiin käyttää CSS-luokkaa kuvaamaan sisältöä. HTML-lähdekoodi saattoi olla sekavalukuista, koska siellä oli paljon *div*-elementtejä.

HTML5 tuo sivulle uusia elementtejä, joilla voidaan ryhmitellä sivu uudella tavalla. Aiemmin kuvatuille *div*-elementeille on korvaavia elementtejä. Otsikkoa ei tarvitse kuvata enää pelkästään id:llä <*div id="header"*> vaan käytössä on <*header*>, joka tarkoittaa sivun yläosaa. Navigointiosassa voidaan HTML5:n määritysten mukaan käyttää samaan tyyliin <*div id="nav"*> sijasta <*nav*>-elementtiä. Samanlaisia uudistuksia on muitakin ja koska HTML5 ei ole vielä valmis standardi, niitä voi tulla vielä lisää. [27, s. 431]

HTML5 ei tarkoita pelkästään HTML:n uusinta versiota, vaan sen alle luetaan erilaisia uusia web-tekniikoita, kuten grafiikan ja liikkuvan kuvan esittäminen selain- ja lisäosariippumattomasti. Lisäksi on esimerkiksi toiminnallisuuksiin päivityksiä ja uusia lomakekenttiä [52, s. 13]. Joskus myös CSS3-tyyliohjekieli asetetaan HTML5 käsitteen alle [52, s. 285]. CSS3:ssa on paljon ominaisuuksia, joita voidaan käyttää HTML5-käyttöliittymän rakentamisessa. Näihin kuuluvat mm. liukuvärit, läpinäkyvyys ja käyttöliittymän käyttökokemukseen vaikuttavat muunnokset.

Muun muassa näitä ominaisuuksia voidaan käyttää HTML5-web-sovellusten käyttöliittymien suunnittelussa ja niiden avulla natiivien sovellusten ja web-sovellusten raja voi tulevaisuudessa häilyä. Erityisesti web-sovelluksiin sopivia uusia ominaisuuksia on olemassa monia [59]:

- Mahdollisuus tehdä yhteydettömiä sovelluksia, eli ei tarvita Internet-yhteyttä, jotta sovellus toimisi.
- Paikallinen tallennus, jolla voidaan tallentaa tietoa käyttäjän tietokoneelle. Käytännössä tieto koostuu nimistä ja niiden arvoista.
- *Canvas API*, joka tarjoaa mahdollisuuden piirtää 2D-grafiikkaa, kuten kuvia, muotoja, tekstiä ja interaktiivista grafiikkaa suoraan web-sivulle.
- Sisäänrakennettu videon ja äänen toisto ilman erillisiä selainlisäosia, niille luoduilla <*video*> ja <*audio*> HTML elementeillä.
- Asynkroninen skriptien lataaminen uudella *async* -attribuutilla, jonka voi asettaa <*script*> elementin sisään.
- Elementtien raahauksen ja pudottamisen tuki, jossa käyttäjä voi käytännössä raahata haluamaansa sisältöä sivulla.
- Ohjelmoitava pikavalikko (*context menu*).
- Viestien lähetys ja vastaanotto eri palvelimien välillä. Javascriptillä pystytään lähettämään viestejä palvelimelta toiselle.
- Muokattavat web-sivut, jossa käyttäjä pystyy muokkaamaan suoraan sivuston sisältöä.

Näiden lisäksi HTML5 tarjoaa muita pienempiä lisäyksiä, kuten esimerkiksi selainhistorian hallinta, uudet lomakekentät, uudet MIME-tyypit ja mikrodata. Uusien ominaisuuksien myötä web-sovellusten tekeminen helpottuu huomattavasti.

3.5.2 Web-sovellusten yhteydetön käyttö

Aiemmin, kun Internet-yhteyksiä käytettiin hitailla modeemeilla ja niiden käyttö oli kallista, saatettiin yhdistää Internetiin ja sitten mennä halutulle sivustolle ja ladata se yhteydettömään tilaan (offline). Tällöin Internet-yhteyttä ei tarvinnut pitää päällä eikä maksua kertynyt. Nykyään kaikilla on kiinteä kuukausihintainen laajakaista eikä edellä mainittua toimintaa enää tarvitse tehdä. [52, s. 268] Korpelan mukaan nopeus on aina suhteellista, vaikka on nopeita laajakaistayhteyksiä. Internet-sivujen palvelut lisääntyvät koko ajan ja niissä on paljon kuvia, videoita, ohjelmakoodia jne. Tällöin sivustot toimivat vielä nopeammin, jos sisältö on ladattu yhteydettömään tilaan, varsinkin mobiililaitteilla, joissa voi olla joskus hitaat yhteydet myös sijainnista johtuen [52, s. 268]. Vaikka olisit palveluntarjoajan mobiiliverkon kantavuusalueella, saattaa silti löytyä katvealueita, joissa yhteys ei toimi halutulla tavalla. Matkapuhelinverkoissa verkon kaista jaetaan kaikkien käyttäjien kesken ja verkkosolujen kesken. Tällöin suuren kaupungin keskustassa voi matkapuhelinverkko olla hyvinkin hidas.

Sivuston vasteaika on myös olennainen käytettävyyden kannalta. Jos voidaan valita muutaman sekunnin viiveen sijasta käytännössä välitön lataus, niin se on paljon parempi ratkaisu. Sivuston saatavuus on aina varma yhteydettömällä käytöllä, vaikka itse sivuston palvelin olisi kaatunut tai huollossa. Käyttäjä saattaa haluta käyttää sovellusta tai web-sivua rauhassa ilman häiriöitä esimerkiksi saapuvista sähköposteista, tai käyttäjä haluaa käyttää sovellusta tietoturvallisesti niin, että data ei liiku esimerkiksi avoimessa WLAN-verkossa. [52, s. 268]

HTML5-määrittelyssä on kaksi ominaisuutta, joita voidaan käyttää yhteydettömien sovellusten ja web-sivujen tekemiseen. Nämä ovat Web-varasto (Web storage) ja yhteydetön sovellus -ominaisuus (offline web application feature). Web-varasto laajentaa vanhan HTTP-evästeiden (cookie) käyttöä. Pääasiallinen käyttö evästeillä on tilojen hallintamekanismi. Lisäksi niitä voidaan käyttää tallentamaan tietoa käyttäjän selaimeen tietyin kokorajoituksin. Evästeillä ei kuitenkaan saada tallennettua kuin nimiä ja niiden arvoja, eikä niihin voi esimerkiksi tallentaa HTML-tiedostoa. HTML5-webvarastoa käytettäessä käyttäjän tietokoneella on paljon tilaa tallentaa lisää dataa. [21]

Yhteydetöntä web-sovellusta käytettäessä voidaan tallentaa kokonaisia HTML-sivustoja käyttäjän päätelaitteelle käyttäen HTML5-määrityksiä. Ominaisuus toimii siten, että ylläpidetään ilmoitustiedostoa (manifest), joka jaetaan HTML-tiedostojen mukana. Se kertoo, mitä tiedostoja ja HTML-objekteja tallennetaan yhteydettömään

tilaan käyttäjän tietokoneelle. [21] Välimuisti-ilmoitus (cache manifest) on tiedosto, josta selain lukee, mitä kaikkea kuuluu HTML-dokumentin yhteyteen ja mitä ladataan yhteydettömään tilaan käyttäjän tietokoneelle. Aineisto latautuu käyttäjän sovellusvälimuistiin, jonka tekninen toteutus riippuu selaimesta.

Käytännössä ilmoitustiedostoon voidaan asettaa tallentumaan kokonainen websivusto sisältöineen, jolloin käyttäjän koneelle ladataan sisältö ja näin sivustoa voidaan käyttää myös yhteydettömässä tilassa. Tämä ei yksinään toimi hyvin websovelluksissa, joissa tallennetaan ja haetaan tietoa tietokannasta, koska data voi olla esimerkiksi riippuvainen muista tauluista tietokannassa. Tässä tapauksessa tieto pitäisi tallentaa käyttäjän koneelle web-varastoon. Kun ollaan taas yhteydellisessä tilassa, data pitäisi synkronoida palvelimella olevan tietokannan kanssa.

Välimuisti-ilmoitusta ei ole hyvä käyttää normaalissa web-sivustossa, koska ilmoitus ohittaa normaalin välimuistikäsittelyn. Tämä sotkee normaalin välimuistin käytön ja HTTP-otsakkeiden välimuistisäännöt eivät päde. Välimuisti-ilmoituksen mukainen välimuistin päivittäminen tapahtuu oman erikoissäännön mukaan, jolloin ladataan koko sisältö kerralla ja se soveltuukin paremmin web-sovellusten käyttöön [52, s. 274]. Yhteydetön ominaisuus on erittäin hyvä web-sovelluksissa käytettynä. Esimerkkinä voisi olla näköislehti sanomalehdestä, jossa voidaan ladata uusin lehti heti, kun käyttäjä on yhteydellisessä tilassa tietokoneellaan tai mobiililaitteellaan. Sisältö ei muutu kuin kerran päivän aikana, kun uusi lehti ilmestyy. Näköislehden tapauksessa edellä mainittu toiminto on toimiva, mutta jos tehdään enemmän sovellusmaisempaa ja rikkaampaa HTML5-lehteä, voi olla mielekkäämpää tallentaa tiedot hallitummin.

Yoshifumi Chisaki, Royyana M. Ijtihadie ja Tsuyoshi Usagawa [21] esittelevät prototyypin yhteydettömää tilaa hyödyntävästä sovelluksesta käytettäväksi opiskeluympäristössä Moodle LMS (Learnin Management System) -lisäosana. Järjestelmässä kurssille ilmoittautuneet oppilaat voivat vastata ja tehdä tehtäviä matkapuhelimella, joko käyttäen Internet-yhteyttä tai tehden sen yhteydettömässä tilassa. Oppilaat voivat synkronisoida tehtävänsä uusimpaan versioon aina koulun verkossa ja viedä sitten tehtävän kotiin. [21] Kurssin suorittamisen kannalta oppilailla ei siis tarvitse olla Internet-yhteyttä, vaan he voivat suorittaa kurssin myös ilman sitä.

Käytännössä ohjelma toimii Moodle:ssa siten, että siellä on painike, jossa lukee *Vie kotiin*, jota klikkaamalla oppilas saa yksilöllisen ID-numeron. ID-numerolla oppilas voi ladata tehtävän yhteydettömään tilaan matkapuhelimeensa. Web-sovelluksessa oppilas kirjautuu tunnuksin ja syöttää saamansa ID-numeron. Oppilas voi teh-

dä tehtävää osissa siten, että jättää sen välillä kesken, jolloin tila tallentuu. Tehtävillä on jokin aikaraja, jota ennen heidän täytyy saada tehtävä tallennettua valmiiksi. Aikaraja on myös tallennettu web-sovellukseen ja tehtävää ei voi tallentaa enää aikarajan umpeuduttua. [21] Tämä on hyvä esimerkki HTML5-sovelluksesta, joka toimii kaikilla HTML5:stä tukevilla laitteilla. Sovellus käyttää yhteydettömän tilan tallennusta ja se toimii itsenäisesti käyttäjän laitteessa ilman Internet-yhteyttä. Web-sovellus toimisi samalla lailla myös esimerkiksi kannettavassa tietokoneessa.

3.5.3 HTML5 vs. Natiivi koodi

Andre Charland ja Brian LeRoux kirjoittavat artikkelissaan *Mobile Application Development: Web vs. Native* web-sovellusten kilpailukyvystä natiiveja sovelluksia vastaan ja webin mahdollisuuksista korvata natiivit sovellukset [20]. Mobiililaitteelle tehty natiivi sovellus alkaa alustalle sopivasta sovelluskoodista. Koodikieliä on monia ja tyypillisiä kieliä ovat C, jota käytetään iPhonessa sekä Java, jota käytetään Androidissa. Windows puhelimet käyttävät .NET C# kieltä [20]. Kieliä on monia ja niiden lisäksi alustoilla voi olla useita eri SDK:ta (Software Development Kit). Jos sovellus halutaan toimimaan useimmilla käyttäjillä, joudutaan sovellus uudelleenohjelmoimaan kaikilla kielillä. Web-sovellusta ei tarvitse tehdä kuin kerran ja koska kaikissa laitteissa on www-selain, toimii se myös kaikissa laitteissa yhdellä koodilla.

Web-ympäristö toimii omassa hiekkalaatikossaan, jossa on estetty alemman tason API:n pääsy sensoreihin ja ominaisuuksiin, johon natiivilla koodilla päästään. Web-sovelluksia varten on olemassa PhoneGap hakkerointitekniikka, jossa käynnistetään laitteessa selainінstanssi ja sen kautta päästään käsiksi matkapuhelimen natiiviin koodiin Javascript rajapinnan kautta [20]. Tämä ominaisuus lisättiin iPhone OS SDK:n ja myöhemmin myös muille mobiilialustoille [20]. Natiivi koodi on käännetty alustalleen ja se on suorituskykyisempää kuin selaimessa suoritettava koodi. Käyttöliittymä luodaan eri tavalla natiivissa koodissa kuin HTML:ssä. HTML:ssä käyttöliittymä luodaan kuvista, koodista ja tyylitiedostoista, kun taas natiivissa koodissa käyttöliittymä piirretään näyttöön käyttäen API-rajapintoja [20]. HTML5 tuo kuitenkin natiivin koodin joitakin ominaisuuksia suoraan selaimeen, kuten paikannuksen ja tallennuksen levylle [20].

Natiiveissa alustoissa ovat valmiina käyttöliittymäkomponentit ja tietynlaiset käyttökokemukset, mutta yhdelläkään alustalla ei ole samoja tai edes samankaltaisia käyttöliittymiä ja paradigmoja [20]. Käyttöliittymän muokattavuus on web-

sovellusten yksi vahvuus. Esimerkiksi painikkeen lisääminen sovellukseen tapahtuu natiivissa koodissa lisäämällä painikeobjekti sovellukseen, jolloin painike on ulkoisesti ja käytettävyydeltään alustan määräämä eikä sen ulkoasua voi muuttaa. Web-sovelluksissa voidaan tehdä ulkoasultaan ja toiminnallisuudeltaan minkälaisia painikkeita tahansa.

Web-sovelluksissa voidaan käyttöliittymä tehdä natiivista sovelluksista poiketen suurilta osin samanlaiseksi eri alustoille, mutta sisäänrakennettuja kontrolleja on rajoitetusti. Web-sovelluksissa käyttöliittymä ei ole täysin samanlainen, koska eri selaimissa voi olla pieniä eroja. Mobiililaitteissa on käyttöjärjestelmässä yleisesti mahdollisuus mennä takaisin edelliseen näkymään tai avata kontekstivalikko, joita voidaan käyttää hyväksi natiiveissa sovelluksissa. Näitä ominaisuuksia on vielä vaikea toteuttaa selaimissa, koska esimerkiksi *takaisin*-painike sulkee selaimen ja menee edelliseen käytettyyn sovellukseen [20].

Kaikissa sovelluksissa suorituskyky on yksi tärkeä asia käyttökokemuksessa. Web-sovelluksissa suorituskykyyn vaikuttavat erityisesti liikkuvan datan määrä sekä koodin suoritusaika. Web-sovelluksissa käytetyn Javascriptin suoritusaika on pidempi kuin natiivin koodin [20]. Javascript kehittyy kuitenkin koko ajan ja selainvalmistajat kilpailevat sen nopeudessa. Kilpailu edistää koodin suorituskyvyn kasvua ja se voi tulevaisuudessa lähestyä natiivin koodin suorituskykyä. Natiivin koodin tekeminen vie ohjelmoijalta enemmän aikaa, koska ohjelmasta täytyy tehdä oma versio jokaiselle alustalle [20]. Natiivi koodi on tehokkaampaa, mutta se vaatii enemmän huoltoa kuin Javascript [20]. Javascriptillä tehty web-sovellus toimii pienin muutoksin jokaisella alustalla ja näin sovelluksella on yksi lähdekoodi, jota on helpompi ylläpitää kuin natiivisovellusten alustakohtaiset koodit [20].

Joissakin käyttöjärjestelmissä on sisäänrakennettu laitekiihdytys selaimille. Esimerkiksi Applen iOS tukee CSS-muunnoksia, jotka saavat aikaan tasaiset siirtymät näkymistä toisiin [20]. CSS-tyyleillä voidaan tehdä erilaisia liukumia ja animointeja, jotka voivat olla raskaita tai raskaampia, kun ne tehdään Javascriptiä käyttäen. Mobiililaitteita on paljon erilaisia, muuttujia ovat esimerkiksi koko, värisyvyys, näytön resoluutio, pikselitarkkuus, kuvasuhde, kosketusnäyttö, fyysinen näppäimistö, mikrofoni, kamera, tietovarasto ja antenni [20]. Laitteiden muuttujien takia sovellukset näyttävät erilaisilta. Web-sovelluksissa on mahdollista tehdä visuaalisempia käyttöliittymiä ja CSS-tyylejä käyttämällä ne ovat mobiililaitteilla myös suorituskykyisiä.

4 Kannettavat laitteet ja käytettävyys

Pöytätietokoneiden suosion jälkeen kannettavista tietokoneista tuli suosittuja mukana kulkevia versioita pöytätietokoneesta. Kannettavaa tietokonetta pystyi kantamaan mukana minne tahansa. Alussa kannettavat tietokoneet olivat kalliita, kunnes LCD (Liquid Crystal Display) teknologia, kiintolevy, kannettavan prosessori ja langattomuus kehittyivät. Kannettavissa tietokoneissa oli kuitenkin rajoitteita kuten se, että niissä täytyi olla näppäimistö ja kosketuslevy tai hiiri käyttöä varten. Kannettavuuden kehitystarpeet ja syöttölaitteiden käytön parannustarpeet kannettavissa laitteissa johtivat tablet-tietokoneen kehittämiseen. [67]

Kosketusnäyttö on erilainen käyttöliittymä kuin perinteinen kotitietokoneen käyttöliittymä. Perinteinen tietokonekäyttöliittymä toimii näytön, hiiren ja näppäimistön yhdistelmällä. Kosketusnäytöllä toimitaan käyttäen sormia tai sille suunniteltua kynää [67]. Sormien käyttö on yhdistävä tekijä kosketusnäytöllisten laitteiden ja erilaisten painettujen paperisten tuotteiden kanssa.

Sanomalehtiä ja kirjoja voidaan käyttää käsillä. Sivuja voidaan vaihtaa käsin ja uutista voi tarkastella katsomalla lähempää tai kääntämällä lehti johonkin tiettyyn asentoon. Kirjat vievät paljon tilaa hyllyissä eikä kirjoja voi kantaa rajattomia määriä mukana. Sanomalehti voidaan tilata kotipostilaatikkoon, mutta useamman päivän matkalla siitä ei ole hyötyä, että sanomalehti on tuotu kotiin. Erilaiset sähköiset laitteet tuovat näihin ongelmiin ratkaisuja. Perinteisen kannettavan tietokoneen avulla voi nykypäivänä selata sanomalehtien näköisversioita ja sähköisiä kirjoja.

Perinteisellä tietokoneella käyttötapa ei ole samanlainen kuin fyysisen kirjan tai lehden lukeminen. Matkapuhelimissa on melko pienet näytöt ja niissä ei välttämättä saavuteta samanlaista käytettävyyttä kuin tablet-laitteilla tai erikseen sähköisiä kirjoja varten suunnitelluilla laitteilla.

Sähköisiä e-kirjoja (eBook) voidaan lukea perinteisten tietokoneiden lisäksi niitä varten tehdyillä lukulaitteilla ja tablet-laitteilla. E-kirjalaitteista poiketen tablet-laitetta voi käyttää myös normaalina tietokoneena. Kannettavilla laitteilla lukukokemus saattaa olla lähempänä painetun paperimateriaalin lukemista. Kannettavaa laitetta voi käännellä ja käyttää eri asennoissa. Sitä voi pidellä ilmassa kuten kirjaa tai lehteä ja sitä voi käyttää sormilla. Kannettavissa laitteissa ei tarvitse käyttää

erillisiä johtoja tai muita lisälaitteita. Laitteeseen voi ladata uuden version lehdestä tai uuden kirjan vaikka jok'ikinen päivä. Sähköiset kirjat ja lehdet eivät vie tilaa hyllyssä, eikä niitä tarvitse kantaa paperinkierrätyspisteille.

4.1 Kannettavat laitteet

Kannettava laite voi olla kevyt, yhdellä kädellä kannettava laite, jota käytetään sormilla. Laitteissa voi olla yleensä kosketusnäyttö ja ehkä joitakin kiinteitä painikkeita, kuten virtapainike tai äänenvoimakkuuden säätö. Sähköisiä e-kirjoja voi lukea niitä varten valmistetuilla laitteilla. Nämä laitteet kuluttavat vähän virtaa ja niissä kestää akku pitkään. Lukulaitteet on suunniteltu kirjan korvaajiksi ja lukukokemusta on viety kirjojen lukukokemuksen suuntaan.

E-kirjojen lukulaitteissa on panostettu luettavuuteen ja siihen, että näytöstä voi lukea myös päivänvalossa. E-kirjalaitteista monipuolisemmat tablet-tietokoneet ovat ehkä lähempänä perinteistä tietokonetta. Tablet-tietokoneilla voi suorittaa normaalin pc-tietokoneen toimintoja kuten selata Internetiä, kuunnella musiikkia ja lukea sähköpostit. Laitteiden kannettavuus ja helppokäyttöisyys saattavat olla niiden tärkeimpiä ominaisuuksia.

4.1.1 E-kirjalaitteet

E-kirja on painetun kirjan sähköinen versio, jota voidaan lukea sille erikseen suunnitellulla kannettavalla laitteella tai tietokoneella [54]. Käytännössä e-kirjalaitteet ovat kannettavia laitteita, joissa on iso näyttö ja kirjakauppasovellus. Kirjakaupasta ostetaan e-kirjat ja ladataan ne laitteeseen. Sähköisen musteen kehittymisen myötä näytöt ovat paperin kaltaisia ja niistä pystyy lukemaan myös auringon valossa. [53]

Michael S. Hart keksi e-kirjan vuonna 1971 ja perusti *Project Gutenberg*:n, joka muuntaa kirjoja sähköiseen muotoon [74]. Hartin mukaan e-kirjasta voi tehdä hakuja, e-kirjoista voi ottaa lainauksia, voi ladata uuden version ja julkaista myös vanhan version, e-kirjoja on yksinkertaista lukea, lainata ja kaikkia näitä voi tehdä yleisillä käytössä olevilla laitteilla ja sovelluksilla sekä niiden yhdistelmillä [77].

Paperikirjoissa oli vielä ominaisuuksia, joita saattoi olla vaikea tuoda sähköisiin kirjoihin. Tällaisia olivat kirjojen tuoksu, edullisuus, pienikokoisuus, kannettavuus ja mahdollisuus aloittaa lukeminen mistä tahansa kohtaa. Lisäksi paperisissa kirjoissa voi kirjoittaa muistiinpanoja sivujen reunoille ja tekstiä pystyy korostamaan

korostustussilla. Paperisissa kirjoissa ei tarvitse käyttää käynnistyspainiketta, akkua ei ole ja kirja ei hajoa tippuessaan. [77]

Paperisia kirjoja voi ostaa melkein mistä tahansa ja ne ovat suhteellisen edullisia. Kirjan lukemista varten ei tarvitse ostaa suhteellisen kallista laitetta. Kirjan voi taittaa ja pakottaa taskuun, se voi kastua ja siihen voi kirjoittaa merkintöjä. Kirjaa voi väännellä ja käännellä ja vaihtaa sivua fyysisesti sormituntumalla. Ensimmäiset e-kirjojen lukulaitteet julkaistiin 1990-luvun alussa [53]. Seuraavat e-kirjojen lukulaitteet ilmestyivät 1990-luvun lopulla [33]. Ne otettiin vastaan hyvin vaikka niissä ilmeni puutteita. 2000-luvun alkupuolella julkaisiin pelkästään e-kirjojen lukemiseen tarkoitetut laitteet [38]. Odotukset olivat suuret, mutta niiden myynti oli vaisua [38].

Vuonna 2000 Microsoft julkaisi lukijan taskutietokoneelle (pocket PC), jossa oli *ClearType* teknologiaa. *ClearType* on tekniikka, jolla tietokoneen tekstin fontit selkeytetään ja pehmennetään helppolukuisemmiksi ja vähemmän silmiä rasittaviksi [73]. Laitteisiin kaivattiin pitkää akunkestoa ja mahdollisuutta lukea kirkkaassa valossa.

Huonoja puolia ensimmäisen sukupolven e-kirjalaitteissa olivat iso koko ja painavuus sekä se, että ne hajosivat helposti [103]. Näyttöjen lukukokemukset olivat paljon huonompia kuin oikeiden kirjojen [38]. Lisäksi laitteiden käytettävyydessä oli puutteita ja navigointi e-kirjan sisällä oli vaikeaa [38]. E-kirjojen lukulaitteet ovat kehittyneet ensimmäisestä sukupolvesta ja niiden rinnalle on julkaistu samankaltaisia laitteita, joita voi käyttää myös muihin tarkoituksiin kuin pääasiallisesti pelkästään sähköisiin kirjoihin. Tällaisia ovat esimerkiksi tablet-tietokoneet: *iPad* [10] ja *Galaxy Tab* [80]. Tablet-tietokoneiden lisäksi myös e-kirjalaitteet ovat kehittyneet ja on julkaistu uusia versioita lukulaitteista [38].

Sähköinen muste (e-ink) kehitettiin vuonna 1997 [44]. Sähköistä mustetta käyttävät näytöt toimivat siten, että näytössä on miljoonia mikroskooppisen pieniä kapseleita. Jokaisessa kapselissa on musta ja valkoinen osa, jossa on positiivinen ja negatiivinen sähkövaraus [44]. Positiivisen tai negatiivisen sähkövarauksen osuessa näytön tiettyyn kohtaan, varauksen mukaan kapselin musta tai valkoinen osa näkyy näytössä [44]. Tällä saadaan piirrettyä tekstiä tarkasti ja virtaa säästävästi näyttöön, joka näkyy kirkkaassa valossa ja käyttäytyy kuin paperi.

Sähköistä mustetta käyttävät näytöt ovat hitaampia kuin perustietokoneissa käytetyt LCD-näytöt. Uusien teknologioiden myötä saattaa olla niin, että sähköistä mustetta käytetään tulevaisuudessa vain perinteisten tekstikirjojen lukemiseen. Kirjojen suojauksesta ja omistuksesta on eriäviä mielipiteitä. Tällä voi olla vaikutusta koko teknologian käyttöön. [38]

Vuonna 2010 e-kirjamarkkinat olivat suuressa kasvussa ja sama jatkuu nyt vuonna 2013 [77]. Seuraavissa kuvissa on e-kirjalaitteita aikajärjestyksessä. Kuvassa 4.1 [56] on *Sony Data Diskman DD8*, joka julkaistiin vuonna 1992 ja kuvassa 4.2 [89] on *SoftBook*, joka julkaistiin 1998. Kuvassa 4.3 [82] on vuonna 2010 julkaistu *Amazon Kindle 3* ja kuvassa 4.4 [4] on *Amazon Kindle Paperwhite*, joka julkaistiin vuonna 2012.

4.1.2 Tablet-tietokoneet

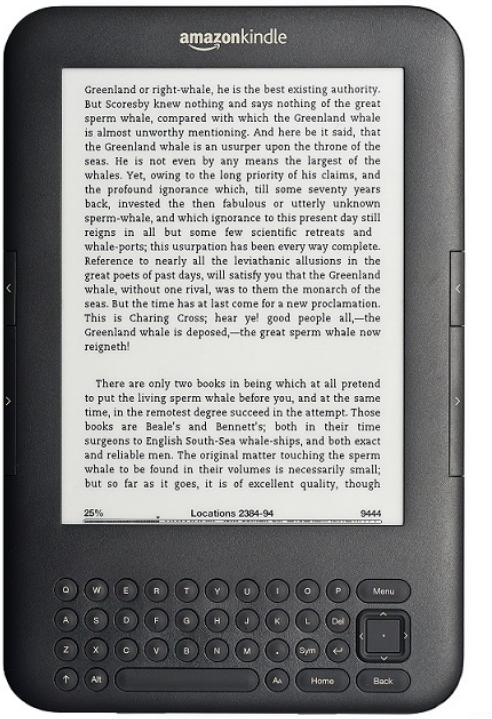
Tablet-tietokone on kompakti kannettava yleistietokone, joka on koteloitu yhteen näyttöpaneelin kanssa [68]. Tabletissa on kosketusnäyttö, jota voi käyttää sitä varten suunnitellulla kynällä tai sormilla [68]. Tablet-tietokoneen ensimmäinen muodollinen esittely tapahtui jo vuonna 1989. Tablet-markkinat olivat kuitenkin niukat aina vuoteen 2002 asti [100]. Microsoft julkaisi tablet-tietokoneen konseptin vuonna 2002, jonka mukaan tabletit ovat pienten kannettavien kokoisia ja yhtä tehokkaita kuin kannettavat tietokoneet. Lisäksi tableteilla voi kirjoittaa käsin tekstiä kosketusnäytöllä [104].

Muutamat ensimmäiset tablet-tietokoneiden käytettävyydestä tehdyt arvostelut olivat erittäin positiivisia. Aiemmin tietokoneet eivät olleet kannettavia ja niitä käytettiin vain tekstinkäsittelyyn. Nykyaikaisilla pöytäkoneilla voidaan tehdä jo paljon muuta ja ne täyttävät tietokoneen käytön tarpeet, mutta niitä ei voi ottaa mukaan. Tähän tarpeeseen kehitettiin kannettava tietokone. Kannettavaa tietokonetta täytyy kuitenkin käyttää hiirellä ja näppäimistöllä sekä tietokone täytyy yleensä sammuttaa aika ajoin. Lisäksi akut eivät kestä riittävän kauan. Tablet-tietokone on ratkaisu näihin ongelmiin. Tablet-tietokone voi olla aina päällä ja on välittömästi käytettävissä tarvittaessa. Tablet-laitetta käytetään yleensä pelkästään kosketusnäytöllä, eikä siinä tarvitse olla muita lisälaitteita. [100]

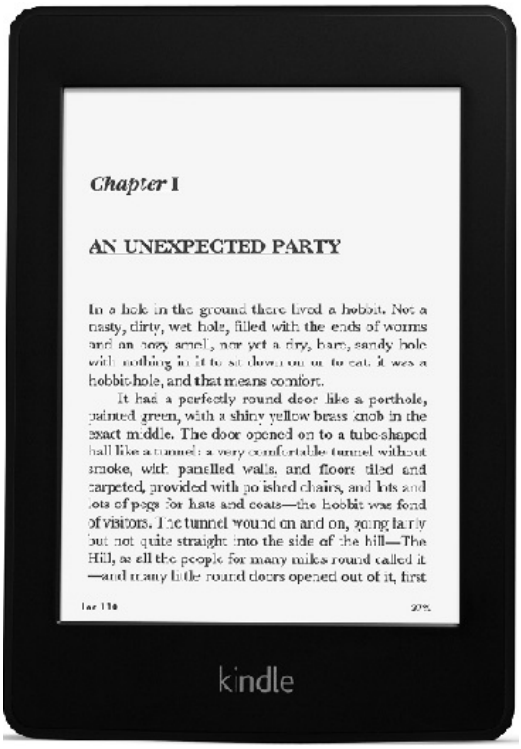
1980-luvun lopulla tablet-tietokoneen konseptin mukaan laitteessa ei ollut näppäimistöä ja tietoa pystyi syöttämään käsin kirjoittamalla. Käsikirjoituksen tunnistus kehittyi ja tarkentui 1990-luvulla. Tekniset ongelmat käsikirjoituksen tunnistamisessa, laitteiden kannettavuudessa, sulautettujen käyttöjärjestelmien kehityksessä ja näytön näkyvyydessä viivästyttivät tablet-tietokoneiden kehitystä ja laajamittaista levitystä. Ei saatu aikaan kaupallista tuotetta, jolla olisi monia käyttöaloja. 2000-luvulla uudet tablet-tietokoneet paranivat ja käyttäjät pystyivät esimerkiksi kirjoittamaan muistiinpanoja kirjoitetun tekstin sekaan. Laitteet pystyivät myös paremmin muuntamaan käsikirjoitusta tekstiksi. Tuolloin tablet-tietokoneiden hinnat vaihtelivat 1.000 dollarista 4.500 dollariin. [67]

Kuva 4.1: Sony Data Diskman DD8 (1992) [56].

Kuva 4.2: SoftBook Press SoftBook (1998) [89].



Kuva 4.3: Amazon Kindle 3 (2010) [82].



Kuva 4.4: Amazon Kindle Paperwhite (2012) [4].

Uudemmissa tableteissa akun käyttöaika parani ja laitteet olivat kevyempiä. Tablet-tietokoneissa oli langaton Internet-yhteys ja niitä käytettiin muistiinpanovälineinä mm. yliopistoissa, rakennustyömailla ja muussa yritystoiminnassa. Vuonna 2005 kehitettiin malli, joka oli yhdistelmä tablet-tietokonetta ja kannettavaa tietokonetta. Laite toimi molempien käyttöjärjestelmillä. Ensimmäiset tutkimukset osoittivat, että tablet-tietokoneille oli markkinoita päivittäiseen tietokoneen käyttöön, ja että tablet-tietokone pystyisi haastamaan kannettavan tietokoneen. [67]

Vuonna 2010 julkaistiin Applen *iPad*. Applen julkaisun jälkeen merkittävimmät laitevalmistajat ovat julkaisseet omia kilpailevia tablet-tietokoneita. *iPadin* ympärillä on käyty paljon keskustelua valmistajien ja sisällöntuottajien keskuudessa. Applen julkaisun jälkeen tablet-laitteiden suosio on kasvanut. Apple on noussut mm. tablet-tietokoneen suosion myötä vuonna 2011 maailman arvokkaimmaksi yritykseksi [84]. Tablet-laitteet ovat kehittyneet ensimmäisistä sähköisten kirjojen lukulaitteista monipuolisiksi helppokäyttöisiksi tietokoneiksi. Tablet-laitteissa oli, kuten monessa e-kirjojen lukulaitteissakin, oma verkkokauppa, josta laitteisiin pystyi asentamaan sekä ilmaisia että maksullisia sovelluksia. [37]

Alla on kuvia tablet-tietokoneista. Kuvassa 4.5 [42] on Intelin 1999 julkaisema tablet-laite *Intel Web Tablet*. Kuvassa 4.6 [102] on vuonna 2003 julkaistu *HP Compaq TC1100* tablet-tietokone, jossa oli *Windows XP Tablet PC* käyttöjärjestelmä. Kuvassa 4.7 [6] on *Apple iPad 3 Retina*, joka julkaistiin vuonna 2012 ja kuvassa 4.8 [81] on 2012 julkaistu *Samsung Galaxy Tab 2*.

4.2 Käytettävyys

Kannettavien tietolaitteiden käyttö on kasvattanut suosiotaan 2000-luvulla. Matkapuhelinten ja tietotekniikan kehittyminen mahdollisti käyttäjille tietotekniikan käyttämisen missä tahansa ja milloin tahansa. Työpöytätietokoneiden suosion jälkeen kannettavista tietokoneista tuli suosittuja mukana kuljetettavia tietokoneita, joissa olivat kuitenkin kaikki tietokoneen toiminnallisuudet. Mobiiliprosessorien, kiintolevyjen, langattomuuden ja LCD näyttöteknologian kehittymisen myötä kannettavista laitteista tuli edullisia ja niitä pystyi ostamaan melkein kuka tahansa. Vuonna 2005 kannettavia tietokoneita myytiin ensi kertaa enemmän kuin pöytätietokoneita. [67]

Kannettavissa tietokoneissa oli pakollista käyttää näppäimistöä ja hiirtä syöttölaitteena. Lisäksi kannettavat laitteet eivät olleet tarpeeksi kevyitä ja kannettavia

Kuva 4.5: Intel Web Tablet (1999) [42].

Kuva 4.6: HP Compaq TC1100 (2003) [102].

Kuva 4.7: Apple iPad 3 Retina (2012) [6].

Kuva 4.8: Samsung Galaxy Tab 2 (2012) [81].

ja samalla helppokäyttöisiä. Kannettavuuden parantamiseksi ja syöttölaitteiden toiminnallisuuden parantamiseksi kehitettiin tablet-tietokoneet. Tablet-tietokoneita oli helpompi käyttää kosketusnäyttöä koskettamalla tai käyttämällä sille suunniteltua kynää. [67]

Pelkästään sähköisiä kirjoja varten kehitettiin laitteita. Laitteisiin pystyi lataamaan useita kirjoja. Näytöistä kehitettiin vähemmän virtaa kuluttavia ja luettavia. Sähköinen muste kehitettiin ja se kulutti aiempaa vähemmän virtaa ja näkyi myös kirkkaissa valo-olosuhteissa. [33]

4.2.1 E-kirjojen käytettävyys

John V. Richardson Jr ja Khalid Mahmood arvioivat artikkelissaan *eBook readers: user satisfaction and usability issues* [77] viiden uusimman e-kirjalukulaitteen käytettävyyttä ja käyttötyytyväisyyttä. Richardson ja Mahmood käyvät ensin läpi vanhempia tutkimuksia ja julkaisuja e-kirjojen käytettävyydestä. Näitä oli yhteensä 13 kappaletta vuodesta 2005 vuoteen 2010 asti. Näistä tutkimuksista selvisi, että e-kirjojen käytettävyydestä ja laitteiden ominaisuuksista oli eriäviä mielipiteitä. E-kirjalaitteita oli pelkästään e-kirjoja varten ja sitten oli laitteita, joilla pystyi tekemään muutakin kuin e-kirjojen lukemista. Näillä laitteilla oli suuria hintaeroja. [77]

E-kirjoissa oli vielä paljon huonoja ominaisuuksia: ei ollut yhtenäistä hinnoittelua, laitteet eivät olleet kierrätettäviä, akunkesto oli yleensä ottaen lyhyt. E-kirjoilla oli rajoitettu saatavuus, laitteissa oli pienet näytöt ja näytöissä oli pienet resoluutiot. Näytöt rasittivat silmiä ja niistä oli vaikea lukea auringonvalossa. Lisäksi e-kirjoissa ei joko ollut kuvia ollenkaan, tai niitä oli vähän. Sovelluksissa oli virheitä (bug), joiden takia laitteet täytyi välillä käynnistää uudelleen. Laitteet eivät kestäneet nesteroiskeita eikä pudottamista. E-kirjat eivät täysin täyttäneet käyttäjien odotuksia. E-kirjalaitteissa oli kuitenkin paljon potentiaalia ja vetovoimaa käyttäjille. [77]

Ruth Wilson testasi tutkimuksessaan *Ebook Readers in Higher Education* [103] viittä ensimmäisen sukupolven e-kirjalaitetta. Ensimmäisen sukupolven laitteet saivat myönteistä palautetta käytettävyydestä, mutta pääongelmakohdat olivat laitteiden koko, helposti hajoavuus ja huono käyttöliittymä [103]. Lisäksi Wilson yhdisti toisen tutkimuksen tuloksia *The revolution starts next week: the findings of two studies considering electronic books* [24], jossa James Dearnley ja Cliff McKnight tutkivat e-kirjalaitteiden käytettävyyttä. Wilson löysi samoja asioita, kuin Dearnley ym. tutkimuksessa. Yleisiä ongelmakohtia käytettävyydessä oli pääongelmien lisäksi liikkuminen e-kirjan sisällä, laitteiden akunkesto, hinta ja näytön kiiltävyys [24].

Laitteita kehitettiin lisää ja monia ensimmäisen sukupolven ongelmista oli jo parannettu. Parannettuja asioita olivat paino, koko, näytön koko ja näytön resoluutio. Laitteissa ei ollut kuitenkaan mitään merkittäviä teknologisia parannuksia. Näihin uudempiin laitteisiin mahtui enemmän e-kirjoja ja niitä oli helpompi käyttää. Niissä oli mm. paremmat näytöt ja niissä oli painikkeita, joilla pystyi navigoimaan e-kirjan sisällä. Uudemmissa laitteissa oli parannettu akunkestoa, joka ei ollut riittävän pitkä vieläkään. Käytettävyys oli parantunut ensimmäisistä e-kirjalukulaitteista, mutta käytettävyydestä oli kuitenkin vielä eriäviä mielipiteitä. [33]

E-kirjoja kehitettiin ja niihin tuotiin vähemmän virtaa kuluttavia ominaisuuksia kuten sähköinen muste. Sähköinen muste mahdollisti lukemisen päivänvalossa. Näyttö ei heijastanut enää niin paljoa vaan käyttäytyi enemmän paperin kaltaisesti. Näytöistä pystyi lukemaan paremmin, ne veivät vähemmän virtaa. Näytöt olivat ohuita ja kevytrakenteisia. [33]

Richardsonin ym. kyselytutkimuksessa ilmeni, että e-kirjalaitteita oli paljon tarjolla ja niiden ominaisuudet vaihtelivat. Yksi halutuimpia ominaisuuksia oli kannettavuus, johon vaikuttivat laitteen koko ja paino. Laitteeseen täytyi pystyä tallentamaan monia kirjoja ja niiden lataaminen piti olla helppoa. Näyttöjen ei ollut pakko olla värillisiä. Lisäksi sanakirjan piti olla helposti saatavilla laitteessa. [77]

E-kirjat kehittyvät koko ajan ja ne ovat keskenään erilaisia. E-kirjojen suosio riippuu käyttäjien mieltymyksistä. Huonoja käytettävyyteen vaikuttavia asioita olivat huono siirtyminen kirjan sivulta toiselle, sitoutuminen tiettyyn e-kirjakauppaan ja se, että kirjoja ei pystynyt lainaamaan. Lisäksi laitteet vanhenivat nopeasti ja suurimmassa osassa akkua ei voinut vaihtaa. [77]

Erilaiset painikkeet ja symbolit olivat tärkeitä seikkoja laitteiden käytettävyydessä. Navigointi ja eri toimintojen oli oltava helppoa ja yksinkertaista. Joissakin laitteissa oli erikseen mukana kynä, jolla laitteen kosketusnäyttöä pystyi käyttämään. Käyttö oli kuitenkin käyttäjien mielestä melko hankalaa. Lukulaitteissa olisi myös hyvä olla mahdollisuus valita, lukeeko laitteesta vaakatasossa vai pystytasossa. Näyttö ei saisi olla kiiltävä ja mieluiten mahdollisimman vähän heijastava. Lukulaitteen näytön ei tarvitse olla värillinen. Sopivan fonttikoon asettaminen ja mahdollisuus tallentaa useita kirjoja laitteeseen olisivat hyviä ominaisuuksia lukulaitteissa. [38]

E-kirjalaitteissa on monia hyviä ominaisuuksia. Alkuinvestointi on noin reilu sata euroa, mutta sen jälkeen kirjoja saa ladattua laitteeseen suuria määriä. Jos laitteen yhteydessä käytetään pilvipalvelua, saattaa kirjojen määrä olla rajaton. Pilvipalve-

lussa e-kirjat tallennetaan palvelimelle, josta ne ovat aina ladattavissa käyttöön. Palvelimella voi olla rajattomasti kiintolevytilaa ja sen myötä e-kirjalaitteissa tilaa ei tarvitse olla suurta määrää. Kirjan saa ostettua nopeasti sähköisestä kirjakaupasta ja kirjoista saa ilmaisia kokeilukappaleita, joita pystyy tutkailemaan ennen ostoa. E-kirjoissa ei ole painoksia, kuten painetuissa kirjoissa. E-kirjakaupassa on saatavilla vanhoja ja uusia kirjoja. Mikä tahansa kerran julkaistu kirja on siis ladattavissa aina ja milloin tahansa. Monessa e-kirjalaitteessa pystyy vaihtamaan marginaalin kokoa, riviväliä ja fontin kokoa. Laitteissa kestää akku yhdellä latauksella useita kuukausia. [53]

4.2.2 Tablet-tietokoneiden ja sovellusten käytettävyys

Tablet-tietokoneet ovat pöytätietokoneen ja mobiililaitteiden välimuoto [39]. Tablettia käytetään ympäri taloa, kuten makuuhuoneessa ja olohuoneessa. Tabletin käyttötapa elää käyttäjän mukana. Tablet-tietokone ei ole vielä kuitenkaan käyttäjien pääasiallinen tietokone. Tablettia on helppo kantaa mukana, mutta se ei ole korvannut älypuhelinta kannettavuudessa. [22]

Vertailtaessa lukemiskokemusta tablet-tietokoneella ja kannettavalla tietokoneella, on näytön koko yksi tärkeä vertailun kohde. Kannettavassa tietokoneessa on yleensä suurempi näyttö ja siihen mahtuu enemmän tietoa yhdelle näytölle. Toisaalta tablet-tietokonetta voi lukea kuten kirjaa ja sitä on helpompi pidellä ilmassa. Kannettavilla tietokoneilla kirjoittaminen on helpompaa erillisen näppäimistön takia, mutta uusimmilla tablet-tietokoneilla voi lisäksi myös kirjoittaa käsin, kuten kirjoittaisi paperille. Kannettavalla tietokoneella ohjelmia avataan ja käytetään hiirellä kun taas tablet-tietokoneella ohjelmia käytetään sormin. [67]

Tablet-tietokoneista on tullut uusia sukupolvia. Tablet-tietokoneille tarkoitettujen www-sivujen ja sovellusten käytettävyys on parantunut. Laitteiden yleistyessä käytettävyyden ongelmat nousevat esiin. Kosketusnäyttö syöttölaitteena on erilainen kuin perinteinen hiiri ja näppäimistö. Painikkeita ja aktiivisia kohteita napautetaan (tap) sormilla, jolloin kosketusnäytöllisissä laitteissa voi tulla vahinkokosketuksia. Näytössä näkyvää aluetta voidaan monikosketuksen myötä koskettaa monesta kohtaa samanaikaisesti. Näytöllä näkyvää sisältöä voidaan muun muassa suurentaa nipistämällä kahdella sormella ulospäin ja pienentää nipistämällä sisäänpäin. Web-sivujen rakenne ja sisältö voidaan optimoida tablet-tietokoneille ja niitä varten voidaan rakentaa omia versioita web-sivuista.

Raluca Budiu ja Jakob Nielsen kertovat tutkimuksessaan *Usability of iPad Apps*

and Websites [19] *iPad*-sovellusten ja web-sivujen käytettävyydestä. Budiu ym. tutkivat *iPadin* käytettävyyttä *iPad*-omistajien haastattelulla ja käytettävyystesteillä, joissa henkilöiden piti tehdä erilaisia toimintoja sovelluksilla ja web-sivuilla. Tablet-sovelluksissa ja web-sivuissa olisi hyvä olla *takaisinpainike*-toiminto ja sovelluksissa saisi olla laajemmin käytössä hakutoiminto. Sovelluksissa tulisi olla kotisivu ja etusivulta olisi hyvä päästä lukemaan artikkeleita suoraan otsikoita klikkaamalla. Napautettavat alueet saisivat olla selkeästi ilmaistu ja pienissä napautettavissa kohteissa kosketus toimisi, vaikka sormi ei aivan tarkalleen osuisi painettavaan kohtaan. [19]

Esimerkkinä huonosta käytettävyydestä on USA Todayn sovellus, jossa pääsi osastolistaukseen painamalla USA Today-päälogoa. Logossa ei ollut mitenkään ilmaistu, että siitä aukeaa osastolistaus. Budiu ym. löysivät käytettävyystutkimuksessaan useita käytettävyyteen vaikuttavia asioita. Web-sivuissa napautettaviin painikkeisiin ja alueisiin voi olla vaikea osua sormella, jos ne ovat liian pieniä. Napautettava alue pitäisi olla tarpeeksi suuri. Painikkeen lähiympäristö voisi vastaanottaa myös napautuksen eikä pelkästään painike itse. Tarkastellaan esimerkiksi tilannetta, jossa web-sivulla on *tallenna*-painike. Painiketta on helppo klikata normaalilla pöytäkoneen hiirellä. Tätä samaa painiketta voi olla hankalampi napauttaa sormella. Ratkaisuna voisi olla se, että painike olisi jonkin laajemman HTML-elementin sisällä ja kun käyttäjä napauttaa elementtiä, niin painiketta painettaisiin virtuaalisesti, vaikka itse napautus ei osuisikaan tarkalleen painikkeeseen. Samankaltainen ongelma ilmenee, jos sovelluksessa on esimerkiksi liian tiiviisti vierekkäin useita painikkeita. Tällöin käyttäjä saattaa vahingossa napauttaa väärää painiketta. [19]

Sovelluksissa täytyisi kiinnittää huomiota käyttäjän vahinkopainalluksiin. Ongelma käytettävyydessä voi tulla esimerkiksi silloin, jos käyttäjä vahingossa koskettaa jotakin väärää kohtaa käyttöliittymässä, josta siirrytään johonkin toiseen näkymään sovelluksessa. Tässä tilanteessa käyttäjä tarvitsee *takaisin*-painiketta, jolla pääsee takaisin edelliseen näkymään. Napautettavat alueet ja painikkeet täytyisi tehdä niin, että niistä selvästi näkee niiden olevan kosketettavia alueita. Lisäksi olisi hyvä minimoida tarpeeton kirjoittaminen. Budiu ym. tutkimuksessa ilmeni, että käyttäjät eivät halua mielellään kirjoittaa kosketusnäytöllä. Web-sovelluksissa ja natiiveissa sovelluksissa täytyisi välttää esimerkiksi sisäänkirjautumisia ja rekisteröitymistä. Sovelluksen vaatiessa sisäänkirjautumista, se voitaisiin rakentaa niin, että sisäänkirjautuminen täytyy tehdä vain kerran. [19]

Joissakin pc-tietokonesovelluksissa on käynnistettäessä näkyvissä käynnistys-

kuva (splash screen). Käynnistyskuvassa on yleensä kuva, joka näkyy sovelluksen käynnistämisen alussa, kun odotetaan, että sovellus latautuu. Kuvassa voi olla myös sovelluksen tai yrityksen logo ja lisätietoja. Budiun ym. tutkimuksessa selvisi, että tämä toiminnallisuus on huono ja se ärsyttää käyttäjää — varsinkin jos käynnistysnäkymällä ei ole mitään muuta tarkoitusta kuin olla olemassa. Käynnistysnäkymä on turha jos taustalla ei ladata mitään tai siinä näkyy ainoastaan jotain tietoja ohjelmasta. Sovellusta avattaessa ei ole hyvä, jos joutuu aina katsomaan käynnistyskuvan. [19]

Käyttöliittymästä tekee huonon myös se, jos siinä on useita alueita, joissa on pyyhkäisy (swipe) -toiminnallisuuksia. Tämä ongelma nousee esiin etenkin sovelluksissa tai sivustoissa, joissa navigoidaan pyyhkäisyllä. Tietoa ei kannata asettaa liian tiiviisti näkyvään alueeseen eikä navigointitoiminnallisuuksia saa olla liikaa. Navigoinnin voi pitää yksinkertaisena. Käyttäjän voi olla helpompi palata edelliseen näkymään ja siirtyä sieltä eteenpäin kuin selata läpi pitkä lista navigointivalintoja löytääkseen hakemansa. Karuselli (carousel) tyyppiset navigoinnit voivat olla hankalia käyttää, jos ne ovat liian pitkiä. Sivussa olevassa karuselliosiossa käyttäjä voi pyyhkäisyllä siirtää karusellia eteenpäin ja taaksepäin. Tällainen on esimerkiksi näytön ylälaidassa oleva palkki, jossa on vaikkapa navigointikuvakkeita sanomalehden eri osastoihin. Palkkia pyyhkäisemällä palkissa olevat kuvakkeet liikkuvat ja käyttöliittymä näyttää siltä, kuin näytön ulkopuolelta tulisi lisää kuvakkeita näkyviin. Kuvakkeet sitten näennäisesti pyörivät ympyrää alkaen aina uudestaan, kun tullaan viimeiseen kuvakkeeseen. [19]

Tablet-laitteissa on huomioitavaa myös se, että niitä ei välttämättä käytä vain yksi henkilö, kuten yleensä matkapuhelimia. Tablettien käyttö jaetaan perheenjäsenten kesken. Sovelluksia tehdään siis laitteille, joissa on monia käyttäjiä, ainakin niin kauan, kunnes tablet-laitteiden hinnat laskevat alas. Monia käyttäjiä huomioidaan sovelluksissa siten, että sovelluksesta voi kirjautua ulos ja siihen voi kirjautua eri tunnuksilla samalla laitteella. Laitteen monet käyttäjät asentavat laitteeseen omia valitsemiaan sovelluksia, joista toiset käyttäjät eivät ehkä ole tietoisia. Sovellusten pikakuvakkeet olisi hyvä olla helposti tunnistettavia ja yksilöllisiä, jotta ne erottuvat sovelluspaljouden joukosta. [19]

5 Sanomalehdestä tablet-uutissovellukseksi

On hyvä miettiä pitävätkö käyttäjät enemmän web-sivun tyylisestä lukukokemuksesta vai halutaanko käyttää sanomalehden tyylistä käyttöliittymää. Luetaanko uutisia Internet-selaimella vai asennetaanko lukua varten aina oma sovellus sovelluskaupasta. Ovatko HTML5 ja Javascript riittävän laajoja kehitystyökaluja luomaan hyvän uutissovelluksen.

Jussi Ahlroth kirjoittaa artikkelissaan *The nine commandments for newspapers on tablet devices* [3] uutisjulkaisun tekemisestä tablet-laitteille. Painetulla sanomalehdellä on vuosisatoja pitkä historia. Sanomalehti kuuluu satojen miljoonien ihmisten päivittäiseen elämään. Vanhempien päätoimittajien ja sanomalehtien johtajien mielestä painettu lehti on paras alusta. Tieto ei perustu mihinkään tutkimukseen tai todisteisiin vaan pelkästään kokemukseen. Todellisuus saattaa olla jotain muuta. [3]

5.1 Tablet-julkaisun suunnittelu

Vielä vuonna 2010 kotikoneet olivat suosituin tapa lukea elektronista sisältöä [38]. Markkinoilla oli jo nousussa olevia lukulaitteita. Applen *iPad* julkaistiin 2010. Se avasi samalla täysin uuden digitaalisen julkaisun alan [3]. *iPadin* julkaisun jälkeen myös muut laitevalmistajat alkoivat valmistaa tablet-tietokoneita. Tablet-tietokoneet ovat herättäneet paljon keskustelua valmistajien ja sisällöntuottajien keskuudessa. Ennen uusien tablet-tietokoneiden julkaisua pohjaa rakensivat e-kirjojen lukulaitteet, joilla pystyi pääosin vain lukemaan. Tablet-tietokoneilla pystyy tekemään samoja asioita kuin perinteisellä tietokoneella ja se on monikäyttöisyytensäkin takia suosittu laite. Vuonna 2011 lähes jokainen suuri mediatalo mietti tablet-julkaisun tekemistä. [37]

Puolitoista vuotta ensimmäisen iPadin julkaisun jälkeen oli vielä epätietoisuutta laitteen asettumisesta uutisten lukualustaksi ja epätietoisuutta myös mediasisällöstä tablet-tietokoneella. Vielä yli kahden vuoden jälkeen oli enemmän kysymyksiä kuin selviä vastauksia. Useat lehdet ja muut mediat olivat tuottaneet monelle eri tablet-alustalle jonkinlaisen version julkaisustaan. Julkaisijat eivät olleet vielä löytäneet tabletille varmaa strategista tai taktista ratkaisua. Tablet-julkaisuista heräsi

kysymyksiä, kuten ketkä käyttävät tablet-tietokoneita ja miten niitä käytetään? Vielä oli epävarmaa, oliko suuri yleisö omaksunut tablet-alustan käyttöönsä ja yhdeksi media-alustoista. [39]

Tablet-uutissovellusta tehdessä mietitään myös, voiko tablet-uutissovellus olla tuottoisa. Liiketaloudellinen logiikka ohjaa kaupallista mediaa ja tablet-markkinat ovat kasvaneet nopeasti [39]. Markkinayhtiöt raportoivat tablet-laitteiden myynnistä, joka on kasvanut valtavasti vuonna 2012 [39]. Tabletit eivät kuitenkaan vielä korvaa perinteisiä tietokoneita. Kesällä 2013 tehdyn tutkimuksen [105] mukaan 34% Yhdysvaltalaisista omisti tablet-tietokoneen. Heistä noin puolet oli 35-45 vuotiaita ja heistä suurin osa oli hyvätuloisia [105]. Tablet-markkinat ovat kasvamassa eikä tablettia voi sivuuttaa.

Picard kirjoittaa artikkelissaan *Business Realities of Tablets and E-Readers* [70] tablet-laitteiden ja e-lukijoiden mahdollisuuksista. Tabletit ja e-lukulaitteet tarjoavat monia mahdollisuuksia ja monet näkevät ne sanomalehtien pelastajina. Laitteiden ympärillä pyörivää keskustelua käydään enemmän julkaisijoiden kuin kuluttajien näkökulmasta. Monien julkaisijoiden mielestä sähköinen julkaisu säästää rahaa ja tuottaa uusia tulonlähteitä. On kuitenkin hyvä keskittyä ensin selvittämään, mitä kuluttaja haluaa ja mitä kuluttaja saa, jos hän vaihtaa lukualustansa sähköiseen muotoon. Tablet-markkinoiden haasteet on tunnistettava julkaisua suunnitellessa. Ihmiset, jotka eivät tilaa ja lue sanomalehteä eivätkä maksa suhteellisen edullista lehden tilausmaksua, eivät todennäköisesti mielellään osta muutaman sadan euron lukulaitetta ja maksa samaa summaa sähköisestä lehdestä. [70]

Tutkimusten mukaan tablet-tietokoneiden omistajista enemmistö käytti laitetta päivittäin ja puolet käyttäjistä seurasi tabletilla uutisia [60]. Toisen tutkimuksen Roger Fidlerin *2012 RJI-DPA Mobile Media News Consumption National Survey* [28] mukaan mobiililaitteiden, joihin kuuluvat kaikenlaiset kannettavat laitteet ja matkapuhelimet, tärkeimmät käyttökohteet olivat henkilökohtainen viestintä, viihde, webin selailu ja uutisten luku. Tutkimusten tuloksista selviää, että tablettia ei käytetä vain yhteen asiaan tai palveluun vaan se toimii käyttäjillä muiden laitteiden ja tietokoneiden kanssa yhtenä kokonaisuutena. [28]

Kuuluisa sanomalehtien suunnittelija Mario Garcia pitää tablet-tietokoneita mediateollisuuden suunnan muuttajana. Garcian mielestä ei ole kysymys siitä tehdäänkö julkaisuista tablet-versioita, vaan kysymys on siitä milloin se tapahtuu. Mario Garcialla on 30 vuoden kokemus sanomalehtien uudistamisesta. Kokemuksen nojalla hänen mielestään monet lehtitalojen toimittajat vastustavat suuria uudistuksia.

Sanomalehtiä ei pidä siirtää tablettiin, vaan on luotava pelkästään tabletille suunnattu julkaisu. Tablet-versio ei ole pelkkä online-julkaisu tai sovellus, vaan se on oma alustansa, joka antaa jutuille lisäarvoa. [91]

Myös Roger Fidler, joka toimi digitaalisen julkaisemisen johtajana *Reynold Journalism Institutessa*, on sitä mieltä, että sanomalehteä ei pidä kopioida suoraan näköisversiona tablet-tietokoneille. Sanomalehdet, jotka kopioidaan muuttumattomana printtilehdestä tablet-lehdeksi menettävät tablet-tietokoneet tuomat suuret mahdollisuudet. *iPad* ei ole jättikokoinen älypuhelin, vaan se voi olla tulevaisuudessa tärkein uutisalusta. Financial Timesin *Online Managing Director* Rob Grimshawn mielestä ihmiset oppivat lapsesta pitäen ympärillä olevasta maailmasta koskemalla ja testaamalla mitä tapahtuu. Grimshawn mielestä tämä sama pätee kosketusnäytöllisiin tablet-tietokoneisiin. [47]

Tablet-uutissovelluksen suunnittelussa voi hyödyntää konkreettista tietoa Poynter Instituten tekemästä tutkimuksesta, jossa on keskitytty erityisesti kuvagallerioiden selaamiseen. Tutkimuksen mukaan tabletissa kuvagallerioita pyyhkäistään automaattisesti laidasta laitaan eikä pystysuunnassa riippumatta kuvan asettelusta [75]. Pyyhkäisytoiminto on tärkeä ominaisuus tablet-uutissovelluksessa [75]. Uutissovelluksen suunnittelussa voi miettiä miten uutisia luetaan. Mihin suuntaan käyttäjä pyyhkäisee vai pyyhkäiseekö ollenkaan. Tapahtuuko selaus samaan tapaan kuin web-sivuilla eli pystysuunnassa rullaten vai selataanko vaakatasossa näyttösivu kerrallaan.

Tablet-laitteet ovat monessa suhteessa kehittyneempi julkaisualusta kuin paperilehti. Kehittyneempiä asioita ovat sisällön lukeminen ja visuaalisuus, interaktiivisuus ja kannettavuus. Sisällön lukeminen on käyttäjälle helppoa. Näyttö on tarkka ja siitä voi katsoa kuvia, videoita ja lukea uutistekstiä. Interaktiivisuudessa tablet on myös omaa luokkaansa. Se, että rikastettua mediasisältöä voi käyttää ja käyttöliittymää voi koskettaa suoraan sormilla, on paljon parempi kuin painetun sanomalehden käyttöliittymä tai perinteisen tietokoneen hiiri ja näppäinyhdistelmä. [3]

Tablet-sovellusta tehdessä voi miettiä, millä tavoin julkaisu tarjotaan käyttäjälle. Jos tehdään natiivi sovellus, siitä joudutaan tekemään oma versionsa jokaiselle tablet-alustalle. American Journalism Review'n artikkelissa [47] ollaan sitä mieltä, että tablet-uutissovellus pitäisi olla osa Internetiä. Monet julkaisut ovat jo tehty internetiin HTML5-perustaisina. Tämä tapa houkuttelee monia, koska sellaisen voi tehdä kuka tahansa ja sitä ei ole sidottu mihinkään sovelluskauppaan ja lisäksi se on aina uusin päivittynyt versio, kun sovelluksen avaa. Tablet-julkaisua ei kannata

sulkea painetun paperilehden lukittuun ympäristöön, vaan sen pitäisi olla vapaampaa ja jotain muuta.

Lukijat käyttäjät Internetiä ja liikkuvat siellä, ovat sosiaalisia verkossa, tykkäävät sivuista ja jakavat sisältöä toisilleen. Todennäköisesti käyttäjät eivät palaa kiinteän painetun sanomalehtityyppisen julkaisun käyttömalliin. Ensimmäiset tablet-uutissovellusten käyttäjät ovat mitä luultavimmin sanomalehden lukijoita. Tällaiset lukijat ovat yleensä hyvin toimeentulevia, koulutettuja ihmisiä. Tällainen yleisö on hyvä kohderyhmä, mutta nämä sanomalehden lukijat vanhenevat vuosi vuodelta eikä uusia lukijoita tule. Mediatalouden tutkija Rober G. Picard on sanonut, että sanomalehden tulevaisuus on kiistämättä digitaalinen [70]. Jotain uutta on keksittävä sähköisten näköislehtien lisäksi. [39]

5.2 Mainonnan esittäminen

Mainoksia voidaan esittää www-sivuilla, mobiilisivuissa, niille varatuilla paikoilla. Mainonta on usein erikseen määritellyissä laatikoissa ympäri sivua. Tehdyt mainokset olisi hyvä olla jonkin mainosstandardin mukaisia, jotta voitaisiin helposti käyttää myös maailmanlaajuisia mainoksia sivustoissa [93]. Joissakin ilmaisissa mainosrahalla toimivissa matkapuhelin- ja tabletsovelluksissa on asetettu yksi tietty mainospaikka, jossa mainokset vaihtuvat. Tablet-uutissovelusta tehdessä voitaisiin kuitenkin asettaa mainonta osaksi sovellusta niin, että se on miellyttävää käyttäjälle.

Googlen mobiili- ja tablet-mainonnan ohjeistuksessa [35] neuvotaan mobiililaitteiden ja tablettien mainonnassa. Ohjeistuksen mukaan tablet- ja mobiilimainonta tulisi näyttää samalta kuin perinteisten tietokoneiden mainonta ja mainonnan tulisi toimia kaikissa internet-selaimissa [35]. IAB (Interactive Advertising Bureau) raportin [43] mukaan tablet-mainonnassa täytyy ottaa huomioon laitteiden erilaisuus. Näytöt ovat erikokoisia ja erilaisilla resoluutioilla. Tällöin tietyn levyiset ja korkuiset mainokset eivät ole hyviä, koska ne näyttävät erilaisilta eri laitteilla. Toisaalta voidaan ryhmitellä tabletit "suuriin näyttöihin" ja "pieniin näyttöihin" ja tehdä mainontaa ryhmille. Www-sivujen mainonnassa olisi hyvä tunnistaa laite ja näyttää mainontaa sen mukaan onko käyttäjä saapunut sivulle tabletilla vai perinteisellä tietokoneella. Lisäksi tablet-mainonnassa on huomioitava näytön eri asennot. Näyttö voi olla vaakatasossa tai pystytasossa. [43]

VivaKin mediatutkimusaloitteen The Poolin tekemässä tutkimuksesta [93], selvitettiin parhaita tabletin mainontamalleja 14 kk kestäneellä tutkimuksella. Tutki-

muksessa oli mukana paljon suuria yrityksiä mm. Bank of America, Best Buy, The Coca-Cola Company, General Motors, ABC Television Network, NBC News. Tutkimuksen mukaan parhaita mainontamalleja tabletille olivat [93]:

- Bannerista koko ruutuun avautuva mainos. Mainos näkyy sivulla bannerina tai jonakin laatikkona. Laatikossa olevan mainoksen voi avata koko ruutuun. Koko ruudun mainoksessa voi olla pientä interaktiivisuutta, kuten osallistuminen arvontaan tai mainoskirjeen tilaus jne.
- Esimainokset videoissa. Ennen kuin varsinainen video toistetaan, näytetään mainos. Mainoksessa käyttäjä voi mainoksen katsomisen lisäksi selata myös muita mainosvideoita sekä jakaa niitä sosiaaliseen mediaan ja pelata pelejä.
- Rikastettu interstitiaali eli välisivun mainos. Mainos näytetään sovelluksessa välisivuna tai koko ruudun osiona. Mainoksessa käytetään hyödyksi kaikkea tabletin tarjoamia mahdollisuuksia, kuten kosketusnäyttöä ja sensoreita. Mainoksen sisällä voi esimerkiksi selata videoita ja kuvia.

Muita mainontatapoja ovat sovelluksen sisällä olevat mainokset, koko ruudun mainokset, joissa on huomioitava tabletin näytön suunta ja sovelluksen sisällön mukaan sponsoroidut mainokset. Mainoksille olisi hyvä olla myös jokin standardi, jonka mukaan mainoksia voidaan tehdä eri laitteille. Mainonnassa voidaan myös hyödyntää tablet-laitteiden sensoreita esimerkiksi niin, että pyydetään käyttäjää ravistelemaan laitetta, jolla osallistuu arvontaan ja näkee jonkin mainoksen. Avautuvassa mainoksessa voi olla myös jokin pieni tehtävä. Mainoksia voidaan räätälöidä myös sovellusten käyttöliittymään sopivaksi. Tämä voidaan tehdä esimerkiksi samanlaisella väriteemalla kuin sovelluksessa tai näyttämällä sovelluksen sisältöön liittyviä mainoksia. [93]

5.3 Lukutottumuksista tablet-julkaisun ideointiin

Amy Mitchell, Leah Christian ja Tom Rosenstiel tutkivat artikkelissaan *The Tablet Revolution and What it Means for the Future of News* [60] tablet-tietokoneiden käytettävyyttä ja uutisten lukutottumuksia. Yhdysvaltojen täysi-ikäisistä 11% omisti jonkinlaisen tablet-tietokoneen 18 kuukautta *iPad*in julkaisun jälkeen. Heistä noin puolet saivat uutiset tablet-laitteisiinsa joka päivä. Suurin osa tabletin omistajista

käytti laitetta päivittäin. Uutisten lukemisessa käyttäjät käyttivät mieluiten tablet-tietokonetta kuin perinteistä tietokonetta, televisiota tai printtilehtiä. Uutisten luku oli yksi suosituimmista aktiviteeteista tabletilla eli suunnilleen yhtä suosittua kuin sähköpostin lukeminen. Uutisten luku oli suositumpaa kuin sosiaaliset mediat, pelaaminen, kirjojen lukeminen ja videoiden katsominen. Vain perinteinen Internetin selailu oli suositumpi aktiviteetti tablet-tietokoneella. [60]

Tablet-tietokoneiden tultua julki pääteltiin, että tablet muuttaa uutisten käyttökokemusta ja digitaalisten uutisten käyttöä. Tämä päätelmä pohjautui sille, että ajateltiin, että käyttäjät asentaisivat jonkin maksullisen uutistenlukusovelluksen tablet-tietokoneisiinsa. Uutisten tuottajat saisivat siitä tuottoa. Mainostaminen olisi myös tällöin miellyttävämpää käyttäjälle. Mitchellin ym. tutkimuksessa selvisi myös, että vaikka suurella osalla käyttäjistä oli uutissovelluksia asennettuna tablet-tietokoneeseensa, uutisia luettiin pääosin Internet-selaimella. Selaimen lukukokemus oli riittävä ja se oli suositumpi uutisten lukutapa kuin asennettavat uutis-sovellukset. [60]

Princeton Survey Research Associates International toteutti kyselyitä tablet-tietokoneiden käytöstä. Kyselyissä selvisi muun muassa se, että tablet-uutisten tuotto-potentiaali voi olla rajallinen. Kyselyiden mukaan vain pieni osa tablet-uutisten lukijoista maksoi uutisten lukemisesta suoraan. Toinen pieni osa lukijoista sai tablet-teihinsa uutiset jonkin paperilehtitilauksen lisäpalveluna. Käyttäjät olisivat valmiita maksamaan vain, jos ei olisi muuta keinoa saada uutisia ja tällöin maksun pitäisi olla pieni. Tutkimuksen mukaan käyttäjät ovat valmiita maksamaan enintään 5 dollaria kuukaudessa. [60]

Kyselyiden mukaan tablet-sovelluksissa brändi on tärkeä. Uutisten olisi hyvä tulla tunnetulta uutistuottajalta. Suurin osa käyttäjistä luki uutiset mieluummin suoraan uutisten tuottajan sivulta, kuin jonkinlaisesta uutisten koostesovelluksesta, jossa uutiset haetaan monesta eri lähteestä. Tablet-käyttäjät lukevat mieluimmin uutiset tabletilta eikä muista medioista. Uutisten lukeminen tietokoneilla on korvattu melkein kokonaan tablet-tietokoneella. Televisiosta, aikakausilehdistä ja sanomalehdistä luetuista uutisista noin puolet vastaanotettiin tablet-tietokoneilla. [60]

Sattumanvarainen uutistenluku on kasvanut tablet-tietokoneiden käyttäjillä. Monet käyttäjistä päätyivät lukemaan pitkiä uutisartikkeleita, joita he eivät alun perin olleet etsimässä. Kyselyiden mukaan tablet-tietokoneella uutissovellusten käyttäjät lukevat uutisia enemmän nyt kuin ennen tabletin hankkimista. Mielenkiintoiset uutiset ja artikkelit leviävät todennäköisemmin suusta suuhun kuin sähköisesti jaka-

malla. [60]

Tablet-tietokoneiden käyttäjät lukevat uutisia useammin kuin muut ihmiset. Toisin kuin muilla ihmisillä, tablet-käyttäjien pääasiallinen uutistenlähde on Internet. Lisäksi tablet-käyttäjillä on todennäköisemmin parempi käsitys suurista uutistapahtumista. Kaksi kolmasosaa tablet-käyttäjistä kertoo seuraavansa uutisia melkein jatkuvasti. Tablet-käyttäjät haluavat mieluummin lukea tai kuunnella uutisia kuin katsoa uutisia televisiosta. Tabletin omistavat käyttäjät kuluttavat enemmän aikaa uutisiin, uutisten hakuun ja heillä on yleensä ottaen parempi kokemus uutisten löytämisessä ja vastaanottamisessa. [60]

Internetissä pitkien uutisten lukukokemus voi olla heikkoa tietokoneilla ja älypuhelimilla. Vuonna 2010 keskimääräinen uutisten lukuaika oli 2 minuuttia ja 30 sekuntia. Pitkien artikkelien lukua varten täytyy kehittää parempia käyttöliittymiä ja tähän tarpeeseen tablet-tietokoneet voivat olla ratkaisu. [60]

Pitkien uutisartikkelien ja korkeiden rullattavien (scroll) web-sivujen lukeminen voi olla aikaa vievää, käyttöliittymä voi olla ruma ja häiritsevä. Web-sivulla voi olla mainoksia ja muita ylimääräisiä ja tarpeettomia asioita. Tätä varten on olemassa palveluita kuten *Instapaper* [41] ja *Pocket* [76], joilla voidaan erottaa sisältö häiritsevältä sivulta. Näillä sovelluksilla artikkelin voi tallentaa myöhempää lukemista varten talteen. Pitkiä tekstejä varten käyttöliittymän pitäisi olla siisti ja luettava. [65]

ESPN.com:n Patrick Hruby:n kirjoittama artikkeli *The Long Strange Trip of Dock Ellis* [40] on hyvä esimerkki mielenkiintoisesta pitkästä artikkelista, jonka haluaa lukea ja selvittää mitä artikkelissa tulee seuraavaksi. Artikkelissa on kuvia ja videoita sekä tausta tavallaan liikkuu ja elää sitä mukaa kun artikkelia kelaa eteenpäin. Artikkelista tulee ilmi selvästi se, että koko ajan ollaan menossa eteenpäin ja tausta elää sen mukana. Sivulla on paljon vapaata tilaa ja tekstin fontti on tarpeeksi suurta, jotta sitä on mielenkiintoista lukea. [65]

Toinen esimerkki pitkien artikkelien luettavuudesta on tekniikkauutisia julkaiseva *The Verge* [95]. Sivustolla artikkelit ovat pitkiä ja niissä on välillä kuvia, videoita ja muotoilu muuttuu, kun sivua rullaa alaspäin. Artikkelit näyttävät suunnitelluilta ja hiotuilta. Uutisista yritetään tehdä mielenkiintoisempia luettavia. Sovellusten ja web-sivujen olisi hyvä toimia kaikilla alustoilla ja erikokoisilla näytöillä. Nykypäivän julkaisuissa pyritään siihen, että se tehdään kerran ja sen pitää toimia monessa paikassa. [65]

Lukijat oppivat käyttämään tablet-julkaisuja nopeammin, jos lehti käyttää painetusta lehdessä tai web-sivuissa käytettyjä navigointikäytäntöjä. Tablet-julkaisuissa

voidaan näyttää enemmän kuvia ja muuta sisältöä, kuin painetussa lehdessä. Tablet-julkaisu voi olla lehtimäinen ja kosketusnäytön myötä kosketeltava. Tabletin voi nostaa ylös sanomalehtimäiseen lukuasentoon. Printtilehdessä toimitus saa valita, miten juttujen tärkeyttä korostetaan. Tämä toiminto olisi hyvä tuoda myös tablet-julkaisuun. Tablet-julkaisussa voidaan esittää reaaliaikaista uutissisältöä kiinteäsisältöisen lehden sijasta. Uutiset voivat päivittyä päivän aikana. Tablet-julkaisu aktivoi lukijaa entistä enemmän interaktiivisuuden myötä. Lukijat voivat esimerkiksi antaa palautetta, kommentoida uutisia ja keskustella keskenään uutisesta. Lukija voisi myös mahdollisesti tehdä itse luettavaa materiaalia julkaisuun. [69]

Tablet-julkaisuun voidaan toteuttaa laadukasta lehtimäistä sisältöä. Ei tarvitse tehdä pelkästään lyhyitä ja nopeita nettiuutisia. Tablet-julkaisu on hieno alusta, mutta sen ei tarvitse olla ainoa jakelutapa. Lehtitalojen ei kannattaisi panostaa ainoastaan yhteen julkaisukanavaan. Jokaiseen eri julkaisukanavaan olisi hyvä tuottaa myös vain niitä varten tehtyä sisältöä, jotta käyttäjä saa vastinetta rahoilleen. Tablet-julkaisulle voitaisi tuottaa pelkästään sille suunniteltua sisältöä. [69]

5.4 Olemassa olevia HTML5-uutissovelluksia

Kirjoitettaessa tablet-laitteille tarkoitettuja HTML5-uutissovelluksia ilmestyi koko ajan uusia ja niitä on olemassa suuri määrä. Web-sovelluksista esitellään tässä neljä.

Financial Times

Financial Timesin web-sovellus on optimoitu *iPadille* ja *iPhonelle*. Se toimii web-sovelluksena eikä sitä tarvitse asentaa *Applen* sovelluskaupasta erikseen. Artikkeleita voi tallentaa myöhempää lukemista varten. Se toimii parhaiten *iOS:n Safari*-selaimella menemällä *iPadillä* tai *iPhonella* osoitteeseen `http://app.ft.com/`. Sovelluksessa uutisosastot ovat omina sivuinaan ja niiden välillä voi navigoida pyyhkäisemällä näyttöä sivulle. Uutisen voi avata lukunäkymään napauttamalla ja valikko on näytön alareunassa.

Times Skimmer

New York Timesillä on maksullisen web-sovelluksen lisäksi ilmainen sovellus. *Times Skimmer*-sovellus toimii muun muassa *iPadillä* ja uusimmilla Internet-selaimilla. Käyttäjä voi vaihtaa sovelluksen ulkoasua mieleisekseen. Käyttöliittymässä voi se-

lata sivu kerrallaan uutisosastoja läpi. Osastot näkyvät oikeassa reunassa listana. Koko sisällön voi selata läpi pyyhkäisemällä sivu kerrallaan alaspäin. Web-sovellus löytyy osoitteesta `http://www.nytimes.com/skimmer`.

HS Päivän lehti

Helsingin Sanomien toteuttama *Päivän lehti*-web-sovellusta voi käyttää mm. *iPadillä*. Käyttöliittymässä on valikko yläreunassa, jossa ovat listattuna kaikki uutisosastot. Uutisosastoja voi selata sivu kerrallaan pyyhkäisemällä vaakatasossa. Uutisen saa avattua lukunäkymään napauttamalla ja uutista voi lukea sivu kerrallaan pyyhkäisemällä sivulle. Sovellus löytyy osoitteesta `http://www.hs.fi/paivanlehti`.

Helppo Aamulehti

Aamulehden toteuttama *Helppo aamulehti*-web-sovellusta voi käyttää tablet-laitteilla ja se löytyy osoitteesta `http://helppo.aamulehti.fi`. Sisältö on selattavissa sivu kerrallaan vaakatasossa pyyhkäisemällä. Näytön alareunassa näkyy millä sivulla on menossa. Uutiset ovat laatikoissa sivulla ja uutisen saa avattua lukunäkymään napauttamalla. Lukunäkymässä uutinen aukeaa koko ruutuun ja sitä voi selata rullaamalla alaspäin.

5.5 Käyttöliittymän suunnittelu tablet-uutissovellukseen

Sovellusta käytetään graafisen käyttöliittymän kautta. Käyttöliittymä on tärkeä osa käyttökokemusta. Käytön pitää olla helppoa ja yksinkertaista. Sovelluksen käyttö olisi hyvä oppia suoraan käyttöliittymän kautta, eikä ohjekirjoja lukemalla. Käyttöliittymän kautta tieto kulkee käyttäjältä sovellukselle ja sovellukselta takaisin käyttäjälle. Tämän rajapinnan olisi hyvä olla luonnollinen. [46]

Uutissovellukset ja muut sovellukset voivat olla erilaisia. Niiden suunnittelussa voidaan kuitenkin käyttää samoja perusperiaatteita. Selkeä ulkoasu on sellainen, jota käyttäjä ei välttämättä erikseen pane merkille. Ulkoasun huomaavat ehkä vain toiset suunnittelijat. Ulkoasu ja käyttöliittymä ovat läpinäkyviä käyttäjille. Voidaan ajatella, että uutissovelluksessa on vähintään kaksi näkymää. Ensimmäinen näkymä on kaukaa katsottu näkymä, jossa nähdään miten uutiset asettuvat laajemmalla skaalalla ja miksi lukijaa kiinnostaa uutinen kokonaisuutena. Toinen näkymä on läheltä katsottu näkymä, jossa uutinen koskee lukijan henkilökohtaisia asioita tai

oman paikkakunnan asioita. Ylipäänsä kaikki ne asiat, jotka koskettavat lukijaa henkilökohtaisesti. [36]

Kosketusnäytöllisillä laitteilla sovellusten käyttö on erilaista kuin perinteisellä pöytätietokoneella. Sovellusta suunnitellessa tärkeitä huomioitavia asioita ovat käyttäjän käden koko, käden eleet (gesture), käyttöliittymän korostus ja katseen ohjaus sovelluksessa. Sovellusta käytetään sormilla ja käyttöliittymässä olevien toimintojen tulisi olla helppo toteuttaa sormilla. Ikonien ja painikkeiden koko, elementtien pituudet, jokaisen näkymän asettelu ja muut käyttöliittymään liittyvät asiat tulisivat olla helposti käytettäviä sormilla. Käyttäjää pitäisi ohjata sovelluksessa siten, että sovellusta pystyy käyttämään. [46]

Sovellusten käyttötavat tulevat monesti tosielämän toiminnoista. Applen *iOS* käyttöjärjestelmässä saa poistotoiminnallisuuden näkyviin pyyhkäisemällä elementtiä vasemmalta oikealle tai oikealta vasemmalle. Tämä toiminnallisuus tulee tosielämästä, jossa käyttäjä yliviivaa jonkin asian paperilta, kun se on tehty. Käyttöliittymän graafisen ulkoasun ja hienojen yksityiskohtien olisi tuettava sovelluksen toiminnallisuutta ja tarkoitusta. [46]

Sovellusta tehdessä pitäisi asettaa sovellukselle tavoitteita. Käyttäjän pitäisi ymmärtää suhteellisen nopeasti, mikä on sovelluksen tarkoitus ja kuinka sitä käytetään. Sovelluksen alkunäkymän tulisi näyttää käyttäjälle mitä ja kuinka sovellusta käytetään. Olisi tärkeää pyrkiä yksinkertaistamaan sovellus niin, että käyttäjä ymmärtää helposti kuinka sovellusta käytetään. Sovelluksen sisältö on erittäin tärkeä osa sovellusta. Käyttäjän on helpompi keskittyä itse sisältöön kuin, että keskittyisi käyttöliittymään. [45]

Käyttöliittymässä oleva sisältö ja eri alueet tulisivat olla linjassa toistensa kanssa. Sisällön asettelu antaa rakenteelle yhtenäisyyttä. Samankaltainen sisältö on hyvä ryhmitellä toistensa kanssa lähekkäin. Toisiinsa liittyvät uutiset kannattaa asettaa käyttöliittymässä lähelle toisiaan. Taulukoilla voidaan selkeästi esittää suuriakin määriä tietoa luettavasti. Seuraavissa kuvissa on lueteltuina osa Suomen tasavallan presidenteistä. Kuvassa 5.1 osa Suomen tasavallan presidenteistä on listattuna taulukossa ja toisessa kuvassa 5.2 presidentit ovat esitetty peräkkäin tekstinä. Kuvista huomaa selkeästi asettelun hyödyllisyyden. [36]

Tekstissä voi käyttää hyväkseen värejä, korostusta ja fontteja. Tekstistä voidaan erottaa jokin tärkeä lause eri värillä tai lihavoituna. Korostusta voidaan käyttää myös uutiskuvissa. Vanha kuva voidaan haalistaa tarkoituksella ja tuoda esiin uusi kuva kirkkailla väreillä. [36]

presidentti	toimikausi	puolue
K. J. Ståhlberg	1919–1925	edistyspuoluelainen
Lauri Kristian Relander	1925–1931	maalaisliittolainen
P. E. Svinhufvud	1931–1937	kokoomuslainen
Kyösti Kallio	1937–1940	maalaisliittolainen
Risto Ryti	1940–1943	edistyspuoluelainen
	1943–1944	
Gustaf Mannerheim	1944–1946	sitoutumaton
J. K. Paasikivi	1946–1950	
	1950–1956	
...		

Kuva 5.1: Presidenttejä taulukossa [87].

K. J. Ståhlberg 1919-1925 edistyspuoluelainen. Lauri Kristian Relander 1925-1931 maalaisliittolainen. P. E. Svinhufvud 1931-1937 kokoomuslainen. Kyösti Kallio 1937-1940 maalaisliittolainen. Risto Ryti 1940-1943, 1943-1944 edistyspuoluelainen. Gustaf Mannerheim 1944-1946 sitoutumaton. J. K. Paasikivi 1946-1950, 1950-1956.

Kuva 5.2: Presidenttejä kirjoitettuna peräkkäin [87].

5.5.1 Käyttöliittymässä huomioitavia asioita

Käyttöliittymän pitäisi olla yksinkertainen ja siinä ei saa olla liikaa valintamahdollisuuksia yhdellä näkymällä. Valintojen ja toimintojen tulisi olla helposti ymmärrettäviä sekä niiden olisi hyvä olla selkeästi näkyvillä ja löydettävissä. Sovelluksessa tehtävien toimintojen tai sormiliikkeiden tulisi olla yhtenäisiä kaikkialla sovelluksessa. Sisältö pitäisi olla helposti saatavilla selkeässä ulkoasussa ja lajiteltuna osioihin ja aliosioihin. [3]

Käyttöliittymässä on hyvä pitää yhteinäinen ulkoasu. Kannattaa käyttää samaa fonttia ja väriteemaa koko sovelluksessa. Lomakkeita ja visuaalisia elementtejä kannattaa myös toistaa. Käyttäjä tietää ulkoasusta, että tämä näkymä kuuluu sovellukseen. Lomakkeita voi toistaa esimerkiksi, jos sovelluksen eri osissa käyttää hakulomakkeita, niiden pitäisi olla joka paikassa samanlaisia. Haun voi asettaa esimerkiksi värillä korostetun laatikon sisään, josta käyttäjän silmä löytää heti hakutoiminnon. [36]

Web-sovelluksia tehdessä täytyy miettiä, minkä kokoisella näytöllä sovellusta käytetään, mikä on näytön resoluutio ja mitkä HTML5:n ominaisuudet ovat toteutettu kohdelaitteen Internet-selaimessa. Toisaalta mietitään myös, minkälainen on tablet-julkaisun ulkoasu verrattuna paperilehteen [61]. Käyttöliittymää miettiessä täytyy ottaa huomioon muun muassa navigointi, näkymän rakenne, tehokas koodi, näytön koko, väriteema ja sovelluksen nopeus [23]. Lisäksi täytyy myös huomioida, että tablet-laitteissa näyttöä voi kääntää, jolloin leveys ja korkeus muuttuvat [99].

Sovelluksen näkymä voi olla rullattava (scroll) tai sivu kerrallaan selattava. Näyttö on mahdollista jakaa moneen osaan tai näytössä voi olla päällekkäisiä sovellusikkunoita. Päällekkäinen sovellusikkuna on esimerkiksi jokin sovelluksen käyttöliit-

tymän päälle avautuva pienempi ikkunapaneeli. Sovelluksessa tulisi yrittää vähentää koko ruudun siirtymiä niin, että näkymä vaihtuu kokonaan toiseen [45]. Tämä voi olla häiritsevää käyttäjälle ja viedä häneltä keskittymisen, koska kaikki mitä hän aiemmassa näkymässä ajatteli, on nyt kadonnut kokonaan [45]. Suunnittelussa on huomioitava muun muassa seuraavia asioita [99, 3, 45, 36]:

- Käyttöliittymä on suunniteltava pystysuuntaan ja vaakasuuntaan, koska laitetta voi kääntää.
- Vältä käyttämästä sivun peittäviä pienikkunoita (modal window/dialog box).
- Päätä käytetäänkö sovelluksessa vain näkyvää aluetta sivusta vai meneekö sisältö yli näytön ja on rullattava.
- Toimintojen ja ulkoasun pitää olla yhteneväisiä koko sovelluksessa.
- Hyödynnä koko näyttöpinta-ala.
- Suunnittele ja mieti navigointi tarkoin.
- Minimoi sovelluksessa tehtävien kosketusten määrä jonkin toiminnon toteuttamiseksi.
- Tee käyttöliittymästä sulava ja suorituskykyinen.
- Tee sovellukselle samankaltaiset käyttötottumukset kuin kohdelaitteen käyttöjärjestelmän käyttötottumukset.
- Tee sovellukseen mahdollisimman vähän koko ruudun siirtymiä.
- Tee käyttöliittymästä visuaalisesti erittäin tyylikäs ja vaikuttava
- Tallenna sovelluksen tila ja käyttäjän tekemät muutokset taustalla niin, että käyttäjän ei itse tarvitse erikseen tallentaa.
- Sovelluksen tulee avautua välittömästi eikä niin, että näytetään jokin latausruutu pitkän aikaa.
- Huomioi sovelluksessa se, että käyttäjä voi missä tahansa vaiheessa sammuttaa sovelluksen kesken käytön.
- Huomioi intuitio käytettävyydessä.

Käyttöliittymässä olevien objektien tulisi aina antaa vinkkiä toiminnallisuudesta. Esimerkiksi painikkeiden kuvakkeiden tulisi ilmaista, mitä niistä tapahtuu, kun käyttäjä painaa niistä. Sovelluksen sisällön ja toimintojen hierarkia olisi hyvä suunnitella selkeäksi ja sovelluksen tarkoitusta palvelevaksi. Näytön jakaminen eri osiin, käyttöliittymän värit ja asettelu pitäisi miettiä tarkoin. Silmät ja mieli hakevat koko ajan toistuvuutta, joten toiminnallisuuden ja käyttöliittymän tulisi olla samanlainen kaikkialla sovelluksessa. [45]

Sovelluksen pitäisi olla niin helppo käyttää, että siihen ei tarvitse ohjekirjaa. Sovelluksessa kannattaa hyödyntää intuitiota. Painikkeen näköinen objekti on painike, hyperlinkin näköinen sininen alleviivattu teksti on linkki ja niin edelleen. Tablet-laitteilla yleistyneet toiminnot olisi hyvä toteuttaa samalla tavalla myös sovelluksessa. Esimerkiksi nipistämällä näyttöä ulospäin suurennetaan (zoom) näkymää. [36]

5.5.2 Dynaaminen ulkoasu web-käyttöliittymässä

Internetin levitessä kaikkialle, sitä käytetään erilaisilla laitteilla, kuten matkapuhelimilla, kannettavilla tietokoneilla, tablet-tietokoneilla ja muilla siltä väliltä olevilla laitteilla. Pienemmille näytöille ei mahdu niin paljon sisältöä kuin suurille näytölle. Laitteissa voi olla myös käännettävä näyttö, jolloin kuvasuhde vaihtuu samalla erilaiseksi. Ei enää riitä, että tehdään web-sivu joka venyy näytön koon mukaan, vaan täytyy tehdä älykkäämpiä sivuja. Sivun komponentit asetellaan sivulle näytön koon mukaan tai jopa muutetaan koko navigointitapa esimerkiksi matkapuhelimia varten. Ei voida luottaa myöskään pelkkään näytön resoluutioon näytön koon määrittämiseksi. Selainikkuna voi olla auki tietokoneella vain osassa näyttöä eikä avattuna koko ruutuun. Lisäksi on hyödyllisempää tehdä yksi sivu, joka taipuu jokaiselle laitteelle kuin, että jos tekisi jokaiselle laitteelle oman sivustonsa. [1]

Käytännössä tällainen erikokoisiin näyttöihin taipuva web-sivusto voidaan toteuttaa monella tavalla, mutta yksi suosittu tapa on käyttää CSS-määritteitä, joilla voidaan määritellä tietyn kokoiselle näyttöalueelle oma CSS-tyyli. Tämä tarkoittaa sitä, että jos sivulle tulee matkapuhelimella, jonka leveys on esimerkiksi 480 pikseliä leveä ja CSS-tyyleissä on määritelty *@media screen and (max-width: 480px)*. CSS-määritteen mukaan sivulle ladataan 480 pikselin leveydelle luodut tyylit. Matkapuhelimen näkymässä esimerkiksi voidaan asetella kaikki sisältö allekkain ja määrätä leveydeksi enimmäisleveys 480 pikseliä. CSS-tyylien lisäksi sivulla voidaan tehdä muutoksia myös käyttäen Javascript-koodia. Koodilla voidaan tunnistaa myös selaimen leveys ja tehdä tarvittavia muutoksia sen mukaan sivun ulkoasuun. Navi-

gointi voidaan esimerkiksi tiivistää matkapuhelimen näkymässä pudotusvalikoksi, ja jos samalle sivulle tulee tietokoneella, valikko mahtuu kokonaan sivulle ja kaikki valinnat näkyvät. [1]

Sanomalehdessä teksti on yleensä aseteltu kapeisiin sarakkeisiin. Uusissa CSS3-määrittelyissä on sitä varten määrite *column-count*, jolla saadaan automaattisesti teksti sarakkeisiin. Määrittelemällä esimerkiksi *column-count:3* saadaan teksti kolmeen sarakkeeseen. CSS3-standardin keskeneräisyyden takia eri selaimet vaativat etuliitteen *column-count* merkintään. Näitä ovat esimerkiksi Mozillan *Firefox*-selaimessa *-moz*, Googlen *Chrome*-selaimessa *-webkit* ja niin edelleen. Lisäksi vanhemmat Internet Explorer versiot eivät tue näitä ominaisuuksia lainkaan. [16]

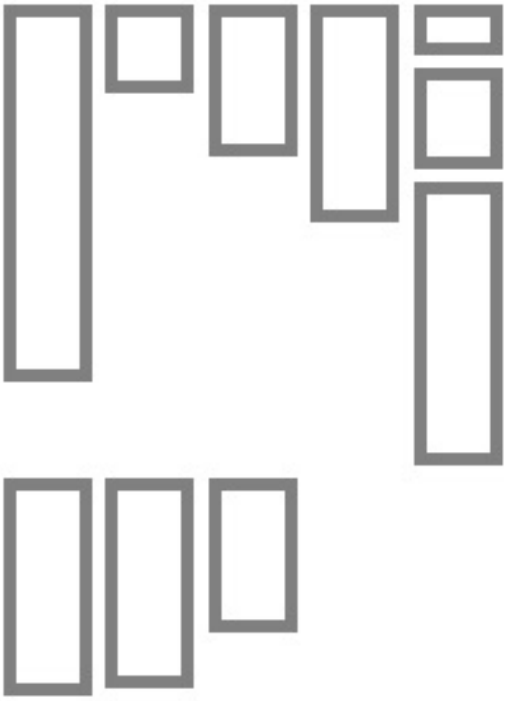
Käyttöliittymässä voidaan asettaa sivun eri komponentit kiinteisiin paikkoihin ja määrätä niille jokin tietty koko. Tällä tavalla voidaan tehdä kiinteitä ja ennalta määrättyjä näkymiä. Käyttöliittymä voidaan asetella myös dynaamisesti. Sivun eri elementtien ei tarvitse olla kiinteästi määriteltyjä. Sivulla voi olla esimerkiksi erikokoisia laatikoita ja niiden koot voivat vaihdella satunnaisesti. Ilman erillistä käsittelyä dynaaminen ulkoasu ei onnistu. Komponentit voivat olla täysin erikokoisia ja niiden väliin jää ylimääräisiä välejä. Elementit voidaan kuitenkin asetella käyttöliittymään dynaamisesti käyttäen Javascript-koodia. Jos sivulla on esimerkiksi erikokoisia uutislaatikoita, joissa on eri määrä tekstiä ja joissakin laatikoissa on kuva ja joissakin ei. Tällöin käyttöliittymä voisi olla suurin piirtein kuten kuvassa 5.3. Laatikkoelementit ovat vain yksinkertaisesti peräkkäin sivulla järjestyksessä vasemmalta oikealle kuten teksti. Laatikot ovat peräkkäin ja kun ei mahdu enempää, niin laatikot menevät seuraavalle riville. Laatikoiden väliin tulee tyhjää väliä, koska ne varaavat tilan tavallaan kuten tekstin rivit. Jos yksi laatikko on korkea, niin seuraava rivi on korkeimman laatikon jälkeen. Kuvassa 5.4 ovat samat laatikot dynaamisesti aseteltuna sivulle niin, että ne eivät varaa ylimääräistä tilaa. [26]

Dynaaminen asettelu voidaan tehdä käyttäen Javascript-koodia. Laatikoiden vievä tila lasketaan ja ne asetellaan sivulle järjestykseen tilan mukaan [26]. Jos sivun avaa pienemmän kokoisella selaimella, laatikot ovat järjestetty dynaamisesti eri kohtiin kuin leveällä näytöllä [26]. Dynaamiseen asetteluun on olemassa avoimen lähdekoodin Javascript-kirjastoja. Näitä ovat mm:

Packery `http://packery.metafizzy.co/`

jQuery Nested `http://suprb.com/apps/nested/`

Masonry `http://masonry.desandro.com/index.html`



Kuva 5.3: Erikokoisia laatikkoelementtejä web-sivussa.

Kuva 5.4: Laatikkoelementtejä web-sivussa dynaamisesti aseteltuna.

Isotope `http://isotope.metafizzy.co/index.html`

BlocksIt.js `http://www.inwebson.com/jquery/blocksit-js-dynamic-grid-layout-jquery-plugin/`

Freetile `http://yconst.com/web/freetile/`

Wookmark `http://www.wookmark.com/jquery-plugin`

Pinterest clone `http://pinterest-clone.heroku.com/`

Flex `http://jsonenglish.com/projects/flex/`

Grid-A-Licious `http://suprb.com/apps/gridalicious/`

6 Tablet-julkaisu Keskipohjanmaa lehdelle

Tämän tutkimuksen yhteydessä Keski-Pohjanmaan Kirjapaino Oyj:lle toteutettiin Keskipohjanmaa-lehdestä digitaalinen webissä toimiva HTML5 tablet-uutissovellus, johon tuotiin sanomalehden aineisto Doris-toimitusjärjestelmästä [5] hakemalla toimittajien sanomalehteä varten tekemät uutiset web-uutissovellukselle.

Uutissovelluksella pystyi lukemaan päivittäisen Keskipohjanmaa-lehden, sekä KP24.fi-verkkosivuilla [51] julkaistuja uutisia. Uutissovellusta varten suoritettiin tutkimusta, toteutettiin prototyyppejä sekä kaksi virallista julkaistua sovellusversiota. Sovellusta kehitetään edelleen ja odotettavissa on uusia versioita. Tablet-laitteille käytettäväksi tarkoitettu web-uutissovellus on osoitteessa `http://tablet.kp24.fi`. Sovellus on maksullinen ja sinne pääsee vain sisäänkirjautumalla. Sovellus kehitettiin käyttämällä ketterää ohjelmistokehitystä ja *scrum*-mallia. Sovellusta aloitettiin heti toteuttamaan ja sitä katselmoitiin tietyin väliajoin. Sovellus muuttui ja eli kehityksen aikana, kunnes se lopulta muovautui valmiiseen muotoonsa.

Uutissovelluksen data tuotiin julkaisujärjestelmästä. Datan tuonti ja palvelimen tekninen toteutus tehtiin Kosila Digimedian kehitystiimin toimesta. Itse lukusovellus oli pääosin tämän tutkimuksen tutkijan tekemä. Ensimmäisen version koodaamiseen osallistui myös muita Kosila Digimedian tiimistä. Toisen version käyttöliittymä ja tekninen toteutus olivat pääosin kokonaan tämän tutkimuksen tutkijan tekemiä. Sovelluksen ulkoasu ja tekninen toteutus tehtiin scrum-mallin mukaisesti. Kehittäjä suunnitteli ja toteutti sovelluksen, sen käyttöliittymän ja käytettävyyden, jota palavereissa käytiin yhdessä läpi ja muutettiin haluttuun suuntaan.

Sovelluksen suunnittelussa oli mukana Keskipohjanmaa-lehden toimitus. Tekninen toteutus tehtiin Keskipohjanmaa-konserniin kuuluvassa it-yksikkössä Kosila Digimediassa. Sovelluksen kehityksen aikana tehtiin yhteistyötä myös VTT:n tutkimusprojektin kanssa ja mukana olivat myös Aalto-yliopiston, Metropolia AMK:n ja Arena Partners Oy:n yhteyshenkilöt. Toisen sovellusversion yhteydessä Aalto-yliopisto ja Metropolia toteuttivat omat sovellusversiot samasta uutismateriaalista. Tavoitteena oli erilaiset käyttökokemukset ja näkökulmat. Näistä sovellusversioista ja meidän sovelluksesta tehtiin käytettävyystutkimus tampereen teknillisen yliopiston toimesta, jossa joukko ihmisiä testasi eri uutissovelluksia.

6.1 Suunnittelu

Päätimme lähteä toteuttamaan web-pohjaista uutissovellusta, koska sitä ei tarvinnut asettaa mihinkään sovelluskauppaan ja yksi sovellus toimisi periatteessa jokaisessa päätelaitteessa. Olimme käyttäneet ja lukeneet esimerkiksi Financial Timesin HTML5-sovelluksesta, joka oli menestynyt hyvin. Aloitimme tablet-lehden suunnittelun palaverilla, jossa päätettiin tehdä aluksi vain yksinkertainen web-sivu HTML5:lla, johon laitetaan uutiset listattuna valkoiselle pohjalle. Peruspohjassa uutiset olisivat lajiteltuna uutisosastoihin ja niiden välillä voisi navigoida helposti. Uutiset olisivat lajiteltu osastokohtaisiin sivuihin. Jokaisella osastosivulla olisivat kaikki kyseisen osaston uutiset. Toiseen osastoon voisi siirtyä jollakin helpolla tavalla, kuten pyyhkäisemällä sivulle. HTML-rakenteessa noudatettaisiin HTML5-määrittelyjä ja uutiset lisättäisiin *<article>*-elementtien sisään ja kuvat asetettaisiin *<figure>*-elementtien sisään. Kun olisimme saaneet toteutettua prototyypin, niin pystyisimme pitämään sitä pohjana suunnitellessamme sovellusta edelleen.

Sovelluksen kehitysympäristö oli *.NET*, palvelimen käyttöjärjestelmänä *Windows* ja kehitystyökaluna *Microsoft Visual Studio*. Palvelinpään koodikielenä käytettiin *C#*-kieltä ja käyttöliittymä toteutettiin *Javascript*-koodilla. Projekti luotiin *Visual Studiolla* ja toteutettiin *Web Forms*-sovellusprojektina. Sovellusprojektina *Web Forms* koostuu web-sivulla olevasta lomakkeesta ja sitä käsittelevästä palvelimella olevasta koodista [13]. Käytännössä *Web Forms* -web-sivulla koko HTML-sisältö on *<form>*-elementin sisällä ja kun käyttäjä esimerkiksi klikkaa painiketta, lomake lähetetään palvelimelle, jossa suoritetaan painikkeen klikkaukselle määritetty koodi ja palautetaan vastaus käyttäjän web-sivulle. Esimerkiksi jos halutaan, että kun painetaan painikkeesta *"Lisää"*, niin tekstikenttään *"Nimi"* ilmestyy jokin teksti. Tämä koodataan siten, että asetetaan painikkeelle *"Lisää"* klikkaus-tapahtuma. Klikkaus-tapahtuman koodiin kirjoitetaan, että aseta tekstikenttään *"Nimi"* vaikkapa teksti *"Hei"*. Palvelin sitten palauttaa web-sivulle HTML-koodin, jossa tekstikentässä on teksti *"Hei"*. Selain näyttää tämän palautteen käyttäjälle.

Päätimme aluksi lähteä kehittämään uutissovellusta pelkästään *iPadille*, koska testilaitteistoa oli jo olemassa ja sille oli luonnollisin lähteä toteuttamaan prototyyppiä. Tutkimme websisällön toteuttamista *iPadin Safari*-selaimelle lukemalla *iOS Developer Librarya* [9]. Selvisi, että iPadille voitiin luoda web-sovelluksia ja niille natiivia sovellusta matkiva oma pikakuvake kotinäyttöön. Pikakuvakkeesta web-sivu aukeaa koko ruutuun aivan kuten natiivi sovellus [8]. Selvisi myös, että websovelluksille voidaan tehdä käynnistyskuva [8]. Web-sivun oletuskokoa ja zoom-tasojen

määrää voitiin rajoittaa [7]. Koko ruudussa web-sovellus näytti enemmän natiivilta sovellukselta ja kun siihen lisättiin vielä käynnistyskuva oli vaikutelma aivan kuin natiivi sovellus. Mahdollisuuksien selviämisen jälkeen ei tarvinnut kuin suunnitella suorituskykyinen ja natiivilta sovellukselta näyttävä käyttöliittymä uutissovellusta varten.

6.2 Palvelinympäristö ja data

Prototyyppejä ja virallista julkaisua varten tarvittiin lehden uutismateriaali toimituksen käyttämästä *Doris*-julkaisujärjestelmästä. Kosila Digimediassa oli jo aiemmin tuotu uutisia julkaisujärjestelmästä Internetiin. Samaa kehitettyä tekniikkaa käyttäen ja soveltaen voitiin tuoda sanomalehden uutissisältö tablet-julkaisua varten käytettäväksi. Käytännössä julkaisujärjestelmästä saatava tieto oli xml-tiedostoja, uutiskuvia ja tekstitiedostoja. Julkaisujärjestelmän data prosessoitiin palvelimella olevalla ohjelmalla joka yö ja uutiset syötettiin sisään uutistietokantaan. Uutisdatan HTML-sisältöä muokattiin websivuille sopivaksi ja uutisia käsiteltiin niiden metatietojen perusteella.

Uutissisällölle toteutettiin myös RSS (Really Simple Syndication) [78]-syöte. RSS-syöte on XML-kielellä kirjoitettu tiedosto, joka sisältää RSS-määritelmän mukaisen tietosisällön [78]. Käyttäjä voi tilata syötteen johonkin RSS-lukijaan. Syöte voi olla päivittyvä, jolloin siitä ladattu uutissisältö päivittyy jatkuvasti. Keskipohjanmaalehden uutissisällöstä tehdyn RSS-syötteen pystyi avata ja lukea millä tahansa RSS-lukijalla ja näin voitiin testata lehden aineistoa monessa eri lukusovelluksessa.

Testasimme RSS-syötettä monella eri lukijalla ja siitä saimme ideoita kehitykselle. Testasimme syötettä *iPad*:llä mm. *Flipboard*:lla `http://flipboard.com/`, *Feedly*:llä `http://www.feedly.com/` ja *Nokia Lumia* puhelimella *Dark RSS Reader*:llä.

6.3 Prototyyppi 1

Ulkoasua varten tutkittiin alustavasti *iPadin* ulkoasua, jonka mukaan tehtiin perusvalikko sovellukseen. Valikko oli palkki sovelluksen ylälaidassa, jossa olivat sovelluksen toiminnallisuudet ja uutisosastot lueteltuna.

Ensin tehtiin HTML5-sivupohja ja tutkittiin siinä käytettäviä HTML5-elementtejä, kuten *header, article, section, nav, footer*. Sitten tutkittiin CSS3:n *transitionin* [98] toiminnallisuutta. CSS-*transitioilla* voitiin tehdä animointeja sivuun käyttäen vain se-

laimen toiminnallisuuksia, eikä sitä varten tarvinnut asentaa erillisiä lisäosia selaimeen. CSS-*transitiolla* voidaan esimerkiksi liikuttaa jotakin laatikkoa web-sivussa laidasta toiseen laitaan niin, että liike näkyy sivulla animoituna. Laatikon liike siis näkyy sivulla, kun se liukuu laidasta laitaan. Tämä CSS-ominaisuus toimii siten, että asetetaan esimerkiksi laatikko *div*-elementti web-sivun toiseen laitaan CSS-tyyleillä. Tyyleihin lisätään *transition* määreet, joilla voidaan määrätä animaation kesto ja mikä CSS-ominaisuus animoidaan. Tyylimääritysten jälkeen laatikko on sivun toisessa laidassa. Asetetaan sivulle esimerkiksi painike, jota klikkaamalla *Javascript*-koodi suoritetaan. Javascript-koodissa asetetaan laatikolle CSS-tyylillä sijainti vastakkaiseen laitaan, jolloin siirtymä animoituu automaattisesti.

Prototyyppiä varten tutkittiin miten web-sivulle saadaan tehtyä uutisosastoja, jotka ovat ikään kuin kokonaisia web-sivuja vierekkäin yhdessä sivustossa. Sovelluksen käyttöliittymä näyttää käyttäjälle siltä, että pyyhkäisemällä vaakatasossa voidaan selata sisältöä osastosivu kerrallaan. Toiminnallisuuteen etsittiin tietoa ja selvitettiin miten se saadaan toimimaan CSS-tyylien ja *Javascript*-koodin yhdistelmällä. Sivut olivat käytännössä *div*-elementtejä, jotka olivat asetettu CSS-tyylimäritteillä koko sivun levyisiksi ja vierekkäin toistensa kanssa. Sivujen animoidut liukumat voitiin toteuttaa CSS-*transitioiden* avulla. Kun käyttäjä koskee sormella näyttöön ja raahaa sormea näyttöä pitkin sivulle, sivu liikkuu sormen mukana ja kun on raahattu tarpeeksi pitkälle, sivu liukuu loppumatkan itsestään. Kuvassa 6.1 on esitetty osastosivujen rakenteen idea sovelluksessa.

Kuva 6.1: Osastosivut vierekkäin.

Sivujen tyyleissä käytettiin *position:absolute* CSS-määrettä [96], jolla voitiin aset-

taa näkyvä sivu alkamaan vasemmasta laidasta alkaen. Näkyvä sivu oli yksi uutisosasto-sivu. Sivu oli koko näytön levyinen. Muut sivut olivat piilotettu näkyvän sivun vasemmalle tai oikealle puolelle asettamalla CSS-määreellä *left:-100%* [96] tai *left:100%*. Muut sivut olivat vasemmalla tai oikealla sen mukaan missä kohdin osastosivuja käyttäjä oli menossa. CSS-määreellä *left* voidaan asettaa HTML-elementin vasemman laidan x-koordinaatti sivulla. Määre *left:100%* tarkoittaa, että HTML-elementti, joka oli tässä tapauksessa osastosivu, alkaa selaimen vasemmasta laidasta 100 prosenttia selaimen leveydestä eteenpäin. Vastaavasti määre *right:100%* tarkoittaa päinvastaista. Käytännössä osastosivu oli näkyvän näyttöalueen oikealla puolella piilossa jos CSS-määre oli *left:100%*. Kaikissa sivuissa kannatti käyttää *left*-määritettä eikä sekaisin *left* ja *right*.

Prototyypissä oli aluksi vaikeuksia rakenteen kanssa. Pystysuuntainen rullaus (scroll) ei toiminut, jos sivun avasi *iPadissä* koko ruudun web-sovelluksena. Sivut tehtiin uudestaan niin, että jokainen osasto oli omassa *div*-elementissään. Elementeillä oli jokin määrätty korkeus ja *divit* tehtiin rullautuviksi asettamalla CSS-ominaisuus *overflow:auto*. Edellä mainitulla CSS-ominaisuudella voidaan määrätä *diville*, että jos sen sisällön korkeus on korkeampi kuin itse *divin* korkeus, niin ylimenevä sisältö menee piiloon ja sen saa näkyviin rullaamalla *div*-elementtiä alaspäin. Normaalisti selain näyttää vierityspalkin *divin* oikeassa reunassa, joka ilmaisee kohtaa sisällöstä, mikä on näkyvissä. Tällä tavoin tehdyssä sivussa ilmeni suorituskykyongelmia siten, että *iPadillä* sivu renderöityi huonosti ja välkkyi valkoisena, jos sivua rullasi ylös tai alas. Ongelma ratkesi asettamalla kaikkiin osastoiden *div*-elementteihin CSS-määreen *position:relative*. Tämän jälkeen ilmeni muita suorituskykyongelmia. Vaihtaessa osastolta toiselle, haluttiin rullata kaikki muut näkymättömissä olevat osiot takaisin ylös niin, että jos osastolle palasi, sivu oli valmiiksi alkutilanteessa rullattuna ylös. Tämä taustalla tehty ylösrullaus tehtiin *Javascript*-koodilla ja se aiheutti tässä versiossa koko sivuston välkkymisen. Välkkymisongelman pystyi ratkaisemaan asettamalla CSS-määreen *transform:translate3d(0,0,0)*, jolla pakotettiin laitteistopohjainen suorituskyky sivun renderöinnissä [85]. Tämä CSS-määre vaati *iPadiltä* resursseja, ja jos sisällössä oli liikaa elementtejä, selain saattoi kaatua eikä sivua pystynyt käyttämään.

Yleisesti ottaen ensimmäistä prototyyppiä tehdessä selvisi, että perinteisellä tietokoneella web-sivu käyttäytyi eri tavalla kuin tablet-laitteessa ja erikoisuuksia ilmeni. Vaikka testasimme sivustoa työpöytätietokoneella samalla Internet-selaimella kuin *iPadissä* esimerkiksi *Safarilla*, sivu saattoi käyttäytyä eri tavalla *iPadissä*. Tablet-

laitteissa on rajoitetummat resurssit ja esimerkiksi *iPadissä* selain vain yksinkertaisesti sammui, jos selainta kuormitti liikaa.

6.4 Prototyyppi 2

Toisessa prototyyppissä keskityttiin enemmän vain testaamaan erilaista käyttöliittymää. Toteutimme prototyypin, jossa oli yhdellä pitkällä listalla koko lehden sisältö. Lehden saattoi lukea läpi rullaamalla kaikki uutiset ylhäältä alas. Tämä prototyyppi tehtiin sen jälkeen, kun olimme toteuttaneet RSS-syötteen lehden uutisaineistosta. Sivu toimi tabletissa, matkapuhelimessa ja perinteisellä tietokoneella. Uutiset oli lajiteltu osastoihin ja haluttuun osastoon pystyi hypätä suoraan valikosta. Sivusto toimi ainoastaan pystysuunnassa.

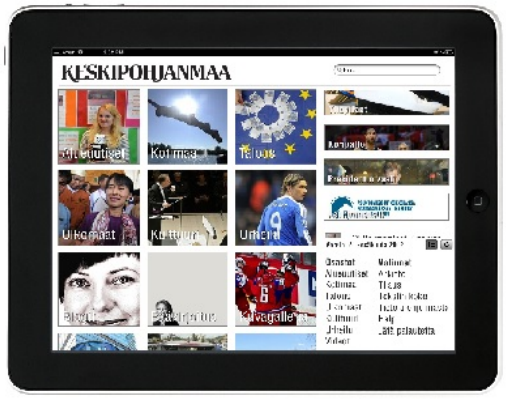
Sovellus oli yksi pitkä sivu, jossa navigointi oli sivun ylälaidassa olevassa palkissa. Rullatessa sivua alaspäin valikko-palkki seurasi mukana ja siitä pystyi avaamaan eri osastojen uutisia listalle. Jotta valikko-palkki saatiin pysymään mukana sivun rullautuessakin, käytettiin CSS-määrettä *position:fixed*, jolla voitiin määrätä HTML-elementti pysymään sille määrätyssä paikassa. Sivun oikeassa laidassa olivat toiminnot, kuten sivun päivitys ja fonttikoon kasvattaminen. Tehdessä tätä prototyyppiä selvisi, että *iPad*:n *Safari*-selain tuki CSS-määrettä *position:fixed* vasta 5.0 versiosta ylöspäin [31]. Sivulla olevista uutisista oli esillä pääkuva, kuvateksti, otsikot ja osa tekstistä. Uutisen koko tekstin sai auki napauttamalla *Lisää*-painiketta. Sivu ei pakottanut käyttäjää käymään läpi kaikkea sisältöä vaan avaamaan ainoastaan ne uutiset, mitkä lukijaa kiinnostivat.

6.5 Prototyyppi 3, virallisen version suunnittelu

Käyttöliittymä hahmoteltiin *Photoshop*-kuvankäsittelyohjelmalla kuvaksi ja suunniteltiin samalla minkälainen sovellus voisi olla. Rakenteeseen muodostui yksi tärkeä vaatimus. Rakenteen tulisi olla dynaaminen ja responsiivinen niin, että se sopeutuisi erikokoisiin näyttöihin. Uudesta ulkoasusta pidettiin palaveri, jossa käytiin läpi tablet-version mahdollista rakennetta ja minkälainen siitä tehtäisiin. Käyttöliittymässä päätettiin lähestyä *FlipBoard*-lukusovelluksen käyttöliittymää. Kuvissa 6.2 ja 6.3 on kaksi näistä suunnitelluista näkymistä.

Prototyypin etusivulla olisi *Windows 8*-tiilikäyttöliittymän tyylisiä laatikoita. Samanlaisia laatikoita oli myös *FilpBoard*-lukusovelluksessa. Etusivulla olisi kaksi osaa.

Oikealla olisi kapea palsta, jossa olisi haku, arkiston selaus ja valmiita uutisteemoja, joista pystyisi selata uutisia teemoittain, kuin selaisi teeman mukaista lehteä. Vasemalla olisivat osastolaatikot, joista käyttäjä voisi valita haluamansa uutisosaston. Uutisosasto-laatikossa olisi taustalla kyseiseltä osastolta pääuutisen kuva. Kuvan päällä laatikossa lukisi osaston nimi. Osaston pystyisi avata napauttamalla laatikkoa, jolloin osastosivu aukeaisi. Sovelluksessa olisi myös työkalupalkki kokoajan näkyvissä näytön alareunassa, jonka kautta voisi käyttää sovellukseen tehtyjä erilaisia toimintoja sekä pääsisi myös navigoimaan sovelluksessa. Kuvassa 6.2 on prototyypin etusivun käyttöliittymän hahmotelma tablet-laitteen näytön ollessa vaakatasossa.



Kuva 6.2: Prototyyppi 3 etusivu.

Kuva 6.3: Prototyyppi 3 osastosivu.

Osastolta toiselle pääsisi navigoimalla takaisin etusivulle ja valitsemalla toisen osaston tai avaamalla valikon näytön alareunassa olevasta työkalupalkista. Valikosta pääsisi navigoimaan muihin osastoihin napauttamalla listasta haluamaansa osastoa. Palkki suunniteltiin nimenomaan alareunaan, jotta sitä olisi helpompi käyttää. Palkin toiminnallisuuteen pääsisi käsiksi kurottamatta kättä näytön yläreunaan. Valikossa olisi myös päivityspainike sekä valikko, josta pääsisi selaamaan lehtiarkistoa.

Data olisi jaoteltu osastosivuihin. Rakenne olisi sivutettu niin, että osastosivuilla olisi koko osaston uutissisältö. Uutissisältö jatkuisi näytön ulkopuolelle ja sitä voi-

si rullata alaspäin kuten web-sivua. Uutiset olisivat osastosivuilla laatikoissa, joissa olisi lyhyt kooste uutisesta ja kuva, jos uutisessa sellainen on. Nämä eri korkuiset uutislaatikot olisivat prioriteetin mukaan järjestyksessä ylhäältä alas dynaamisesti aseteltuna kahteen sarakkeeseen. Kaikista ylimpänä olisi pääjuttu. Pääjuttu olisi suuremmassa laatikossa kuin muut osastolla olevat uutiset. Pääjutun kuva olisi iso ja sen alla olisi uutisteksti kahdessa palstassa. Kuvassa 6.3 on käyttöliittymähahmotelma osastosivusta tablet-laitteen näytön ollessa pystyssä. Vaakatasossa näyttöön mahtuu leveyssuunnassa enemmän tavaraa. Osaston pääjuttu olisi kahden sarakkeen levyinen ja oikealla olisi yksi sarake lisää, jossa olisi tärkeimmät uutiset.

Sivulla olevan uutisen pystyisi napauttaa auki, jolloin se aukeaisi omaan ikkunaan ikään kuin sovelluksen päällimmäiseksi oikeaan laitaan. Uutisikkuna peittäisi näytön vain osittain ja ylijäävästä tilasta näkyisi läpi taustalla uutisosastosivu, mistä uutisen on avannut. Käyttäjä hahmottaisi missäpäin sovellusta on menossa ja ei joutuisi hukkaan. Uutissikkunassa olisi ensin uutisen kuva ja sitten uutisteksti jaettuna kahteen palstaan. Uutisen fonttikokoa pystyisi suurentamaan ja pienentämään napauttamalla niille tehtyjä työkalukuvakkeita. Uutissovellus suunniteltiin alustavasti osittain maksulliseksi niin, että kun uutisen avaisi, niin uutisteksti olisi vain osittain luettavissa. Uutisia voisi ostaa yksittäin niin, että käyttäjä voisi ostaa luettavaksi vain ne uutiset, mitkä häntä kiinnostavat. Uutisikkunan pystyisi sulkemaan esimerkiksi napauttamalla taustalta läpinäkyvää osastosivua, mistä uutinen oli avattu.

Prototyypin tekstien fontiksi valittiin lähelle sama kuin *FilpBoard*-lukusovelluksessa. *FlipBoardissa* fontti oli *Helvetica condensed* [17]. CSS3:n ominaisuudella *@font-face* voitiin tuoda Internet-selaimeen ja web-sivuille käyttöön melkein mikä tahansa fontti [97]. Kehittäjien ei tarvitse enää käyttää pelkästään selaimissa yleisesti käytössä olevia niin kutsuttuja web-turvallisia fontteja.

Prototyypissä ei suunniteltu mainontaa erikseen. Sanomalehdessä olevat mainokset näytettäisiin mainoksille tehdyllä omalla sivulla. Sivu olisi samanlainen kuin osastosivut ja sinne pääsisi navigoimalla samalla tavalla kuin osastosivuille. Käyttäjän ei olisi pakollista katsoa mainontaa. Sanomalehden mainoksissa oli monesti tarjouksia tai muita ilmoituksia, joita käyttäjät saattoivat haluta lukea. Mainossivu olisi tavallaan ilmoitussivu, koska sivulla näkyisi myös esimerkiksi työpaikkailmoitukset.

Seuraavaksi aloitettiin tekemään ensimmäistä virallista julkaisua. Sovittiin, että aloitetaan yksinkertaisista ominaisuuksista. Ensin tehtäisiin sovelluksen rakenne

osastoittain ja sitten sinne tuotaisiin uutissisältö. Tekemisen ohessa tietyin väliajoin katselmoitaisiin sovelluksen edistyminen ja tehtäisiin tarvittaessa muutoksia ja lisäyksiä sovellukseen.

6.6 Ensimmäinen virallinen julkaisu

Aiemmin oltiin jo toteutettu prototyyppejä eikä niitä haluttu lähteä jatkojalostamaan, vaan aloitettiin puhtaalta pöydältä tekemään uutta. Heti alussa valittiin *jQuery Mobile* [88] *-Javascript* kirjasto sovelluksen rungoksi (framework). Kirjasto oli tarkoitettu nimenomaan kannettaville laitteille ja siinä oli valmiiksi toteutettuna monenlaisia ominaisuuksia ja käyttöliittymäulkoasuja, joita voitiin käyttää hyödyksi sovellusta tehdessä. Tavoite oli tehdä ensin yksinkertainen uutissovellus kohtuullisen nopealla aikataululla julkaisukuntoon, jonka jälkeen sitä kehitettäisiin lisää.

Sovellus julkaistiin ensimmäistä kertaa tuotantoon marraskuussa 2012 osoitteeseen `http://tablet.kp24.fi`. Versiossa oli vielä puutteita ja kankeutta, mutta sovellus kuitenkin julkaistiin ja sitä kehitettiin samalla lisää. Sovellus julkaistiin myös testipalvelimelle testipaikalle, jossa sovellusta voitiin testata tuotantoympäristössä.

6.6.1 Uutisen lukunäkymä

Uutiset tuotiin joka yö *Doris*-toimitusjärjestelmästä tablet-julkaisua varten tehtyyn tietokantaan. Sovellusta toteutettiin ketterin menetelmin, joka tarkoitti sitä, että sovellusta koodattiin ja sitten tietyin väliajoin katselmoitiin palaverissa ja sen mukaan sovellukseen tehtiin muutoksia. Ensimmäisessä kehitysvaiheessa uutisen tekstistä leikattiin 4 lausetta lyhyttekstiksi, joka näytettiin sovelluksessa uutislaatikossa uutisen kuvan lisänä. Uutisen pystyi sitten avaamaan erikseen niin, että uutisen koko teksti tuli näkyviin. Uutislaatikkoon lisättiin lyhyttekstin jälkeen painike, jossa luki *Näytä koko teksti*. Painikkeesta painamalla kyseisen uutisen koko sisältö avautui näkyviin uutislaatikkoon. Uutislaatikon korkeus venyi sisällön mukaan.

Jossain vaiheessa kehitystä siirryttiin erilliseen uutisikkunaan, jonka sai auki omaan ikkunanäkymään napauttamalla osastosivulla olevaa uutislaatikkoa. Uutislaatikoista otettiin pois *Näytä koko teksti*-painike. Painike ei ollut visuaalisesti tyydyttävä ja lisäksi jotkut uutiset saattoivat olla todella pitkiä, kun ne avautuivat uutislaatikossa kapeassa sarakkeessa. Uutissikkuna oli käytännössä <*div*>-elementti, joka oli asetettu laatikoksi ikäänkuin sovelluksen päälle. Laatikko ei peittänyt ko-

ko näyttöä vaan vasempaan laitaan jätettiin suikale, josta näkyi tummennettuna uutisosasto-sivu, josta uutinen avattiin. Vasempaan laitaan jätettiin suikale, josta näkyi osastosivu sen takia, että käyttäjä ei joutuisi hukkaan sovelluksessa. Kun uutisikkunan avasi, niin tiesi silti missäpäin sovellusta oli ja mistä pääsi takaisin osastosivulle.

Ikkunalle toteutettiin pyyhkäisytoiminto, jolla uutisikkunan pystyi sulkemaan. Ikkunan pystyi tavallaan pyyhkäisemään sivulle ja pois näkyvistä. Ikkunan pyyhkäisy toteutettiin käyttäen apuna *touchSwipe Javascript* -kirjastoa [18]. Uutisikkunan pystyi pyyhkäisemään vasemmalta oikealle pois näkyvistä. Ikkuna seurasi sormen sijaintia näytössä, kunnes ikkuna oli liikkunut tarpeeksi, jolloin se liukui itsekseen animoidusti loppumatkan pois ruudulta. Animoituun liukumiseen käytettiin CSS3:n *transition*-määreitä. Sama pyyhkäisytoiminnallisuus lisättiin myös mainosikkunaan, ilmoitus ja rivi-ilmoitusikkunaan, jotka olivat samanlaisia kuin uutisikkuna.

Sovellus ladattiin yhteydettömään tilaan niin, että päivän lehteä pystyi lukemaan ilman Internet-yhteyttä, kun sen oli kerran käynyt lataamassa. Myös kuvat ladattiin yhteydettömään tilaan. Kuvista pidettiin yllä listaa, josta kuvan pystyi hakemaan kuvan tiedostonimellä. Yhteydetön tila oli huomioitava uutisikkunaa avattaessa, kun uutiskuva ja uutissisältö avattiin ikkunaan. Uutissisältö oli kokonaisuudessaan HTML-koodissa piilotettuna, niin että sen pystyi hakemaan sieltä. Kuva haettiin yhteydettömän tilan kuvalistasta. Internet-yhteyden tila tarkastettiin ennen uutisikkunan avaamista ja mikäli kuva oli tallennettu yhteydettömään tilaan, niin se haettiin sieltä eikä Internet-yhteyden läpi palvelimelta.

Uutisikkunoiden avaukseen toteutettiin käytettävyyttä parantava toiminto siten, että kun uutisen avasi luettavaksi, niin avatun uutisen uutislaatikon tekstisisällön väri muuttui mustasta harmaaksi. Selatessa uutisosastosivua näki heti harmaasta väristä, että minkä uutisen oli lukenut.

6.6.2 Sovelluksen etusivu

Sanomalehdestä tuodussa uutisdatassa oli automaattisesti monenlaisia uutisosastoja, joita ei käytetty jokaisessa sanomalehdessä tai niitä käytettiin vain sisäisessä uutisten editoinnnissa. Uutistietokantaan asetettiin osastoille bitti (1 tai 0), joka kertoi käytetäänkö osastoa sovelluksessa vai ei. Jos osastolle oli merkattu 1, niin osasto näytettiin uutissovelluksessa ja jos 0, niin ei näytetty. Sovelluksen etusivulla näytettiin laatikoissa kaikki osastot, joita julkaisussa oli. Nämä osastot haettiin tietokan-

nasta, jossa oli merkattu, mitkä osastot ovat käytössä uutissovelluksessa. Osastolaatikoiden taustakuviksi haettiin kyseisestä osastosta tärkeysjärjestyksen mukaan ensimmäisen kuvallisen uutisen kuva. Uutisen kuva oli joko laajakuva tai pystykuva, jolloin kuva piti leikata neliön muotoiseksi. Kuvan rajauksessa kävi monesti niin, että kuva rajautui huonosti ja esimerkiksi kuvassa esiintyvän henkilön pää saattoi rajautui pois. Automaattinen rajaus aiheutti ongelmia esimerkiksi, jos osaston laatikkokuvassa henkilö katsoo sivulle ja sivulla on jokin toinen osaston kuvalaatikko. Toisessa osastokuvassa on jokin sopiva kuva, jolloin nämä kaksi uutislaatikkokuvaa muodostavat jonkin yhteisen merkityksen, mitä ei ole tarkoituksella haettu. Osasto-laatikoille asetettiin linkki, josta napauttamalla pääsee kyseiselle osastosivulle. Sovelluksen runkona toimivassa *jQuery Mobilessa* oli automaattisia animointeja sivusiirtymille. Animoitua sivusiirtymistä käytettiin, jotta sovellus olisi visuaalisesti hienompi.

Yhteydettömiä toimintoja selvitettäessä testattiin lataussivua sovelluksessa. Sovellukseen tultaessa selain ohjattiin ensin lataussivulle, jossa uusin lehti ladattiin ennen sen näyttämistä. Lataussivun kautta ohjautumista varten testattiin <*iframe*>-kehys-elementtiä, niin että pääsivuna oli <*iframe*>-kehys, johon ladattiin aina ensin lataussivu. Kehys piti asettaa selaimen täyteen leveyteen ja korkeuteen, jotta sovellus toimi koko ruudussa. Ylimenevä sisältö oli <*iframe*>-elementissä rullattavissa kuten normaalissakin web-sivussa. Selaimen osoite pysyi aina samana ja käyttäjä pystyi lisäämään kirjanmerkin ja ohjautui aina kuitenkin lataussivun kautta. Kehys ei toiminut *iPadissä Safari*-selaimessa samoin kuin yleisesti <*iframe*>-elementin odotetaan toimivan kaikissa muissa Internet-selaimissa. *iPadissä Safari* venyttää <*iframe*>-elementin korkeuden sen sisällön mukaan eikä siis toimi perinteisellä tavalla. Jotta sivu olisi saatu toimimaan, sen korkeus olisi pitänyt erikseen määrittää *Javascript*-koodilla <*iframe*>-kehyksen mukaan sivun latauduttua tai sen koon muuttuessa. Kehystä käytettäessä sovelluksessa sillä kehityshetkellä oleva alaosan työkalupalkki ei toiminut kuten sen piti.

Kehyksen tuomat ongelmat yritettiin ohittaa asettamalla <*iframe*>-elementti sivulle <*div*>-elementin sisään ja asettamalla <*div*>-elementin leveys ja korkeus selaimen leveyteen ja korkeuteen. Näytöstä ylimenevä osuus olisi rullattava. Jotta saatiin rullaus toimimaan <*div*>-elementille asetettiin CSS-määre *overflow:scroll;*, jolla sisältö rullautui, mikäli sisältö oli korkeampi kuin itse <*div*>-elementti. Vanhemmissa *iPadin* käyttöjärjestelmäversioissa <*div*>-elementin rullaus ei toiminut.

Tässä vaiheessa testattiin myös keinotekoista sivurullausta testaamalla sitä var-

ten suunniteltuja *Javascript*-kirjastoja, joilla voitiin matkia *iPadin* rullaustoimintoa. Ne olivat kuitenkin kankean oloisia tai erilaisia kuin käyttöjärjestelmän aito rullausominaisuus. Vanhempaa *iPadiä* käytettäessä web-sivulla olevaa <*div*>-elementtiä pystyi rullaamaan kahdella sormella, mutta ei yhdellä. Rullaustoiminto ei ollut natiivin kaltainen, vaan rullattava <*div*>-elementti liikkui vain kiinteästi sormen osoittaman matkan verran. Lisäksi harvat käyttäjät luultavasti ymmärtäisivät käyttää yhden sormen sijasta kahta sormea. Käyttöjärjestelmäversiosta *iOS 5* lähtien asettamalla *overflow:scroll;* -CSS määreen sekä *-webkit-overflow-scrolling: touch;* -CSS määreen <*div*>-elementtiä pystyi rullaamaan myös yhdellä sormella ja selain käytti siihen *iPadin* natiivia rullaustoimintoa, joka oli pehmeä ja miellyttävä [101]. Nämä ratkaisut olivat monimutkaisia ja web-sovellusta päätettiin olla käyttämättä aiemmin testatussa <*iframe*>-kehyksessä.

Sovellusta tehdessä pidettiin välillä palaveri, jossa käytiin läpi siihen asti toteutettu sovellus. Palavereissa päätettiin muutoksista ja lisäyksistä sovellukseen. Etusivun osastolaatikoihin haettiin kyseisen osaston pääjutun kuva. Artikkeli- ja mielipideosastoissa pääjuttuina olivat monesti toimittajien tekemiä juttuja, joissa oli heidän kasvokuviaan uutisen kuvana. Toimitus halusi, että näissä osastoissa ei haettaisi osaston pääjutun kuvaa ollenkaan etusivun osastolaatikkoon, vaan siihen tulisi jokin kiinteä kuva, joka esittää osastoa. Sama kiinteän kuvan toiminnallisuus tehtiin myös ilmoitukset-, rivi-ilmoitukset-sivulle. Lisäksi, jos uutisosastossa ei ollut yhtään kuvallista juttua, niin etusivun osastolaatikkoon tuli näkyviin osastoa vastaava ikonikuva sinisellä taustalla. Esimerkiksi kulttuuri-osaston ikoni oli teatterimaski ja talous-osaston ikoni oli kasa kolikoita. Lisäksi toimitus halusi mahdollisuuden vaikuttaa uutisosastojen uutisten järjestykseen. Muutos tehtiin uutisten tuotantojärjestelmässä *Doriksessa*, niin että toimittajat pystyivät halutessaan käydä asettamassa uutiselle prioriteettiarvon, jonka mukaan uutiset järjestäytyivät osastosivulla. Käytännössä uutisen prioriteetit olivat väliltä 1-9. Uutiset asetetaan oletuksena prioriteetille 5. Jos prioriteetti on 9, niin uutinen tulee sovelluksessa uutisosaston viimeiseksi. Mikäli prioriteetiksi asetettiin 1 tai 2, niin uutinen oli ensimmäisenä uutisosastolla. Prioriteetin mukaan tärkein kuvallinen uutinen oli uutisosaston pääjuttu. Lisäksi toimitus halusi, että uutisen esiotsikko asetetaan näkymään ennen pääotsikkoa ja sovelluksen logoon täytyi asettaa teksti *"TESTIVERSIO"*, joka kuvasti sitä, että sovellus oli vielä kehitysvaiheessa eikä lopullinen.

6.6.3 Yhteydetön tila

Sovellus suunniteltiin alusta lähtien käytettäväksi myös yhteydettömässä tilassa, kun lukulaitteessa ei ole Internet-yhteyttä. Päivän lehden pystyi lataamaan sovellukseen ja sen jälkeen lukemaan sitä ilman verkkoyhteyttä. Sovellus toimi ilman verkkoyhteyttä, vaikka sovellus oli käytännössä web-sivu. Tämän toiminnallisuuden pystyi tekemään käyttäen HTML5:n tuomia yhteydettömiä ominaisuuksia. Sovelluksessa käytettiin välimuisti-ilmoitusta (cache manifest) [99]. Välimuisti-ilmoitus toimii siten, että tehdään ilmoitus-tiedosto, johon voidaan määritellä, mitkä tiedostot web-sovelluksesta ladataan käyttäjän laitteelle yhteydettömään tilaan. Jotta yhteydetön tila saatiin toimimaan sivun HTML-koodirakenteen piti olla HTML5-otsikkomääritteellä määritelty. HTML5-sivun otsikkotieto tuli määritellä pelkästään käyttämällä *<!DOCTYPE html>*-merkintää koodin alussa. Kuvassa 6.4 on esimerkki välimuisti-ilmoituksesta [99]. Välimuisti-ilmoituksessa määriteltiin *CACHE:* otsikon alle tiedostot, jotka ladataan yhteydettömään tilaan käyttäjän laitteelle. Lisäksi voidaan määritellä sivut, joita ei ladata *NETWORK:* otsikon alle ja web-sivu jolle mennään, jos lataus epäonnistuu *FALLBACK:* otsikon alle. Tiedostojen polut ovat suhteellisia polkuja siitä kansiosta, missä itse välimuisti-ilmoitus sijaitsee.

```
CACHE MANIFEST
#Tähän jokin kommentti

CACHE:
web-sivu_joka_ladataan.html
tyylitiedosto.css

NETWORK:
web-sivu_jota_ei_ladata.html

FALLBACK:
/ /web-sivu_johon_mennaan_jos_ei_saada_ladattua.html
```

Kuva 6.4: Välimuisti-ilmoitus (cache manifest) esimerkki.

HTML-sivuun lisätään viittaus välimuisti-ilmoitukseen, kun se on valmis. HTML-sivun koodiin lisätään *manifest* atribuutti *<html>*-elementtiin. Tämä tapahtuu esi-

merkiksi näin <*html manifest="/offline.manifest"*>, jossa *offline.manifest* on tiedosto nimeltä *offline* ja jonka tiedostopääte on *.manifest*. Tiedosto on kansion juuressa. Välimuisti-ilmoitus menee helposti väärin, jolloin se ei toimi ollenkaan eikä sivulle tule mitään ilmoitusta epäonnistumisesta, ellet ole määrännyt *FALLBACK:* otsikon alle web-sivua, jolla käsittelet virhetilanteet.

Välimuisti-ilmoituksen toimintaa voi seurata esimerkiksi käyttäen Google *Chrome* -Internet selaimen kehitystyökaluja. Kehitystyökalut saa *Chromessa* auki painamalla näppäinkomentoa *F12*, josta aukeaa selaimen kehitystyökalu. Avautuvassa työkalussa on *Console*-välilehti, jossa näkyy tiedostojen latausprosessi. Mikäli tiedostojen latauksessa tuli jokin virhe, niin se näkyi *Console*-välilehdellä. Lisäksi voitiin asettaa *Javascript*-kuuntelija jota käyttäen voitiin ohjelmoida haluttu käsittely esimerkiksi silloin, kun tiedosto oli ladattu. Kuuntelijan avulla voitiin tehdä esimerkiksi edistymispalkki, joka näytti latauksen edistymisen. Kuuntelija voitiin asettaa seuraavanlaisella *Javascript*-koodilla [99]:

```
window.applicationCache.addEventListener('progress',
     function(event){
          Jokaisen tiedostonlatauksen jälkeen tullaan tänne.
      }, false);
```

Koodin suoritus siirtyy kuuntelijassa määriteltyyn funktioon heti, kun jokin välimuisti-ilmoituksessa määritelty tiedosto on ladattu. Kuuntelijaan voidaan ohjelmoida esimerkiksi toiminnallisuus, jossa asetetaan web-sivulle tieto jokaisen tiedoston latauksesta. Välimuisti-ilmoitus voi olla kiinteä tiedosto *cache.manifest*, joka sisältää tiedot ladattavista tiedostoista. Sovelluksen palvelinympäristö ei oletuksena tunnistanut *.manifest*-tiedostopäätteellä olevia tiedostoja, vaan se piti lisätä erikseen websovellukselle. Kiinteä välimuisti-ilmoitus ei ollut hyvä meidän sovelluksessa, koska sisältö vaihtui joka päivä. Kiinteää tiedostoa käytettäessä ilmoitusta pitäisi joka päivä käydä erikseen muokkaamassa.

Välimuisti-ilmoituksen voi toteuttaa myös dynaamisella tiedostolla siten, että määrittelee välimuisti-ilmoituksen joksikin skripti-tiedostoksi, joka palauttaa välimuisti-ilmoituksen. Käytännössä määrittely tehdään *HTML*-koodissa metatietona esimerkiksi näin <*html manifest="/jokinskripti.ashx"*>. Välimuisti-ilmoituksen tiedostonimi voi siis olla mikä tahansa ja eri koodikielillä toteutettu. Siihen voidaan laittaa esimerkiksi *php*-sovelluksessa jokin *.php*-tiedosto. Toteutetussa uutissovelluksessa välimuisti-ilmoitus toteutettiin dynaamisena tiedostona, siten että luotiin skriptitiedosto, joka palautti välimuisti-ilmoituksen sisällön. Välimuisti-ilmoitus oli erilainen

joka päivä, koska sovelluksessa ladattiin päivän lehden uutiset ja niiden kuvat yhteydettömään tilaan. Välimuisti-ilmoituksessa oli aina kyseisen päivän lehden sisältö ja käyttäjän selain latasi sen yhteydettömään tilaan.

Testatessa sovellusta huomattiin, että päivän lehdessä ladattavaa dataa oli suuri määrä. Välimuisti-ilmoitus muuttui joka päivä ja aina kun se muuttui, niin käyttäjän selain latasi kaikki siinä merkityt tiedostot, vaikka siinä saattoi olla joitakin kiinteitä muuttumattomia tiedostoja joukossa, esimerkiksi joitakin *Javascript*-koodikirjastoja ja CSS-tyylitiedostoja sekä niissä määriteltyjä kuvia. Toisaalta oli hyvä, että kaikki tiedostot päivittyivät esimerkiksi silloin, kun sovellusta oli päivitetty. Muutoin käyttäjälle olisi saattanut jäädä sovelluksen vanha versio niin kauan, kunnes selaimesta olisi tyhjennetty välimuistit käsin. Sovelluksessa oli erittäin paljon HTML-koodia ladattavana, koska sovellus oli toteutettu niin, että kaikki uutiset ja sisällöt ladataan sivulle HTML-koodina, eikä esimerkiksi Javascript-taulukkona, josta ne voitaisiin dynaamisesti tuoda sivulle. Tässä lataustavassa oli tutkittavaa ja koska eri selaimissa oli toteutettu erilaisia HTML5 yhteydettömän tilan ominaisuuksia, haluttiin selvittää mikä olisi sopiva sovellukselle.

Tiedon tallennusta yhteydettömään tilaan tutkittiin. Yritettiin selvittää mahdollisuuksia ja mikä olisi paras tapa web-uutissovellusta varten. Suurin ladattava osuus olivat uutisten kuvat. Kuvat olivat jo pienennetty siitä, minkälaisena ne tulivat toimitusjärjestelmästä, joten optimointi oli sen osalta toteutettu. Sovellusta tehdessä mietittiin myös, ladataanko kuvat vasta kun ne näytetään, eli kun käyttäjä menee uutisosastolle.

Välimuisti-ilmoituksen huonona puolena oli se, että kun käyttäjä tuli sovellukseen, sovellus näytti käyttäjälle automaattisesti eilisen päivän julkaisun, koska tämä löytyi tallennettuna yhteydettömään tilaan. Latausprosessi toimi siten, että käyttäjän saapuessa sivulle, selain tarkisti taustalla palvelimelta välimuisti-ilmoituksen. Jos ilmoitus oli päivittynyt, niin selain alkoi ladata taustalla web-sivun uusinta versiota yhteydettömään tilaan. Tällä välin käyttäjä ei nähnyt vielä uusinta julkaisua vaan edellisen päivän julkaisun. Selaimen ladattua uuden version taustalla käyttäjän piti päivittää web-sivua selaimellaan, jolloin uusi ladattu julkaisu avautui. Kaikki tapahtui taustalla eikä käyttäjälle tullut minkäänlaista ilmoitusta valmistumisesta. Tämä toiminnallisuus oli käytettävyyden kannalta huono ratkaisu. Sovellusta testattiin alueella, jossa oli huono matkapuhelinkuuluvuus ja näin ollen hidas Internet-yhteys. Lataus oli todella hidasta. Ensin sovellus ladattiin esiin ja sitten vasta sovellus alkoi lataamaan uuden julkaisun tiedostoja yhteydettömään tilaan. Käyt-

täjä joutui odottamaan kauan latausta ja käytännössä hitaalla Internet-yhteydellä lehden lataus oli mahdotonta.

Ratkaisuna latausongelmaan yritettiin luoda erillinen kevyt lataussivu, jossa uusi lehti ladattiin, ja jossa käyttäjä näki edistymispalkin uuden lehden latauksen tilasta. Tämä ratkaisu oli periaatteessa hyvä, mutta olisi tietysti luonnollisempaa päästä suoraan selaamaan uutta julkaisua siten, että julkaisu latautuisi taustalla yhteydettömään tilaan. Ongelmana erillisessä lataussivussa oli se, että käyttäjä saattoi tallentaa sovelluksen selaimensa kirjanmerkiksi vasta siinä vaiheessa, kun on siirrytty lataussivulta sovelluksen lukunäkymään. Käyttäjän avatessa kirjanmerkin ei mennäkään lataussivun kautta, vaan mennään suoraan lehtinäkymään, jolloin uusi julkaisu ei lataudu. *iPadissä* käyttäjä pystyy tallentamaan minkä tahansa web-sivun kirjanmerkiksi tai kotinäytön web-sovelluspikakuvakkeeksi. Ratkaisuna ongelmaan sovellus voittaisiin yrittää pakottaa ohjautumaan aina lataussivun kautta. Lataussivun kautta ohjaamista yritettiin mm. <*iframe*>-kehyksellä. Ohjaamisen tuomia ongelmia ei saatu selvitettyä niin, että sivu olisi toiminut kaikilla *iPad*-versioilla tyydyttävällä tavalla.

Sovelluksessa edettiin siten, että yhteydettömään tilaan ladatiin vain itse sovellus ja uutissisältö, mutta ei uutiskuvia. Kuvat näkyivät vain, jos laitteessa oli Internet-yhteys. Kuvien lataus tehtiin niin, että niitä ei asetettu normaalin websivun tapaan suoraan HTML-koodiin, vaan ne aloitettiin lataamaan yksitellen *Javascript*-koodin avulla vasta, kun itse HTML-sivu oli latautunut kokonaan. Websovellus latautui nopeammin selaimeen ja uutisia pystyi lukemaan jo ennen kuin kuvia oli ladattu ollenkaan. Nyt kun kuvia ei ladattu yhteydettömään tilaan, niin sovellus latautui nopeammin välimuisti-ilmoituksen mukaan. Tämä ratkaisu paransi käytettävyyttä hitaalla Internet-yhteydellä.

Välimuisti-ilmoitusta käytettäessä huomattiin, että ladattavasta HTML-sivusta pitää poistaa kokonaan käytöstä tavanomainen välimuistin käyttö. Tavallisesti selaimet tallentavat web-sivuja välimuistiin, jotta kun käyttäjä palaa sivulle, niin sitä ei tarvitse joka kerta ladata uudestaan. Tavanomainen välimuisti nopeuttaa selaamista. Web-sivu ei saa tallentua tavanomaiseen selaimen välimuistiin, jos käytetään yhteydettömän tilan välimuisti-ilmoitusta. Sivusto toimii sekavasti, jos tavanomaista välimuistia ei ota pois päältä, koska sivu saattaa ladata uuden julkaisun selaimen välimuistista eikä palvelimelta. Tällöin käyttäjälle saattaa latautua vanha tai osittain vanha julkaisu uuden sijasta. Sivustolta pakotettiin pois perinteinen välimuistitus lisäämällä HTML-otsikkotietoja sekä metatietoja HTML-koodiin.

Sovelluksesta yritettiin edelleen tehdä luettava myös ilman verkkoyhteyttä ja latoustoiminnallisuutta yritettiin parantaa. Koska web-sivun täytyi aina ladata päivän uutissisältö välimuisti-ilmoituksen mukaisesti ennen kuin uutta lehteä voitiin näyttää käyttäjälle, web-sovellukseen tehtiin ominaisuus, joka näytti päivitys-painikkeen käyttäjälle, kun uusi lehti oli ladattu. Lisäksi luotiin edistymispalkki, joka osoitti latauksen edistymisen tiedosto tiedostolta. Toiminnallisuuden periaate oli samankaltainen kuin *iPadillä* olevissa natiiveissa uutissovelluksissa. Periaate oli se, että sovelluksen sai auki, mutta uusin lehti täytyi ladata sovellukseen lukemista varten. Käytännössä uutissovellukseen tultaessa sovellus alkoi heti lataamaan uutta sisältöä ja sovelluksessa näytettiin yläreunassa edistymispalkki latauksen vaiheesta. Kun lataus oli valmis, niin edistymispalkin paikalle ilmestyi painike, jossa kehotettiin päivittämään sivua painamalla painikkeesta. Sivun päivityttyä uusin sisältö latautui sivuun.

Sovelluksesta ladattiin yhteydettömään tilaan kaikki sisältö poislukien kuvat, koska ne veivät niin paljon tilaa ja lataaminen kesti kauan. Kuvien tallennusta tutkittiin ja yritettiin selvittää, miten myös kuvat saataisiin ladattua käyttäjäystävällisesti yhteydettömään tilaan. Ratkaisuna tähän yritettiin käyttää jotakin muuta HTML5:n yhteydettömän tilan ominaisuutta kuin välimuisti-ilmoitusta. Tutkimuksen jälkeen selvisi, että kuvat pystyi muuntamaan tekstimuotoon käyttäen HTML5:n <*canvas*>-elementtiä. Kuva saadaan tekstimuotoon lataamalla se ensin *Javascript*-koodilla kuvaolioksi. Lisätään sitten luotu olio <*canvas*>-elementtiin, josta kuva sitten muunnetaan *base64*-koodattuun tekstimuotoon käyttäen *toDataUrl()*-funktiota [92]. Tämä kuvasta muodostettu teksti voidaan sitten tallentaa johonkin HTML5:n yhteydettömän tilan varastoon. Varastot eivät ole vielä standardeja, joten eri selaimissa on toteutettu erilaisia varastoja. Tekstimuotoista kuvaa voidaan käyttää suoraan HTML-koodissa kuvaelementissä asettamalla tekstimuotoinen kuva kuvaelementin osoittamaksi kuvaksi <*img src="tekstimuotoinenkuva"*/>, eli samalla tavalla kuin normaali kuva <*img src="kuva.jpg"*/>.

Toisista HTML5:n yhteydettömistä varastoista ensimmäisenä testattiin *localstorage*-varastoa ja siitä selvisi heti, että sinne ei mahdu oletuksena tarpeeksi dataa. *localstorage* on yhteydettömän tilan muisti, johon voi tallentaa avainsanoja ja niille vastaavia arvoja tekstimuodossa. Arvoja voi hakea varastosta avaimen perusteella. Muisti on toteutettu kaikilla uusilla tunnetuimmilla selaimilla. *localstorage* sallii tallentamaan enintään 5 megatavua dataa [71]. Tallennustilaa tarvittiin enemmän, joten *localstorage* ei sopinut sovellukseen kuvien tallennusta varten.

Aiemmin mainitun *localstoragen* lisäksi on olemassa *Web SQL Database*, johon voi tallentaa tietoa *SQL* tyylisten tietokantakyselyjen avulla. *Web SQL Database* on tuettu kaikissa muissa yleisimmissä selaimissa paitsi *Internet explorerissa* ja *Firefoxissa*. Tietokantaan voi tallentaa rajattoman määrän dataa. Selaimissa on asetettu jokin oletusarvo enimmäistallennusmäärälle. Jos datan määrä ylittää oletusarvon, niin selain kysyy käyttäjän lupaa lisätilaa varten ja näin voidaan tallentaa enemmän dataa. [71]

Kolmas tallennustapa on *IndexedDB*, johon voidaan tallentaa rajaton määrä dataa. Selain kysyy käyttäjältä lupaa lisätilalle, kun selaimen asettama oletus enimmäistallennusmäärä ylittyy. *IndexedDB:n* voidaan tallentaa tekstin lisäksi *Javascript*-olioita. Kaikki uusimmat selaimet tukevat tätä tallennustilaa paitsi *iPadissä* oletusselaimena oleva Applen *Safari*. [62]

Yhteydettömän tilan tallennustavan valinta oli tässä vaiheessa vaikeaa, koska ei ollut olemassa tallennustapaa, joka olisi ollut tuettu kaikilla yleisimmillä selaimilla. Sovellukseen toteutettiin kuvien tallennus *IndexedDB:n* avulla. Sen jälkeen käytettiin *IndexedDBShim-Javascript* -kirjastoa [64], joka muuntaa *IndexedDB:n* automaattisesti *WebSQL:ksi*, jos selain ei tue *IndexedDB:tä*. Ensin tehtiin oma *IndexedDB*-toteutus datan tallennusta varten, mutta se ei soveltunut *IndexedDBShim*-kirjaston kanssa käytettäväksi. Sitten käytettiin valmista *IndexedDB* kirjastoa nimeltä *db.js* [72], jolla sovellus saatiin toimimaan oikein. Tallennus toimi myös Applen *Safari*-selaimella *iPadissä*. *db.js* oli suorituskyvyltään hidas, kun ladattiin kuvia vastastosta. Kirjaston lähdekoodia muutettiin nopeammaksi, jotta lataus nopeutui ja sitä pystyi käyttämään.

Sovelluksen yhteydettömän käytössä esiintyi ongelmia, vaikka selaimen perinteinen välimuisti oli asetettu pois käytöstä. Osastojen kuvat saattoivat tulla välimuistista päivittymättä. HTML-sivuun asetetut metatiedot eivät vaikuttaneet tässä, koska kuva ladattiin suoraan palvelimelta erikseen eikä HTML-tiedoston mukana. Tämä välimuistitus ohitettiin asettamalla *.NET*-sovelluksen kyselyiden käsittelyyn *Global.asax* -tiedostoon otsikkomääritykset, joilla välimuisti otettiin pois päältä kaikista palvelimelle tulevista kyselyistä.

6.6.4 Lehtiarkisto, haku- ja tapahtumat-sivu

Sovellukseen toteutettiin myös aiempien päivien lehtien lukutoiminto. Käyttäjä pystyi valitsemaan päivämäärän mukaan, minkä päivän lehteä halusi lukea. Ensin yritettiin toteuttaa myös aiempien päivien lehtien lataus yhteydettömään tilaan. Tämän toteuttamisessa oli ongelmia. Koska ladattavaa oli paljon ja käyttäjän laitteeseen tulisi paljon tiedostoja, päädyttiin siihen, että vain uusimman lehden voi ladata

yhteydettömään tilaan. Tällöin käyttäjän laite ei täyttyisi datasta eikä aina tarvitsisi ladata lehtien vanhempia versioita ennen lukemista. Edellisten päivien lehdet olisivat luettavissa Internet-verkon välityksellä, kuten normaalit web-sivut. Arkistosivu oli yksi osastosivu sovelluksessa. Sivulla näkyivät allekkain arkistosta kaksi viikkoa taaksepäin. Listauksessa näkyi päivämäärä ja viikonpäivä. Päiväystä napauttamalla sivu päivittyi ja siirryttiin historiasivulle, joka toimi vain Internet-yhteyden ollessa päällä. Historiasivulle ladattiin valitun päivän julkaisu.

Lisäksi sovellukseen toteutettiin tapahtumat-osastosivu. Tapahtumat-sivulla esitettiin päivän tapahtumat Kokkolassa. Tapahtumat olivat lajiteltuna alkamisajankohdan mukaan. Tapahtumat haettiin `http://tapahtumat.kpk.fi/` tapahtumakalenterin materiaaleista. Tapahtumat tuotiin ensin sivulle vain taulukkona. Tapahtumat-sivulle suunniteltiin oma miellyttävä ulkoasu, jossa tapahtumat olivat tunneittain ryhmiteltynä. Uudessa ulkoasussa päivän tapahtumat olivat laatikoissa joiden reunassa luki tunti ja laatikon sisällä olivat kaikki sillä tunnilla alkavat tapahtumat.

Sovellukseen toteutettiin myös hakusivu. Hakusivulla pystyi hakemaan vanhoja uutisia hakusanoilla. Haulla haettiin uutisia tietokannasta vertaamalla uutisten otsikoita sanahaulla. Hakutulokset tulivat sivulle allekkain listaan ja hakutulos-uutisen pystyi napauttamaan auki uutisikkunaan.

6.6.5 Testaus ja suunnittelu

Web-sovellusta testattiin käytännössä lukemalla päivän lehti normaaliin tapaan ja käyttämällä sovellusta sille tarkoitetulla tavalla. Tässä on esimerkki sovelluksesta löydetyistä kehityskohteista jossakin vaiheessa kehitysprosessia:

- Sivujen oli oltava oikean kokoisia, niin että ei tarvinnut kokoajan uudelleenlaskea sivun kokoa *Javascript*-koodilla.
- Koodia piti optimoida suorituskykyisemmäksi.
- Jos uutisesta puuttuu otsikko, niin oli yritettävä muodostaa se uutistekstin ensimmäisestä lauseesta. Tässä oli huomioitavia hautajaisilmoitusten arkaluonteisuus.
- Uutisosastot eivät menneet aina oikein. Joskus uutisia on väärässä osastossa tai uutiset olivat joissakin ylimääräisissä teemaosastoissa, joita ei ollut tarkoitus käyttää sovelluksessa.

- Ladataan yhteydettömään tilaan vain viimeisin lehti ja aiemmat lehdet jätetään toimimaan vain verkossa.
- Kun uutinen on avattu luettavaksi, niin muutetaan uutislaatikon teksti harmaaksi. Harmaasta tekstistä näkee, mitkä uutiset on jo lukenut.
- Mietittävä, että ladataanko kuvat dynaamisesti vasta sitten kun mennään uutisosasto-sivulle missä kuvaa käytetään?
- Mietittävä tehdäänkö sovellukseen ominaisuus, jossa kun uutinen avataan niin voisi selata uutisia yksitellen, valitsemalla *Seuraava* tai pyyhkäisemällä sivulle.
- Uutisosastosta on voitava siirtyä toiseen osastoon pyyhkäisytoiminnolla.
- Sovelluksen fontti on muutettava joksikin luettavammaksi.

Testatessa huomattiin, että tapahtumat-osastosivu ei jostain syystä aina toiminut oletetulla tavalla, kun sovellusta käytti *iPadillä Safari*-selaimella. Näin kävi varsinkin, jos käytti web-sivua *Safarin* koko ruudun web-sovelluksena. Palvelinympäristössä *Windowsin* käyttämä *IIS* (Internet Information Services) ei joissakin tilanteissa tunnistanut *Safari*-selainta, jolloin esimerkiksi *Javascript*-koodia ei suoritettu ollenkaan ja sovellus ei toiminut lainkaan. Vian pystyi tarvittaessa korjaamaan lisäämällä *.NET* web-sovelluksen konfiguraatiotiedostoon (*web.config*) merkinnän *<browserCaps userAgentCacheKeyLength="256"/>* ja tarvittaessa asettamalla ongelmasivun palvelinpään koodiin erikseen *Safari*-selaimen tunnistuksen ja asettamalla keinotekoisesti sen ominaisuuslistan korkeammalle tasolle ja näin saaden sivun toimimaan oikein [86]. Vika johtui siitä, että palvelin ei tunnistanut *Safari*-selainta ja määritti sen alemman tason selaimeksi, jossa *Javascriptin*-koodin suoritusta ei sallittu [55].

Tablet-sovellusta testattiin käytännössä käyttämällä sovellusta ja lukemalla päivittäin päivän lehti. Käytössä huomattiin, että kun laitteen kääntää pystysuuntaan, uutislaatikoiden pitää ryhmittyä eri tavalla, koska pystysuunnassa ei mahdu olemaan kolmea uutissaraketta järkevästi. Sovellusta muutettiin niin, että kun tablet-laitteen kääntää pystysuuntaan, osaston pääjuttu on koko sivun levyinen ja sen alla on kahdessa sarakkeessa loput osaston uutiset. Osaston pääjuttu oli suurempi laatikko kuin muut uutislaatikot. Pääjuttu oli kahden sarakkeen levyinen ja siinä oleva kuva yritettiin levittää koko uutislaatikon leveyteen. Jos pääjutun kuva oli kuvasuhteeltaan pystykuva, kuvasta tuli aivan liian korkea. Kuvalle lisättiin tarkistus, jos kuvasuhde on pystysuuntainen, määrätään kuvalle enimmäiskorkeus, että se

ei pääse venymään liian suureksi. Sama tarkistus lisättiin myös uutisen lukunäkymään uutiskuvalle.

Sovellusta testattiin myös vanhimmalla *iPad 1:llä*. Vanhemmat *iPadin* käyttöjärjestelmän *iOS* versiot ei tue natiivisti <*div*>-elementtien rullausta. Käytännössä sivun osia ei voi tällöin rullata erikseen vaan koko sivua rullataan. Tämä toiminto voitiin korjata CSS-määreellä *-webkit-overflow-scrolling:touch;*. Testatessa selvisi, että vaikka CSS-määre oli asetettu, niin *iPad 1:llä* rullaus ei toiminut siltikään suoraan sovelluksessa.

Sovelluksessa huomattiin testatessa sellainen virhe, että joskus sovellus ei alkanut lataamaan kuvia ollenkaan. Kuvat aloitettiin lataamaan yksitellen sivulle sitten, kun sivu oli latautunut ja tallentunut yhteydettömään tilaan. Joskus tässä yhteydettömän tilan tallennuksessa sattui jokin tuntematon virhe, jolloin latauksen loppumista odottava *Javascript*-kuuntelija ei lauennut eivätkä kuvat päässeet aloittamaan latautumista. Koodia muutettiin niin, että tapahtuvat virheet käsitellään ja latausta yritetään uudestaan mikäli virhe sattuu. Näin koodin suoritus ei jäänyt kesken mihinkään, vaan kuvat latautuivat aina, vaikka virheitä sattuisi.

6.6.6 Aineisto

Sovellukseen tuotiin uutisten lisäksi myös lehdessä olevat mainokset ja ilmoitukset. Palvelimella oleva muunnin ajoi päivän lehdestä mainokset, rivi-ilmoitukset ja muut ilmoitukset kuvina sovellusta varten. Rivi-ilmoituksille ja ilmoituksille luotiin omat osastot, jotka näkyivät sovelluksessa omina osastosivuinaan. Mainoksissa oli lehdessä julkaistuja mainoksia. Rivi-ilmoitukset ilmestyivät lehdessä joka keskiviikko ja ne ajettiin myös uutissovellukseen keskiviikkoisin. Mainosten lisäksi lehdessä oli muita ilmoituksia, muun muassa työpaikkailmoituksia sekä eri järjestöjen ilmoituksia.

Sovelluksessa oli huomioitava, että toimitusjärjestelmästä tuodussa uutismateriaalissa saattoi olla joukossa uutisia, joissa ei ollut otsikkoa. Lisäksi uutismateriaalissa saattoi olla uutisia, joissa oli toimituksen sisäistä viestintää. Sisäistä viestintää saattoi olla uutisen esiotsikkossa, jossa oli asetettu ohjeita toimitukselle. Tämä uutismateriaali ei käytännössä ollut siis täysin sama kuin valmiissa painetussa sanomalehdessä, koska painetusta sanomalehdestä poistettiin edellä mainitun kaltaiset sisäiset kommentit ja viestintä. Toiveiden mukaisesti uutissovellus haluttin toimimaan täysin automaattisesti. Uutisen otsikon puuttuessa yritettiin otsikko luoda

automaattisesti uutisen ensimmäisestä lauseesta. Otsikon louhiminen uutismateriaalista osoittautui hankalaksi, koska tekstin seassa oli HTML-elementtejä ja niitä ei voinut vain yksinkertaisesti poistaa. HTML-elementit piti hallitusti poistaa ja sulkea XML-kielen määritysten mukaisesti. Jos HTML-elementti meni epästandardiksi tai niin sanotusti rikkoutuneeksi, rikkoi se myös koko web-sovelluksen käyttöliittymän. Joissakin tapauksissa automaattisesti luodusta uutisotsikosta tuli liian pitkä tai otsikossa saattoi olla jokin todella pitkä sana. Pitkä sana saattoi venyä sovelluksen käyttöliittymässä sille määrätyn laatikon reunojen yli rikkoen näin käyttöliittymää. Tästä johtuen tutkittiin, millä tavoilla liian pitkä sana voitaisiin katkoa kahdelle riville. Pitkä sana voitiin pilkkoa määräämällä uutislaatikolle CSS-määre *word-wrap:break-word;*, joka katkaisi sanan siitä kohdasta, missä sana ylitti laatikon reunan. Toinen vaihtoehto oli asettaa uutislaatikoihin tavutus, jolla sana tavuttui toiselle riville.

Uutiset käytiin läpi palvelimella yksitellen ennen kuin ne palautettiin käyttäjälle HTML-koodina. Uutiset aseteltiin osastosivulle vuorotellen kolmeen eri sarakkeeseen sen mukaan, kuinka paljon uutisia oli kussakin sarakkeessa. Teknisesti uutisille muodostettiin korkeuskerroin sen mukaan, onko uutisessa kuva tai onko uutisen lyhytteksti pitkä vai lyhyt ja onko uutisen kuva laajakuva vai pystykuva. Uutiset asetettiin sitten kolmeen eri sarakkeeseen niin, että uutinen asetettiin sarakkeeseen, jossa oli pienin korkeuskerroin. Tällä saatiin lajiteltua uutiset sivulle kohtuullisen tyydyttävästi. Joissakin tapauksissa uutisosastossa saattoi jokin sarake olla pitempi tai jokin sarake oli lyhyempi kuin muut. Tällä saatiin kuitenkin kohtuullisen suorituskykyisesti uutiset lajiteltua sarakkeisiin.

Web-sovellusta testattiin joka päivä uusimman lehden uutismateriaalilla. Mainokset ja ilmoitukset tulivat sovellukseen kuvina ja selvisi, että jotkin näistä kuvista olivat liian suuria. Latausaika oli pitkä ja lisäksi *iPadissä Safari*-selain ei osannut näyttää liian isoja kuvia. Jos kuva oli liian iso, sivulla näkyi vain tyhjä rikkinäistä kuvaa esittävä laatikko kuvan paikalla [11]. Ilmoitussivu muutettiin uutisosastosivujen tapaan myös kolmeen sarakkeeseen. Mainoksen sai auki samanlaiseen ikkunanäkymään kuin uutiset.

Mainosten lisäksi myös rivi-ilmoitukset tulivat kuvina sovellukseen. Rivi-ilmoituksille oli oma osastosivu, jossa ilmoitukset olivat kolmessa sarakkeessa samoin kuin muualla sovelluksessa. Rivi-ilmoituksia ei voinut ryhmitellä dynaamisesti eri sarakkeisiin, koska jokainen ilmoitus kuului johonkin rivi-ilmoitus -kategoriaan. Rivi-ilmoitukset ryhmiteltiin ensin kategorioittain ja sitten kategoriaryhmät asetet-

tiin sarakkeisiin. Ryhmille asetettiin otsikoksi kategorian nimi. Rivi-ilmoitukset pystyi avaamaan erilliseen ikkunanäkymään. Tällä tavalla jatkettiin yhtenäistä käyttötapaa sovelluksessa. Ilmoitusten kategoriaryhmät olivat usein erikokoisia. Esimerkiksi *myydään*-kategoriasta tuli usein yksi todella suuri ryhmä ja se teki yhdestä sarakkeesta todella korkean. Tätä ei lähdetty korjaamaan mitenkään, koska aiemmin oli hylätty kokonaan dynaaminen sivurakenne. Tyydyttiin yksinkertaiseen lajitteluun ja hyväksyttiin, että joskus yksi sarake saattoi olla todella korkea.

Sovellukseen ladattiin päivittäinen sanomalehden sisältö ja sen mukana myös mainokset. Sovelluksen kehityksen aikana palvelimella tapahtuneen ongelman vuoksi mainoksia ei tullut muutamaan päivään. Vikaa selvitettiin ja mainokset ajettiin uudestaan. Monen päivän mainokset ilmestyivät kaikki kerralla yhden päivän lehteen. *iPadin Safari*-selain ei kestänyt tätä kuormaa, vaan selain kaatui kun kuvia ja sisältöä oli liikaa. Tämä oli huomioitava toteutuksessa siten, että vaikka mainoksia tai uutissisältöä ei ollut tullut, niin näytettiin silti vain viimeisimmän lehden sisältö ja viimeisimmät mainokset. Huomioitavaa oli myös se, että sovellus latautui yhteydettömään tilaan, ja jos joku käyttäjä olisi ehtinyt lataamaan kaatuvan sovelluksen yhteydettömään tilaan, niin hän ei olisi saanut millään uutta julkaisua ennen kuin olisi tyhjentänyt selaimensa välimuistin asetusten kautta. Käytännössä kaatuvaa sovellusta ei saa päästää latautumaan käyttäjien laitteille.

Uutissovellukseen määriteltiin alustavasti joitakin sovelluksen omia mainospaikkoja, joihin voisi myydä mainoksia. Mainokset ladattiin yksitellen sovellukseen luotuihin <*iframe*>-kehyksiin sitten, kun itse sovellus oli latautunut. Mainosten lataaminen toimi siten, että mainospaikoille asetettuihin <*iframe*>-kehyksiin asetettiin osoitteeksi erikseen luotu mainos web-sivu. Kehykseen ladattavaan url-osoitteeseen asetettiin parametri, jonka mukaan mainossivu palautti mainospaikkaa vastaavan mainoksen. Kehykset poistettiin käytöstä, koska testatessa huomattiin, että sivun eri osien rullauksessa oli ongelmia jos mainokset olivat <*iframe*> kehyksissä.

Mainospaikat otettiin pois kehyksistä ja muutettiin niin, että ne ladattiin *Javascriptilla* mainospaikoille <*div*>-elementtien sisään. Tämä toimi siten, että *Javascriptissä* oli lista mainospaikkojen url-osoitteista samoin kuten aiemmassa ratkaisussa, jossa käytettiin <*iframe*>-kehyksiä. Tässä ratkaisussa ladattiin url-osoitteen palauttaman web-sivun HTML-koodi erillisellä *Javascript*-kyselyllä ja saatu koodi lisättiin mainospaikkana toimivan <*div*>-elementin sisään.

6.6.7 iPad web-sovellus

Web-sivun pystyy lisäämään *iPadissä* kotivalikkoon web-sovellukseksi [8]. Kotivalikko on käytännössä sama kuin *Windows*-käyttöjärjestelmässä oleva työpöytä. Jotta web-sivu käyttäytyisi *iPadissä* erillisenä web-sovelluksena eikä perinteisenä kirjanmerkki-linkkinä web-sivulle, HTML-koodissa täytyy olla metatieto sitä varten [8]. Metatiedolla *<meta name="apple-mobile-web-app-capable" content="yes"/>* voidaan piilottaa normaali Internet-selaimen käyttöliittymä jolloin web-sivu näyttää natiivilta kokoruudun sovellukselta [8]. Metatieto astuu voimaan, kun sovelluksen lisää kotinäyttöön pikakuvakkeeksi ja avaa sovelluksen siitä.

Natiivia sovellusta voitiin matkia myös muilla metatiedoilla. Web-sovellukselle voitiin asettaa ikoni, joka näkyi kotinäytössä pikakuvakkeena samalla tavalla kuin laitteeseen asennetuilla natiiveilla sovelluksilla. Tämä toiminnallisuus voitiin toteuttaa lisäämällä *<link rel="apple-touch-icon" href="/ikoni.png"/>* metatieto HTML-koodiin [8]. Avatessa koko näytön web-sovellusta, se avautui koko näyttöön natiivin sovelluksen tapaan, mutta siinä saattoi välähtää aiemmin avattu sivu ja sitten valkoinen tausta. Valkoinen näkyi kunnes web-sovellus oli latautunut eli käytännössä kun web-sivu oli ladattu palvelimelta. Tätä varten sovellukselle voitiin asettaa myös käynnistyskuva. Kuva näkyi web-sovelluksen alussa sillä aikaa kun selain latasi web-sivun näkyviin. Kuva voitiin asettaa metatietoihin *<link rel="apple-touch-startup-image" href="/startup.png"/>* -määritteellä [8]. Ikoneista ja käynnistyskuvista voitiin määritellä useita versioita HTML-koodin metatietoihin eri näyttöresoluutioita varten.

Sovellukseen tuotiin uutislaatikot osastokohtaisiksi sivuiksi. Uutislaatikot aseteltiin ensin osastokohtaiselle sivulle ja sitten ne ladottiin *Javascript*-koodilla dynaamisesti järjestykseen käyttäen *Masonry-Javascript*-kirjastoa [25]. Uutislaatikot asetettiin ensin keskitetysti sivulle ja sitten ladottiin järjestykseen. Käytettäessä *Masonry*-kirjastoa sovelluksessa, siitä tuli hidas *iPadissä*, varsinkin käytettäessä koko ruudun web-sovelluksena. Tästä johtuen *Masonry*-kirjasto jouduttiin hylkäämään sovelluksesta.

Sovellus oli käytännössä yksi suuri web-sivu, joka oli jaoteltu uutisosasto-sivuihin. Uutisosastoa katsoessaan käyttäjä näki vain osaston osuuden koko web-sivusta. Käytettäessä *Masonry*-kirjastoa dynaaminen asettelu jouduttiin tekemään koko sovelluksen uutissisällölle, eikä ainoastaan yhdelle HTML-sivulle. Kuormaa oli liian paljon ja sovelluksen rakenteen vuoksi käyttöliittymä oli jumissa noin 1-3 sekuntia latoessaan uutiset dynaamisesti sivulle. Uutislaatikoiden korkeus vaihteli laati-

kon sisällön mukaan. Dynaaminen ladonta hylättiin, koska se ei tässä sovelluksessa ollut suorituskykyinen, vaan se aiheutti käyttöliittymän jumittamista. Jotta sovellus olisi saatu suorituskykyisemmäksi, olisi rakenne täytynyt suunnitella alusta asti uudestaan, eikä sitä enää tässä vaiheessa aloitettu tekemään.

Dynaamisen asettelun sijaan uutiset aseteltiin osastosivulle kolmeen sarakkeeseen. Sarakkeille jaettiin näyttöalueen leveys tasan prosentteina. Sarakkeet venyivät ja kutistuivat näytön koon mukaan. Osastosivun alussa oli kahden sarakkeen levyinen pääjuttu isommalla ja sen oikeassa laidassa oli kolmas sarake. Kaksi muuta saraketta alkoivat pääjutun alapuolelta. Uutiset lajiteltiin sarakkeisiin yksinkertaisella logiikalla palvelimella ja näytettiin sivulla. Sivua ei saatu täydellisesti tasapainoiseksi, mutta tällä kuitenkin saatiin tyydyttävä perustaso uutisosastosivulle, eikä suorituskyky ollut ongelma.

Sovellus oli tarkoitettu kosketusnäyttöjä varten. Jos käyttäjä raahasi sivua sormellaan esimerkiksi rullatakseen sivun alareunaan tai avatakseen uutislaatikon napauttamalla, sivulla oleva teksti saattoi maalautua aktiiviseksi samoin, kuin jos hiiren kursorilla maalaa tekstin kopioimista varten. Tämä ominaisuus häiritsi käyttöä ja selvisi, että tekstin maalauksen pystyi estämään Internet-selaimissa asettamalla CSS-tyylimääritteen *user-select:none;* haluamalleen HTML-elementille [63]. CSS-määrite ei ollut virallisen standardin mukainen ja eri selaimia varten täytyi asettaa alkuliite esimerkiksi *Safarille -webkit-user-select:none;* ja vastaavasti *Firefoxille -moz-user-select:none;* ja niin edelleen [63].

Sovellusta kehitettiin ensisijaisesti *iPadiä* varten. Sovelluksessa ilmeni kehityksen aikana paljon suorituskykyongelmia. *iPadissä* oletuksena oleva *Safari*-selain yrittää säästää suorituskyvyssä, jotta web-sivut toimisivat paremmin eikä laite kaatuilisi. Web-sivuista renderöitiin vain näkyvissä oleva osuus ja näytön ulkopuolella oleva sisältö käsiteltiin ja renderöitiin vasta, kun käyttäjä selasi sivua eteenpäin. *iPad* yritti pihistellä muistinkäytössä ja laitteistokiihdytyksessä aina kun mahdollista. Jos uutissovelluksessa vaihdettiin osastolta toiselle, niin osastosivu välkkyi valkoisena ensin ja sitten näkyi normaalisti. Tämä ongelma häiritsi selvästi käyttöä.

Safari ei renderöi näytön ulkopuolisia elementtejä. Sitten kun siirrytään toiseen sivuun tai rullataan näytöllä esiin jokin elementti, niin sivu välkähtää valkoisena noin sekunnin ajan. Tämä on käyttöä ärsyttävä ongelma. Sen sai korjattua asettamalla jonkin tai kaikki näistä CSS-määreistä *-webkit-transform: translate3d(0, 0, 0);*, *-webkit-backface-visibility: hidden;*, *webkit-perspective: 1000;* ja *webkit-transform: translateZ(0);* halutulle HTML-elementille. Aluksi kaikille elementeille asetettiin edellä

mainittuja CSS-määreitä. Sovellus ei enää välkkynyt ja se toimi enemmän natiivin sovelluksen omaisesti. Näiden määreiden käyttö kuitenkin aiheutti sen, että selain kaatuili satunnaisesti jos selain kuormittui liikaa. Käytännössä kaatuminen tarkoitti sitä, että *Safari*-selain vain sulkeutui ihan yhtäkkiä ja hävisi näkyvistä kesken käytön. CSS-määreitä täytyi käyttää harkiten ja vain sillä hetkellä käytössä oleviin ja näkyviin HTML-elementteihin. Määre piti ottaa pois aina kun sitä ei tarvittu. Sovelluksessa piti seurata jatkuvasti, millä osastosivulla ollaan, ja lisätä sekä poistaa CSS-määreitä sen mukaisesti. Lisäksi CSS-määreet aiheuttivat *jQuery Mobile*-kirjastossa erikoisia ominaisuuksia kuten se, että sovelluksessa sillä hetkellä ollut alavalikko ei enää toiminut oikealla tavalla.

Web-sovelluksia tehdessä täytyi ottaa huomioon *iPadin* ja siinä olevien selainten puutteet. Toisaalta oli myös huomioitava muiden valmistajien tablet-laitteiden suorituskyvyt ja selainominaisuudet. *iPadin IOS*-käyttöjärjestelmässä oli ominaisuus, jossa kun selainta rullasi ylös tai alas, niin mitään *Javascript*-koodia ei voinut suorittaa rullauksen aikana. Sivusto ikäänkuin jäätyi paikalleen rullauksen ajaksi.

Perinteisellä HTML-sivulla voitiin asettaa jokin elementti CSS-määreellä *position:fixed* pysymään tietyssä kohdassa näyttöä, vaikka sivua rullattaisiin johonkin suuntaan. Tämä CSS-määre ei toiminut *iPadissä*, vaan rullauksen mukana seuraavaksi haluttu elementti jäi paikalleen sivun rullauksen ajaksi ja hyppäsi sitten paikalleen kun rullaus oli ohi. *jQuery Mobile*-kirjastossa oli toteutettu valikko, joka pysyi paikallaan vaikka sivua rullasi. Sovellukseen toteutettiin alavalikko käyttäen tätä toiminnallisuutta.

Vanhemmilla *iPadin* käyttöjärjestelmillä ei pystynyt rullaamaan <*div*>-elementtejä web-sivussa. Web-sivulle ei voinut siis laittaa sisäisiä rullattavia elementtejä. Jopa <*iframe*>-kehykset aukeavat kokonaan sivulle auki sisällön korkeuden mukaan. Ongelmaan oli olemassa *Javascript*-kirjastoja, jotka loivat keinotekoisen rullausominaisuuden sivulle. Sovelluksessa testattiin erilaisia *Javascript*-rullaus -kirjastoja. Kirjastolla voitaisiin hoitaa web-sivulla osittaiset rullaukset, kuten esimerkiksi sovelluksen uutisikkunan sisällön rullaus. Sovelluksessa testattiin mm. *iScroll-Javascript* kirjastoa ja todettiin, että vaikka kirjastot toimivat, niin ne eivät ole yhtä hyviä kuin natiivi rullaus. Osittaista sivun rullausongelmaa selvitettiin, koska se ilmeni myös uudemmassa *iPadissä* ja ilmeni, että jos web-sivulla on <*iframe*>-kehys elementtejä, niin samalla sivulla ei toimi edellä mainittu sivun elementtien rullaus. Lopulta päätettiin poistaa <*iframe*>-kehykset sivulta, jotta natiivi rullaus toimisi kaikilla *iPad* -versioilla.

iPadille tehdylle web-sovellukselle voitiin asettaa käynnistymiskuva, joka näkyi heti kun sovelluksen avasi. Kuva näkyi niin kauan kunnes web-sivu latautui. Kun sovellusta testattiin silloisella uusimmalla *iPad 3:lla* huomatiin, että *Retina*-näytön suuresta resoluutiosta johtuen, käynnistyskuvasta täytyi olla oma isompi versio sovelluksen metatiedoissa [12]. Käynnistyskuvasta tehtiin kaksi eri versiota vanhemmalle ja uudemmalle *iPadille*. Lisäksi käynnistyskuvasta piti tehdä oma versio vaakatasoon ja pystytasoon. Metatieto käynnistyskuvasta ja sovellusikonista asetettiin HTML-koodissa *header*-elementin sisään. Käynnistyskuva määriteltiin esimerkiksi seuraavalla tavalla *<link rel="apple-touch-startup-image"href="img/ipad-landscape-retina.png"media="screen and (min-device-width: 481px) and (max-device-width: 1024px) and(orientation:landscape) and (-webkit-min-device-pixel-ratio: 2)"/>*. Metatiedoille oli siis oma merkintä vaakatason ja pystytason kuville sekä eri resoluutioille.

Etsiessä tietoa web-sovellusten kehityksestä *iPadille*, sille löytyi simulaattoreita. Simulaattorit eivät vastanneet todellista ympäristöä, joten niitä ei käytetty erityisemmin mihinkään kehityksessä. Simulaattoreista mainitaan kuitenkin kaksi *ipadgeek* [79], joka löytyy osoitteesta `http://ipadpeek.com/` ja *alexw ipad2* [94], joka löytyy osoitteesta `http://alexw.me/ipad2/#!safari`.

6.6.8 Valikko

Sovellukseen tehtiin erikseen valikko, josta pääsi navigoimaan suoraan jollekin osastolle. Valikko tehtiin käyttäen sovelluksessa yleisesti käytettyä *jQuery Mobile*-kirjastoa. Kirjaston avulla pystyi tekemään sovelluksen alareunaan työkalupalkin, johon osastot asetettiin vierekkäin painikkeina. Valikkopalkki pysyi aina näytön alareunassa vaikka näyttöä rullasi eri suuntiin ja vaihteli osastosivuja. Käytännössä valikkopalkki oli ahdas ja epäkäytännöllinen, koska osastot eivät aina mahtuneet valikkoon. Osastot olivat palkissa vierekkäisinä painikkeina, joissa luki osaston nimi.

Alavalikko hylättiin, koska se oli ahdas, vei turhaa tilaa näytössä ja se oli epäkäytännöllinen. Sovellukseen kehitettiin oma räätälöity valikko, jossa ei käytetty valmista *jQuery Mobile*-kirjastoa. Kirjasto ei tarjonnut valmista rakennetta halutulle valikolle ja siksi päätettiin koodata oma valikko *Javascriptilla*. Valikkoon otettiin mallia *Financial Timesin* HTML5 web-sovelluksesta, jossa oli piilovalikko. Valikko näkyi sivun ylälaidassa tummana laatikkona, josta napauttamalla valikko aukesi. Uutissovellukseen kehitettiin samankaltainen valikko. Näytön oikeaan yläreunaan tehtiin painike, jossa luki *Osastot*. Myöhemmin valikon tekstiksi vaihdettiin *Valik-*

ko, joka oli kuvaavampi. Painiketta napauttamalla valikko liukui esiin ylhäältä näytön ulkopuolelta. Valikossa osastot olivat rivissä osastolaatikoina. Tilan riittämättömyys ratkaistiin siten, että mikäli osastolaatikot eivät mahtuneet näkymään kerralla valikossa, osastolista jatkui näytön ulkopuolelle piiloon. Osastolistaa pystyi rullaamaan vaakatasossa sormella pyyhkäisemällä. Osastoa napauttamalla sovelluksessa siirryttiin valitun uutisosaston sivulle.

Uutisosastosivut olivat asetettu niin, että niitä pystyi pyyhkäisemällä siirtymään seuraavaan osastoon. Tämä pyyhkäisytoiminto oli ristiriidassa tehdyn osastovalikon kanssa. Jos valikossa pyyhkäisi vaakatasossa vasemmalta oikealle tai toisinpäin, niin koko osastosivu vaihtui eikä valikossa oleva osastojen selaus toiminut odotetulla tavalla. *jQuery Mobile*-kirjastoon täytyi asettaa intervalli pyyhkäisystä. Tietty määrä pikseleitä piti pyyhkäistä ennen kuin osastosivu vaihtuu. Intervalli voitiin asettaa koko sovelluksen laajuisesti asettamalla arvolle *horizontalDistanceThreshold* haluttu pyyhkäisyn pikseliarvo. Asettamalla tämä pikseliarvo suureksi voitiin estää osastosivun vaihto, ellei pyyhkäisy ole tarpeeksi leveä. Valikon pyyhkäisytoiminto saatiin tällä korjauksella toimimaan.

Valikkoon lisättiin myös painike uloskirjautumista varten ja päivityspainike, jolla sivua pystyi päivittämään esimerkiksi ongelmatilanteessa. Valikossa oli myös toiminto, jolla pystyi poistamaan yhteydettömään tilaan ladatut tiedostot.

6.6.9 Siirtyminen osastojen välillä

Sovellukseen tehtiin osastojen välinen siirtyminen pyyhkäisy-toiminnolla. Käyttäjä pystyi pyyhkäisemään ruutua sormella vasempaan tai oikeaan ja viereinen uutisosastosivu liukui esiin sen mukaisesti. Tätä varten *jQuery Mobile*-kirjastossa oli olemassa valmis toiminnallisuus, jossa *Javascript*-kuuntelija odotti käyttäjän pyyhkäisyä. Pyyhkäisyn tapahtuessa tunnistettiin, mihin suuntaan käyttäjä pyyhkäisi ja sen mukaan vaihdettiin osastosivua seuraavaan tai edelliseen. Uusi osasto liukui animoidusti näkyviin. Animointi toimi kirjastossa siten, että selain rullasi ensin automaattisesti näkyvissä olevan sivun ylös ja sitten animoi siirtymisen toiselle sivulle. Tämä ominaisuus ei ollut käytettävyyden kannalta kovin kaunis, koska käyttäjän piti seurata selaimen hyppäys ylös ja sitten siirtymä toiselle osastosivulle. *jQuery Mobile*-kirjaston lähdekoodia muokattiin siten, että sivu ei hyppää ensin ylös ennen siirtymää toiselle osastolle. Osastosivu, jolta lähdettiin, piti kuitenkin myöhemmin rullata ylös, jotta kun käyttäjä palasi osastosivulle, niin se oli rullattuna takaisin

ylös. Koodiin asetettiin *Javascript*-kuuntelija, jolla jokaisen osastosivun vaihduttua rullattiin aiempi osastosivu takaisin ylös.

jQuery Mobile Javascript-kirjastossa oleva valmis sivusiirtymän toiminnallisuus ei ollut kovinkaan intuitiivinen, koska käyttäjän pyyhkäistessä sivua, se ei liikkunut sormen mukana. Käyttäjän piti pyyhkäistä ensin paikallaan olevaa sivua ja vasta tämän jälkeen osasto vaihtui. Sivu havaitsi käyttäjän pyyhkäisyn ja vasta sitten sivu liikkui käyttäjän pyyhkäisyn suuntaan uutisosaston vaihtuessa. Tämä ei ollut käytettävä eikä tyylikäs, koska sivupohja ei liikkunut sormen liikkeen mukana. Sivusta ei siis saanut tavallaan tarrattua kiinni. Käytettävyyden kannalta tämä oli huono ratkaisu ja se antoi sovelluksesta kankean kuvan. Sovellukseen ohjelmoitiin oma pyyhkäisytoiminto, jolla osastosivut liukuivat sormen liikkeiden mukana ja sivusto oli sujuvampi käyttää. Pyyhkäisyyn käytettiin *Swipe-Javascript* -kirjastoa [14]. Jotta uusi pyyhkäisytoiminto saatiin toimimaan, sivuston rakennetta täytyi muuttaa. Sivustossa käytetty *jQuery Mobilen*-sivunvaihto täytyi ohittaa kokonaan. Uusi pyyhkäisytoiminto paransi käytettävyyttä huomattavasti. Jotta sujuvuus saatiin aikaan, sovelluksessa täytyi poistaa käytöstä kaikki osastosivut, jotka eivät olleet käytössä tai seuraavana pyyhkäisyvuorossa. Suorituskyky parani, koska selaimen ei tarvinnut käyttää resursseja piilossa oleviin osastosivuihin.

6.6.10 Suorituskyky ja optimointi

Sovelluksen suorituskykyä yritettiin jatkuvasti optimoida mahdollisimman kevyeksi ja pieneksi kehitysympäristön, käytettävien *Javascript*-kirjastojen ja kohdelaitteiden rajaamassa hiekkalaatikossa. Sovellusprojektissa käytetyssä *.NET:n Web Forms* sovellusympäristössä oli käytössä *ViewState*, joka lisäsi HTML-koodin määrää sovelluksessa. HTML-koodin lisäys lisäsi ladattavan sisällön määrää ja päätettiin jättää *ViewState* kokonaan pois sovelluksesta.

ViewState on ominaisuus *.NET*-ympäristössä, jolla voidaan piilottaa HTML-koodin sekaan tietoa. *ViewState* säilyttää web-sivun lomakkeen tilan sivupäivitysten ja tietojen lähetyksen jälkeen [13]. Tieto kulkee siis HTML-koodin mukana, mutta ei näy käyttäjälle. *ViewStateen* tallentuu esimerkiksi sivulla olevien tekstikenttien, painikkeiden ja komponenttien tila, toiminnallisuus sekä niihin syötetty data. *ViewStateen* voidaan myös erikseen koodilla asettaa tietoa. Tieto voi olla esimerkiksi joitakin parametreja, kuten jonkin tietokantarivin *id* tai vain jokin tieto joka halutaan välittää palvelimen päähän.

Nämä parametrit lähetetään palvelimelle usein selaimen *URL*-osoitteessa, mutta ne voidaan lähettää palvelimelle myös piilotettuna *ViewStateen*. Esimerkkinä voidaan käyttää henkilölistausta web-sivulle. Oletetaan, että web-sivulla on lista henkilöiden nimiä ja halutaan tehdä toiminnallisuus, jossa käyttäjä voi klikata henkilön nimeä. Klikkaamalla henkilön tiedot avautuvat web-sivulle. Oletetaan, että web-sivun osoite on osoite on `http://www.testi.fi`. Henkilön tietojen avaaminen voidaan toteuttaa esimerkiksi niin, että listassa olevien henkilöiden nimet ovat perinteisiä hyperlinkkejä. Linkkien *URL*-osoitteisiin voidaan asettaa henkilön nimi tunnisteeksi esimerkiksi näin `http://www.testi.fi/?henkilo=Matti`. Tunnisteeksi pelkkä *Matti* ei välttämättä ole hyvä, jos *Matteja* on monia. Tällöin linkkiin voidaan asettaa tietokannassa oleva henkilön tunniste *ID* esimerkiksi näin `http://www.testi.fi/?id=123`. Käyttäjä klikatessa henkilön nimeä, selain ohjataan hyperlinkissä määriteltyyn osoitteeseen `http://www.testi.fi/?id=123`. Sivun latautuessa tarkistetaan palvelimen koodissa, että onko osoitteessa parametria *id*. Jos parametri löytyy, niin ladataan tietokannasta *id*:llä henkilön tiedot ja näytetään ne web-sivulla. *ViewStatella* voidaan toteuttaa sama toiminnallisuus siten, että *id*-parametri, voidaan asettaa koodissa *ViewStateen* ja sivun latautuessa palvelimen koodissa hakea se sieltä. Tällöin *URL*-osoite pysyy samana, eikä siinä ole mitään ylimääräisiä parametrela.

Suorituskyvyn parantamiseksi uutisten kuvia aloitettiin lataamaan vasta sitten, kun sovellus oli ladattu yhteydettömään tilaan. Näin nopeutettiin yhteydettömään tilaan lataamisnopeutta, koska ei ladattu samalla myös kuvia. Tutkittiin mahdollisuutta, voiko sovelluksen ladata yhteydettömään tilaan vasta, kun uutisten kuvat ovat latautuneet. Tällä tavalla sovellusta olisi päässyt mahdollisimman nopeasti käyttämään kuvineen ja yhteydettömään tilaan lataus olisi suoritettu taustalla. Välimuisti-ilmoituksella toteutettua yhteydettömän tilan latausta ei voinut erikseen viivästyttää, joten lataus oli suoritettava heti sivulle tultaessa. Uutiskuvien viivästetty lataus myös nopeutti web-sivun alkulatausta, koska ei tarvinnut ladata kuvia ja näin sovelluksen sai nopeammin auki. Tosin, vaikka sovelluksen sai auki, käyttäjä joutui kuitenkin odottamaan uuden julkaisun latausta ja sitten avata sen päivittämällä sivua.

Sovelluksen nopeutta tutkittiin ja yritettiin optimoida edelleen. Sovelluksen uusien ominaisuuksien ja kehittämisen myötä siitä oli tullut hidas. Itse web-sivu latautui hitaasti. Lopulta sivun latauksesta oli tullut niin hidas, että sitä ei voinut käyttää. Hitauden syytä yritettiin selvittää tutkimalla koodia. Tutkimusten jälkeen sel-

visi hitauden aiheuttaja. Hitaus aiheutui koodauslogiikan virheistä. Sovellus toimi kehityksen tässä vaiheessa teknisesti siten, että ensin ladattiin etusivulle kaikki uutisosastot laatikoihin. Sitten aloitettiin lataamaan uutisosastosivuja yksitellen. Jos osastossa ei ollut sille päivälle uutisia, niin uutisosastolaatikko käytiin poistamassa etusivun laatikoiden joukosta. Joka kerta, kun osasto käytiin poistamassa etusivulta, jouduttiin käymään läpi jokainen osasto ja lataamaan uudestaan jäljelle jääneet osastot etusivulle. Tämä ristiintarkistus ja -lataus aiheutti sen, että sivu latasi pahimmillaan 40 sekuntia. Logiikkaa muutettiin siten, että etusivulle ladattiin suoraan vain ne osastot, joissa oli uutisia ja ristiintarkistusta ei tarvinnut tehdä.

Sovellusta optimoitiin myös minimoimalla CSS-tyylitiedosto ja *Javascript*-koodi. Minimoiminen tarkoittaa sitä, että pienennetään merkkien määrää ja yritetään esittää jokin asia mahdollisimman pienellä määrällä merkkejä. Tämä nopeuttaa koodin suoritusta, sivu renderöityy nopeammin ja *Javascript* latautuu nopeammin. Minimoimista varten on olemassa työkaluja, joihin voi asettaa minimoitavan koodin tai CSS:n ja sitten generoida siitä minimoidun version. Tällaisia työkaluja ovat esimerkiksi CSS-tyylejä varten *CSS Compressor* `http://www.csscompressor.com/` ja *Javascript*-koodia varten *Closure Compiler* `http://closure-compiler.appspot.com/home`. Käytännössä koodi esimerkiksi *Javascript*-koodista poistetaan kaikki välilyönnit ja rivinvaihdot niin, että koko koodi on yksi pitkä rivi. Jotkut työkalut jopa uudelleen nimeävät funktiot ja niiden parametrit, jotta merkkien määrä vähenee.

Uutisosastosivuilla olevia uutislaatikoiden ulkoasua päivitettiin. Uutiskuvien kuvatekstit tuotiin uutiskuvien yhteyteen. Kun uutisten kuvatekstit tuotiin sovellukseen näkyviin, ne eivät aluksi toimineet. Toimimattomuus johtui siitä, että kuvatekstissä oli rivinvaihtoa tarkoittavia merkkejä, jotka aiheuttivat ongelmia *Javascript*-koodissa. Rivinvaihdot poistettiin ja kuvatekstit alkoivat taas toimimaan. Kuvatekstit korostettiin ja uutislaatikoihin lisättiin alareunaan häivytys, jossa uutislaatikon alareuna hävisi liukuvärin tavoin näkymättömäksi. Häivytyksellä yritettiin antaa mielikuva, että uutisen sisältö jatkuu ja että uutisesta voi lukea lisää, jos käyttäjä avaa uutisen.

Sovelluksen suorituskykyä kehitettiin ja optimoitiin. Sovelluksen osastojen välinen pyyhkäisytoiminto oli toteutettu käyttäen *Javascript*-kirjastoa. Kun pyyhkäisytoiminnon käynnisti koodissa, niin se aiheutti käyttöliittymässä hetken ajan jumittamista. Sovellusta optimoitiin niin, että pyyhkäisytoiminto alustettiin toimimaan vasta, kun sivu oli latautunut. Tällä tavoin sivu saatiin latautumaan näkyviin mahdollisimman nopeasti. Sovelluksen uutismateriaali tuotiin automaattisesti toimitus-

järjestelmästä ja siinä ilmeni joitakin ongelmia ajan myötä. Ongelmat korjattiin sitä mukaa kun niitä tuli ja sovelluksesta saatiin pikku hiljaa vakaa. Yksi ongelma oli esimerkiksi se, että uutisten kuvien joukkoon tuli pdf-tiedostoja ja joidenkin urheilutulos-uutisten tulostaulukot olivat tekstitiedostossa. Näitä ei voitu käsitellä sovelluksessa automaattisesti kuvina, vaan ne piti karsia pois sovelluksesta.

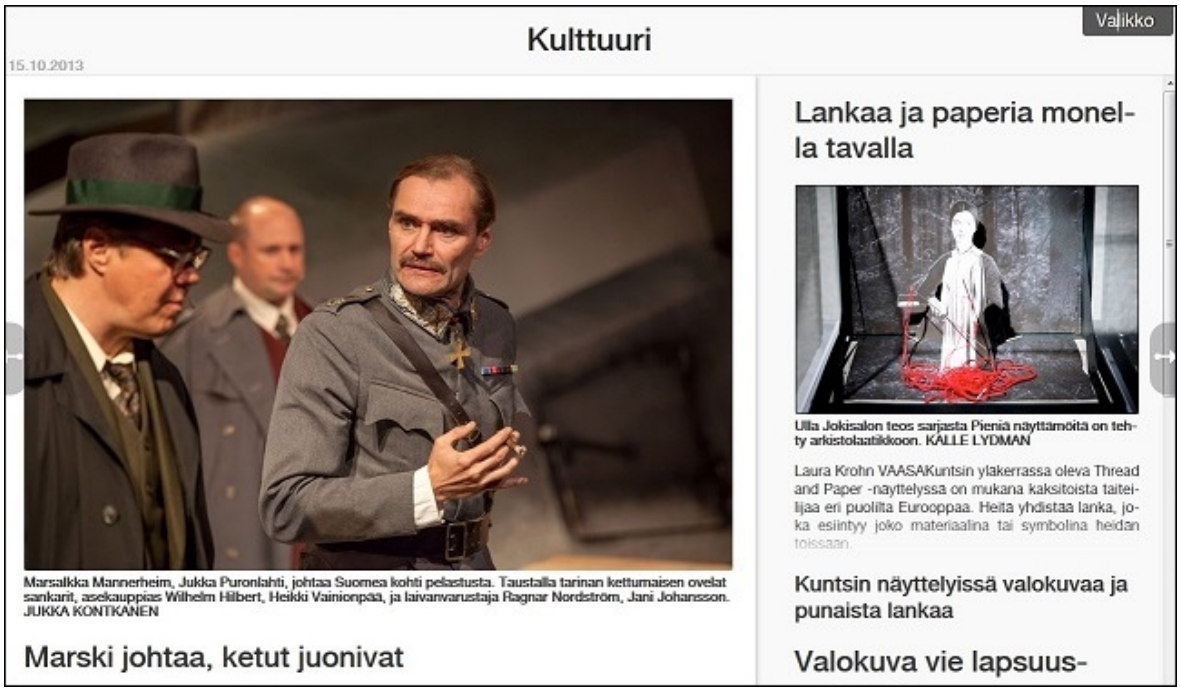
Uutissovellusta optimoitiin myös samalla, kun sitä kehitettiin. Sivun *Javascript*-suorituskyky oli vielä melko hidas *iPadissä* sekä palvelimen tietokantahaut ja osastojen kuvien haut hidastivat sivun latautumista. Tietokantakyselyitä optimoitiin ja uutissisältö sekä osastokuvat asetettiin palvelimen välimuistiin aina päivän ajaksi. Päivän ensimmäinen sovelluksen avaaja ja materiaalin lataava käyttäjä joutuu odottamaan hieman kauemmin kuin muut, mutta sen jälkeen muille data tulee suoraan välimuistista käytännössä välittömästi. Ainoastaan käyttäjän verkkoyhteyden nopeus määritteli latauksen nopeuden.

6.6.11 Julkaisu

Sovellusta oli viety lopulta niin paljon eteenpäin kuin oli järkevää. Käytetty ympäristö ja sovellus oli yritetty kehittää niin suorituskykyiseksi kuin mahdollista halutuilla ominaisuuksilla. Alkuvaiheessa valittu *jQuery Mobile*-kirjasto sekä muut sovelluskehitykseen liittyvät valinnat asettivat sovellukselle suuntalinjan. Sovellukseen latautui nyt päivittäin uusin julkaisu sanomalehdestä automaattisesti. Sovellus toimi yhteydettömässä tilassa, kunhan julkaisun oli ladannut. Sovelluksessa oli etusivu, jossa olivat kaikki uutisosastot laatikoissa.

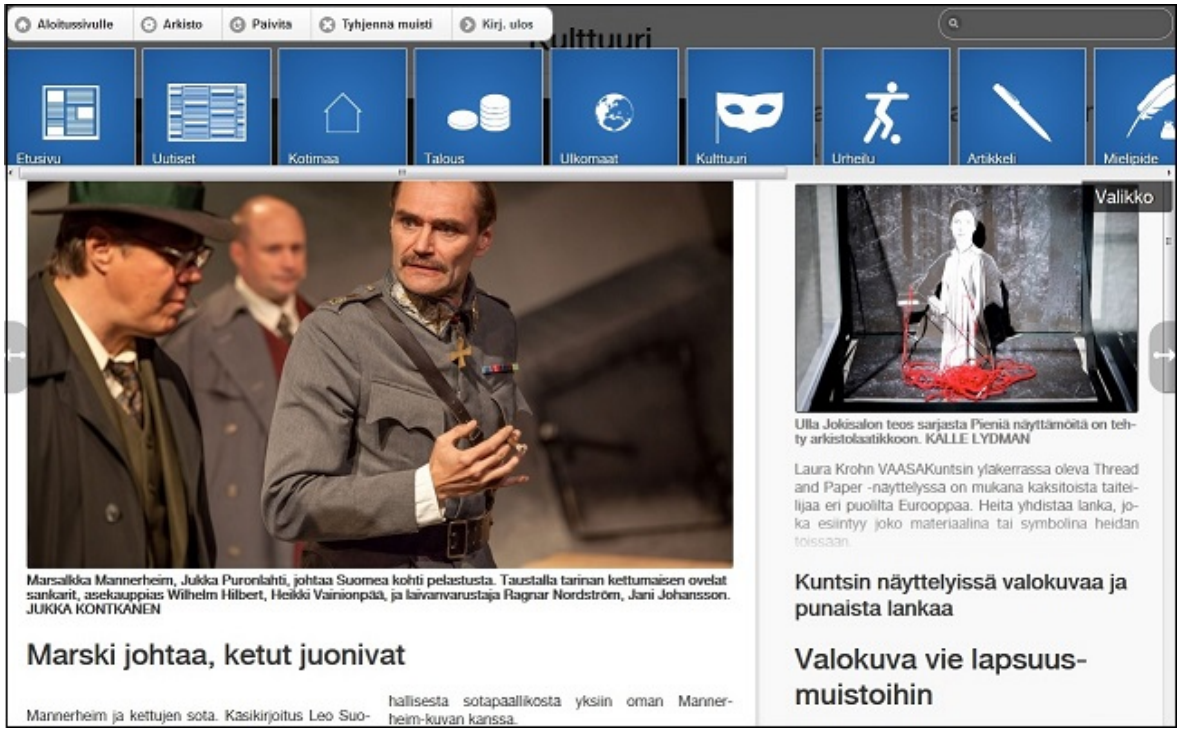
Etusivulta pääsi helposti osastoa napauttamalla navigoimaan haluamaansa osastoon. Osastosta toiseen pystyi liikkumaan pyyhkäisemällä ja näin pystyi selaamaan koko lehden läpi osasto kerrallaan. Uutiset avattiin omaan uutisikkunaan luettavaksi. Sovelluksessa oli myös koko ajan näkyvissä valikko, josta pääsi navigoimaan haluamalleen osastosivuille. Sieltä sai käytettyä myös sovellukseen toteutettuja toimintoja, kuten uloskirjautuminen ja osastovalinta. Sovellus oli toiminnassa ja sitä käytettiin päivittäin sille tarkoitetulla tavalla. Kuvassa 6.5 on sovelluksen alkunäkymä, jossa ladataan sovellusta yhteydettömään tilaan. Latauksen jälkeen ladataan uutisten kuvat ja alkunäkymän uutisosastolaatikoiden kuvat. Kuvassa 6.6 on avattuna *Kulttuuri*-uutisosasto ja kuvassa 6.7 on avattuna uutinen lukunäkymään. Kuvassa 6.8 on sovelluksen valikko, jonka saa esiin napauttamalla yläkulmassa olevaa *Valikko*-laatikkoa.

Kuva 6.5: Ensimmäisen virallisen julkaisun alkunäkymä, jossa ladataan sisältöä.



Kuva 6.6: Ensimmäisen virallisen julkaisun uutisosasto.

Kuva 6.7: Ensimmäisen virallisen julkaisun uutisen lukunäkymä.



Kuva 6.8: Ensimmäisen virallisen julkaisun valikko.

Sovellusta tehdessä opittiin paljon uusia asioita mm. yhteydettömästä tilasta ja *iPad*-web-sovelluksen tekemisestä. Sovelluskehityksen myötä opittujen asioiden jälkeen sovelluksen alkuvaiheessa olisi voitu tehdä toisenlaisia valintoja. Esimerkiksi ei olisi ollut pakko käyttää *jQuery Mobile*-kirjastoa, ja kaikkea uutissisältöä ei olisi tarvinnut työntää HTML-koodin sekaan. Sovelluksessa olisi voitu erottaa itse sovellus ja data. Olisi voitu luoda kokonaan *Javascriptilla* toimiva sovellus, niin että käyttöliittymä luodaan lennosta. Uutisartikkelit voitaisiin ladata erillisellä kyselyllä. Kun käyttöliittymä ja rakenne olisivat luotu kokonaan itse, niin niistä olisi helpompi tehdä kevyt ja yksinkertainen kokonaisuus. Nyt kun tiedettiin enemmän halutuista toiminnoista ja ominaisuuksista, voitiin kehittää parempi sovellus.

Ensimmäinen julkaisu oli käytössä ja sitä ei enää juurikaan kehitetty. Samalla tutkittiin ja kehitettiin uutta versiota samasta sovelluksesta. Tämän uuden version kehittämisessä pidettäisiin mielessä ensimmäisestä versiosta opitut asiat. Uusi sovellus pyrittäisiin pitämään mahdollisimman kevyenä ja yksinkertaisena. Pyrittäisiin välttämään kaikkia ylimääräisiä *Javascript*-kirjastoja ja tekemään itse tarpeellinen koodi sovellukselle. Sovellus yritettäisiin tehdä mahdollisimman natiivinomaiseksi ja itse sovellus erotettaisiin uutisdatasta. Käytännössä käyttäjä lataisi laitteellensa tyhjän web-sivun, jossa olisi vain sovelluksen *Javascript*-koodi. Sovellus sitten lataisi uutisdatan ja generoisi käyttöliittymän sovellukseen. Uuden version olisi tarkoitus olla kevyt, yksinkertainen ja vakaa.

6.7 Toinen virallinen julkaisu

Sovelluksen ensimmäisen version oli julkaistu käyttöön ja se oli kehityksensä päässä. Aloitettiin tutkimaan ja tekemään käyttöliittymää uudesta sovellusversiosta. Uusi versio olisi kopio vanhasta, mutta paremmin tehtynä. Sovellus toimisi kuten natiivi sovellus ja se olisi kokonaan *Javascriptilla* tehty. Itse sovellus olisi eroteltu uutisdatasta. Tämä mahdollistaisi myös muualta ladatun uutissisällön näyttämisen sovelluksessa omina uutisosastoinaan.

Toista versiota aloitettiin kehittämään ensin kehitysdemona, jossa oli kiinteä uutisdata. Data koostui muutamasta uutisosastosta ja niissä olevista uutisista. Demoa kehitettiin siihen pisteeseen, että sitä pystyi käyttämään *iPadillä* ja perustoiminnallisuudet toimivat. Kun demo oli valmis, se siirrettiin virallisesti kehityksen alle ja siihen tuotiin oikea uutisdata. Sovelluksen kehitystä jatkettiin ja selvitettiin ensimmäisessä sovellusversiossa esiin tulleita ongelmia. Uutisdata tuotiin erillisellä HTTP-

pyynnöllä sovellukseen. Sanomalehden uutisten lisäksi sovellukseen tuotiin `http://www.kp24.fi`-palvelusta etusivun uutiset RSS-syöteenä. RSS-syöte näkyi sovelluksessa omana uutisosastonaan. Ensimmäisestä versiosta poiketen sovelluksesta karsittiin kokonaan pois yhteydettömyys siinä esiintyneiden ongelmien vuoksi.

Sovellusta kehitettiin, testattiin ja käytettiin. Siitä korjattiin kaikki löytyneet bugit ja ongelmat. Lopulta saatiin aikaan uusi vakaa sovellus ja se otettiin käyttöön virallisena julkaisuna ensimmäisen sovellusversion tilalle samaan osoitteeseen `http://tablet.kp24.fi`.

6.7.1 Kehitysdemo

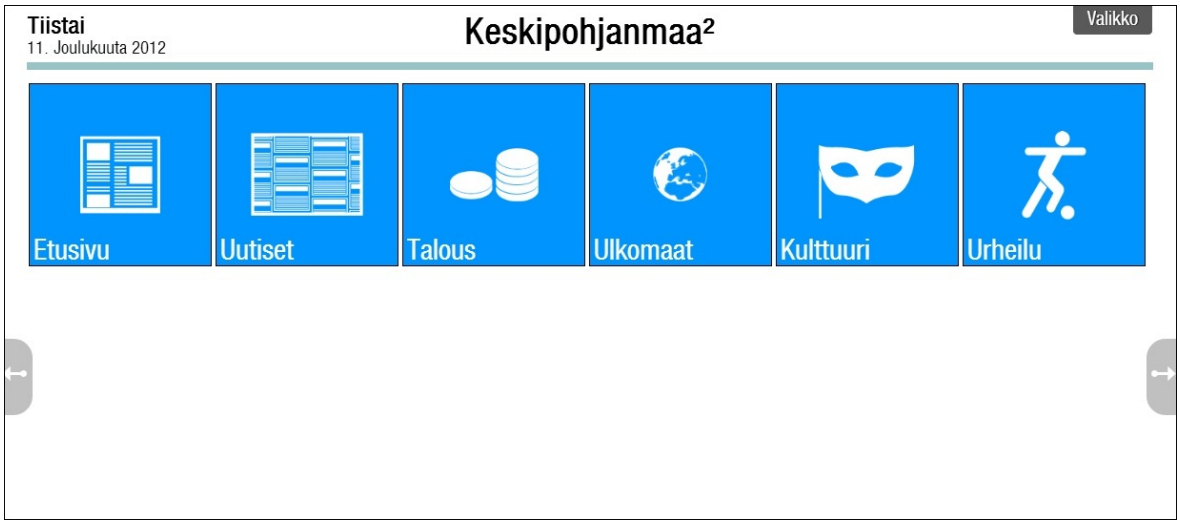
Sovelluksesta rakennettiin käyttöliittymädemo testipalvelimelle, jossa käytettiin kiinteää uutisdataa. Sovellus toteutettiin täysin *Javascript*-koodilla, niin että web-sivu ei sisältänyt juuri ollenkaan HTML-koodia, vaan kaikki luotiin dynaamisesti *Javascriptillä*. Web-sivun HTML-koodissa oli vain yksi <*div*>-elementti, jossa näytettiin sovelluksen latautuminen.

Tyhjä web-sivu, jossa oli liitteenä vain sovelluksen *Javascript*-koodit, latautui nopeasti käyttäjän selaimeen. Tällä tavoin web-sivusta saatiin enemmän natiivin sovelluksen omainen, kun sovellus käynnistyi melkein heti ja sitten vasta ladattiin sisältö. Uutisten lataus voitiin näyttää sovelluksessa edistymispalkilla.

Demossa luotiin *Javascript*-luokka uutissovellukselle. Sivun latauduttua luokasta luotiin olio ja aloitettiin luomaan käyttöliittymää. Ensin ladattiin uutissisältö ja näytettiin latauksen edistyminen. Sitten luotiin käyttöliittymä, johon kuuluivat samat osastosivut, etusivu ja valikko kuin ensimmäisessä versiossa. Ulkoasua viilattiin tyylikkäämmäksi. Sivut olivat koko ruudun kokoisia <*div*>-elementtejä, jotka olivat pystysuunnassa alaspäin rullattavia. Osastosivujen välillä pystyi navigoimaan pyyhkäisemällä sivulle. Uutiset aseteltiin dynaamisesti uutislaatikoihin uutisosastosivuille niin, että osastosivu täytettiin ylhäältä alaspäin. Uutisosasto jaettiin kolmeen sarakkeeseen, mutta dynaamisuuden ansiosta uutislaatikoilla saatiin täytettyä sivu tasaisesti. Sivuissa ei ollut yksittäisiä liian pitkiä sarakkeita, kuten ensimmäisessä virallisessa julkaisussa.

Kehitysdemon ulkoasuun otettiin mallia sanomalehdestä. Kehitysdemossa lataussivu näkyy ensimmäisenä sovellukseen tultaessa. Siinä näkyy latauspalkki uutismateriaalien lataamisesta. Kuvassa 6.9 on kehitysdemon kotinäkymä, jossa näkyvät kaikki julkaisussa olevat uutisosastot laatikoissa. Kotinäkymä näkyy lataus-

sivun jälkeen, kun uutisdata on ladattu. Kotimaa-uutisosasto on avattuna kuvassa 6.10. Kuvassa 6.11 on avattuna uutisen lukunäkymä.



Kuva 6.9: Kehitysdemon kotinäkymä.

Kehitysdemoa vietiin niin pitkälle, että sitä pystyi käyttämään demona *iPadillä*. Kun demoa oli tarpeeksi kehitetty, niin siitä tehtiin virallisesti uusi kehitettävä sovellusversio. Sovellusta alettiin jatkokehittämään ja siihen tuotiin oikea uutisdata.

6.7.2 Sovelluksen rakenne

Sovelluksen toisen version rakenne koostui tyhjästä web-sivusta, jossa oli mukana oma *Javascript*-sovellus sekä kolme *Javascript*-kirjastoa. Ensimmäistä kirjastoa käytettiin dynaamisen käyttöliittymän muodostamiseen (*Masonry*), toisella tehtiin rakenteeseen uutisosastojen välinen pyyhkäisyominaisuus (*Swipe.js*) ja kolmannella saatiin sovelluksen uutisissa tavutus toimimaan (*Hyphenator.js*). Kaikki muu tehtiin itse. Myöhemmin selvisi, että *iPadin Safari*-selain tuki CSS-tavutusta, jolla tavutuksen sai automaattisesti toimimaan. Tätä CSS-määrettä *hyphens: auto* ei vielä tuotu sovellukseen, vaan se oli vielä tuleva kehityskohde.

Sovellus latasi uutisdatan palvelimelta *JSON*-muodossa *XMLHttpRequest*-pyynnöllä, käsitteli datan ja loi sovellukseen datan määrittelemät uutisosastot ja niihin uutiset. Sovellus loi myös kaikki muut käyttöliittymäkomponentit kuten valikon, navigointipainikkeet, etusivun uutiosasto-laatikot, valikon ja sen toiminnot sekä

Kuva 6.10: Kotimaa-osasto kehitysdemossa.

Kuva 6.11: Uutisen lukunäkymä kehitysdemossa.

kaiken sovelluslogiikan.

Käytännössä käyttäjä syötti Internet-selaimen osoitekenttään url-osoitteen `http://tablet.kp24.fi`. Selain siirtyi osoitteessa olevaan *Default.aspx* -web-sivuun. Sovellus oli kirjautumisen takana, mutta jos oli jo kirjautunut, niin sovellus muisti sen ja ohjasi käyttäjän suoraan sovellukseen. Web-sivu oli melkein tyhjä HTML-sivu. Sivussa olivat vain viittaukset sovelluksessa käytettyihin *Javascript*-kooditiedostoihin. Itse sovellus oli tiedostossa *Sivut.js*. HTML-koodissa olivat vain viittaukset kirjastoihin sekä tarvittavat metatietomääritteet ja kansikuva sovellukselle.

Kansikuva oli uutisdatasta haettu *Etusivu*-osaston ensimmäinen vaakatasoinen kuva. CSS-määreellä *background-size: cover;* voitiin asettaa jokin taustakuva käyttäytymään kansikuvan tavoin web-sivulla. Kuva sopeutui erikokoisille näytöille ja taustakuvan sai näkymään keskiosassa näyttöä CSS-määreellä *background-position: center center;*.

Javascript-koodissa oli kuuntelija sivun lataukselle. Sitten kun web-sivun HTML-rakenne oli ladattu, niin käynnistettiin *Javascript*-sovellus. Sovellus loi latauslaatikon kansikuvan oikeaan alareunaan. Latauslaatikossa oli *Keskipohjanmaa*-lehden logo, julkaisun viikonpäivä ja päivämäärä sekä latauspalkki, jossa näkyi latauksen edistyminen. Sovellus latasi ensin uutisosastot taustalla *XMLHTTPRequest*-kyselynä. Kysely tehtiin *Javascript*-koodissa, jossa lähetettiin kysely palvelimella olevalle skripti-tiedostolle. Skripti palautti kyseisen päivän uutisosastot.

Sitten sovellus alkoi ladata jokaisen osaston uutiset *XMLHTTPRequest*-kyselyillä. Jokaisesta osastosta muodostettiin kysely samanaikaisesti ja palvelin palautti vastaukset satunnaisessa järjestyksessä. Edistymispalkkia liikutettiin eteenpäin sen mukaan, kuinka paljon osastojen uutisia oli ladattu. Sitten sovellus loi jokaisen osastosivun ja etusivulle osastolaatikot sekä valikkoon osastot. Samalla luotiin sovelluksen valikko, navigointiominaisuudet, siirtyminen osastojen välillä jne. Uutisten kuvat ladattiin sovellukseen vasta, kun käyttöliittymä oli luotu valmiiksi. Uutisia pääsi selaamaan jo ennen kuin kaikki kuvat oli ladattu.

Sovelluksessa pystyi avaamaan vanhemman julkaisun valitsemalla valikossa olevasta pudotusvalikosta haluamansa julkaisun päivämäärän. Julkaisuja näytettiin 2 viikkoa taaksepäin. Tässä versiossa ei siirrytty toiselle sivulle tai päivitetty koko sivua, jotta vanha julkaisu saatiin ladattua sovellukseen. Sivu toimii täysin *Javascriptin* avulla ja siinä on ajateltu, että sivu on ikään kuin sovellus, jota ei tarvitse joka ikinen kerta ladata uudestaan kuten web-sivua. Käyttäjän valitessa vanhemman julkaisun sovelluksesta poistettiin sisältö ja näytettiin vanhemman julkaisun kansikuva käyt-

täjälle. Taustalla ladattiin vanhemman julkaisun uutissisältö ja luotiin käyttöliittymä uudestaan uutissisällön pohjalta.

Tässä sovellusversiossa on eroteltuna sovellusosuus ja data. Erottelu mahdollistaa sen, että palvelimen puolelta voitaisiin tuoda muutakin uutissisältöä eri lähteistä, kun ne vain muunnetaan lehden tukemaan muotoon. Toista uutislähdettä testattiin käytännössä kun sovellukseen tuotiin *KP24.fi*-palvelusta etusivun uutiset omaksi uutisosastokseen. *KP24*-uutisosasto päivittyi aina, kun sovellukseen saapui ja siellä oli aina uusia uutisia. Sovellukselle yritettiin miettiä keinoja, millä käyttäjät saataisiin palaamaan sovellukseen uudestaan, vaikka itse lehden sisällön oli jo lukenut. Tämä uusi *KP24*-osasto oli yksi hyvä esimerkki siitä.

Etusivun uutisosastolaatikoihin tuotiin myös *KP24*-osastolaatikko, jossa oli tausta *KP24:n* teeman mukaisesti harmaa ja yläreunassa oli pienellä *KP24* logo. Laatikosta tehtiin rikkaampi siten, että sen sisällä vaihdeltiin muutamaa viimeisintä *KP24*-uutisosaston uutisotsikkoa niin, että otsikot liukuivat siinä vasemmalta oikealle vuorotellen. Otsikkoa klikkaamalla sai suoraan auki uutisen ja samalla siirryttiin *KP24*-osastosivulle sovelluksessa.

6.7.3 Perustoiminta

Käyttöliittymä oli perustoiminnoiltaan samanlainen kuin ensimmäisessä sovellusversiossa, mutta erikokoiset uutislaatikot olivat dynaamisesti lajiteltu osastosivulle. Ulkoasu matki enemmän sanomalehteä, mutta muuten sovelluksen käyttöliittymässä oli suurilta osin viilattu versio ensimmäisestä sovellusversiosta. Sovelluksen toisesta versiosta päätettiin tehdä vain Internet-yhteydellä toimiva, ensimmäisessä versiossa vastaan tulleiden yhteydettömän tilan käytettävyysongelmien takia. Sovelluksessa oli myös uusia ominaisuuksia vanhempaan versioon nähden.

Kehitettäessä web-sovellusta on hyvä muistaa, että selain asettaa kaiken sisällön välimuistiinsa tietyksi ajaksi. Seuraavalla kerralla, kun käyttäjä saapuu sovellukseen web-sivu latautuu nopeammin. Myös sovelluksen *Javascript*-koodit menivät välimuistiin. Varsinkin *iPadissä* käytetyssä koko ruudun web-sovelluksessa *Javascript* pysyi välimuistissa kohtuullisen kauan. Jos itse sovellusta päivitetään, niin pitää muistaa, että joillakin käyttäjillä sovellus ei välttämättä päivity heti. HTML-koodissa olevaan *Javascript*-viittaukseen asetettiin versionumero ja aina kun tehtiin sovelluspäivitys, niin kasvattiin numeroa yhdellä. Viittaus tehtiin näin *<script type="text/javascript"src="Sivut.js?v=1"/>*. Testatessa sovellusta viitteeseen arvottiin

satunnainen luku (*Sivut.js?v=0.1235678*), jotta sovellus aina lataisi viimeisimmän version palvelimelta.

Yksi uusi ominaisuus yritti ratkaista uutisten linkityksen. Sanomalehdessä etusivun uutiset monesti jatkuivat jossakin lehden alasivulla. Tähän ei ollut olemassa mitään linkitystä sähköisesti, joten siihen luotiin automaattisesti kohtalaisen hyvin toimiva linkitys etusivun jutuista alajuttuihin. Käytännössä, kun avasi etusivun uutisen lukunäkymään, niin uutistekstin alla oli linkkejä samankaltaisiin uutisiin. Linkkiä napauttamalla aukesi alasivun uutinen lukunäkymään. Linkitys tehtiin poimimalla valitusta uutisesta hakusanoiksi uutisen otsikon sanat, kuvatekstin sanat sekä uutistekstin tummennetut tekstit. Mikäli tarpeeksi monta vastaavuutta löytyi, niin luotiin linkki uutiseen. Parhaimmassa tapauksessa linkitys toimi täydellisesti ja huonoimmassa tapauksessa linkitystä ei saatu muodostettua ja ehdotetut samankaltaiset uutiset olivat aivan eri uutisia. Linkit toimivat kuitenkin tyydyttävästi.

Käyttöliittymä oli yritetty tehdä suorituskyvyltään mahdollisimman kevyeksi haluttujen ominaisuuksien puitteissa. Suorituskykyä parannettiin piilottamalla kokonaan käyttämättömät osastosivut ja kaikki kuvat. Kuvat, jotka eivät olleet käytössä, poistettiin kokonaan HTML-koodista. Kuvien poistaminen osastosivuilta aiheutti sen, että kun osastosivulle palasi, niin kuvat jouduttiin lataamaan uudestaan sivulle. Tässä latausta nopeutti se, että selain yleensä latasi kuvat välimuistiin ja kuvat latautuivat sieltä nopeasti esille. Joissakin tapauksissa, kun välimuisti päivittyi, niin käyttäjän selain latasi kuvat uudestaan. Sovellus toimi vain Internet-yhteydellä, joten kuvien uudelleenlatausta ei nähty tässä vaiheessa ongelmaksi.

Sovelluksessa oli kuvien uudelleenlatauksen estämiseksi ominaisuus, jossa kuvat ladattiin kerran ja tallennettiin *Javascript*-taulukkoon tekstimuodossa. Tässä käytettiin samaa *base64*-koodattua tekstimuotoa kuin ensimmäisessä sovellusversiossa. Kuvia ei tarvinnut siis ladata Internet-verkon yli joka kerta uudestaan, vaan ne ladattiin kerran ja sitten ne tuotiin käyttöliittymään *Javascript*-taulukosta. Tämä ominaisuus kuitenkin kuormitti liikaa sovellusta. Sovellus oli optimoitu tablet-laitteita varten ja kuvat poistettiin käytöstä tarkoituksella, jotta säästettiin sovelluksen kuormaa. *iPad* kaatoi koko selaimen, jos sovelluksessa oli liikaa kuormaa, jota ei tarvinnut olla erityisen paljon.

Käytettäessä sovellusta kosketusnäytöllä huomasi sen, että kun napautti jotakin painiketta tai uutislaatikkoa, siinä oli viive ennen kuin napautuksesta tapahtuva toiminto toteutui. Esimerkiksi, kun napautti valikkopainiketta, jolla sai piilotetun valikon näkyviin, niin viiveen huomasi heti. Viivettä ei lähdetty selvittämään sen

kummemmin, koska se ei häirinnyt käyttöä merkittävästi. Kehittäessämme erästä toista tablet-sovellusta, jossa oli tärkeää saada välitön palaute napautuksesta. Asiaa selvitettiin ja selvisi, että mobiililaitteilla web-sivujen *onclick* tapahtumassa on viive. Viive on noin 300 millisekuntia ennen kuin klikkaustapahtuma suoritetaan. Viive johtuu siitä, että selain odottaa toista napautusta, jolloin kyseessä on tuplanapautus [29]. Tämä ilmenee jos asetat mille tahansa HTML-elementille *Javascriptilla* kuuntelijan klikkaukselle. Sama klikkauskuuntelija *onclick* toimii myös sormen napautukselle.

Kosketukselle on olemassa monia kuuntelijoita, joita voidaan asettaa HTML-elementeille. Kuuntelijoita ovat muun muassa *touchstart,touchmove,touchend*, joista *touchstart* suoritetaan välittömästi ilman viivettä kun sormi koskettaa ruutua. Tässä pitää kuitenkin huomioida sormen liike ja kosketuksen päättyminen, ennen kuin voidaan olettaa, että käyttäjä on todella napauttanut ruutua eikä vain esimerkiksi raahannut sormea sivun yli. Kuuntelijoiden avulla voidaan rakentaa keinotekoisesti oma napautuksen kuuntelija, jolla napautus saadaan kuuntelijaan välittömästi. Tällöin viivettä ei tule käyttöliittymässä. Ryan Fioravanti esittää ongelman ja siihen ratkaisun artikkelissaan *Creating Fast Buttons for Mobile Web Applications* [29]. Viive ei ollut merkittävä ongelma uutissovelluksessa, joten sitä ei vielä lähdetty sinne korjaamaan.

Sovellukseen oli toivottu yleisesti lähennystoimintoa (zoom). Haluttiin, että sovelluksessa voisi nipistämällä lähentää ja pienentää käyttöliittymää. Sovelluksen käyttöliittymä perustui kuitenkin selaimen kokoon ja osastojen välinen sivupyyhkäisy toimi selaimen koon perusteella. Käytännössä, jos nipistystoiminto oli käytössä sovelluksessa, käyttöliittymä meni sekaisin eivätkä osastovaihtumiset toimineet enää normaaliin tapaan. Lisäksi dynaaminen asettelu aiheutti suorituskykyongelmia. Sovelluksesta päätettiin estää lähennystoiminto kokonaan. Sen sijaan kehitettiin avatulle uutiselle oma lähennystoiminto. Uutisen tekstin kokoa pystyi nipistämällä suurentamaan, jolloin teksti suureni, mutta käyttöliittymä pysyi samana. Tekstin suurennuksen lisäksi uutisen kuvan pystyi napauttamaan koko ruutuun suureksi. Kuvan vasempaan yläkulmaan lisättiin läpinäkyvä painike, jossa oli plusmerkki. Painikkeesta napauttamalla kuva aukesi koko ruutuun.

6.7.4 Suorituskyky

Myös toisessa versiossa esiintyi lopulta suorituskykyongelmia, sitten kun sinne tuotiin oikea uutismateriaali ja *KP24*-uutisosasto. Sovellusta kehitettiin edelleen pää-

osin *iPadillä* käytettäväksi. Dynaaminen uutisten asettelu aiheutti suorituskykyongelmia. Ongelmat johtuivat siitä, että käsiteltäviä uutisia oli paljon. Normaalisti web-sivussa on vain näkyvissä oleva sisältö, joka voidaan asetella dynaamisesti. Sovelluksessa jouduttiin käsittelemään koko uutissisältö, johon kuului useita osastosivuja. Sovelluksen käyttöliittymä pysähtyi kokonaan hetkeksi, kun uutiset aseteltiin sivuille. Dynaamista asettelua piti testata ja lopulta huomattiin, että sen suorituskykyä saatiin parannettua paljon muuttamalla osastosivujen rakenteita sekä *Masonry Javascript*-kirjaston käyttötapaa.

Sovelluksessa selvitettiin muitakin suorituskykyongelmia, jotka johtuivat ohjelmointitavoista ja sovelluksessa käytetyistä tekniikoista. Tällaisia olivat esimerkiksi osastosivun vaihdon hitaus ja uutisikkunan avauksessa esiintyneet ongelmat. Monet ongelmat eivät ilmenneet lainkaan perinteisellä tietokoneella käytettäessä, mutta kun sovelluksen avasi tabletilla, niin ongelma esiintyi.

6.7.5 Käyttäjien seuranta

Sovellukseen toteutettiin myös käyttäjien seuranta. Käyttäjiltä pystyttiin keräämään käyttötietoa sovelluksessa. Käyttäjätietojen keräys toteutettiin pääosin käytettävyystutkimusta varten, mutta myös mahdollista myöhemmin sovellukseen rakennettavaa mainontaa varten. Tietoa kerättiin käyttäjän saapuessa sovellukseen, lähtiessä, siirtyessä uutisosastolta toiselle, avatessa uutisen, sulkiessa uutisen ja avatessa ulkoisen linkin. Jokaisen tapahtuman mukana kerättiin aikaleima, selain, näytön resoluutio, sijainti, näytön asento ja laitteen orientaatio. Lisäksi kerättiin tietoa siitä, avattiinko uutisosasto napauttamalla vai pyyhkäisemällä.

Tietoja kerättiin vain käyttäjän sen salliessa ja sijainnin hakemista varten sovellus kysyi lupaa erikseen. Kerätyt tiedot mahdollistivat valtavan määrän erilaisia raportteja. Käyttäjän saapumisesta sovellukseen, voitiin muodostaa raportti, josta näki, milloin sovellusta käytetään päivän aikana tai luetaanko lehti aamulla vai päivällä. Uutisten avauksista saatiin kerättyä suosituimmat uutiset. Uutisen avauksen ja sulkemisen välisestä ajasta näki, kuinka kauan uutista on luettu. Tästä ajasta voitiin muodostaa raportti, josta selvisi ajallisesti luetuimmat uutiset. Samaten sovelluksen avaamisen ja lähtemisen välisestä ajasta voitiin päätellä kuinka kauan sovellusta käytettiin keskimäärin. Käyttäjän selaimesta saatiin tietoon millä selaimella ja millä laitteella sovellusta käytettiin. Näytön resoluutio paljasti minkä kokoisilta näytöiltä sovellusta käytettiin. Nämä olivat hyödyllisiä tietoja myös jatkokehitystä varten.

Sijainnin keruusta avautui uusia mahdollisuuksia. Sijainnista nähtiin mistä so-

vellusta käytettiin, mutta sen lisäksi voitaisiin tarjota käyttäjälle sijaintiin perustuvaa mainontaa. Uutissisältö voisi olla enemmän painotettu käyttäjän sijainnin mukaan. Lisäksi saatiin tieto siitä käyttivätkö käyttäjät sovellusta kotona, mökillä, työpaikalla tai matkustaessa esimerkiksi junassa. Näytön asennosta selvisi käyttivätkö käyttäjät sovellusta tabletin näytön ollessa vaakatasossa vai pystyssä. Laitteen orientaatio tarkoittaa laitteen asentoa. X, Y ja Z -koordinaatit antoivat tiedon siitä, missä asennossa käyttäjän tablet-laite oli. Tästä voitiin päätellä esimerkiksi se, että käyttikö käyttäjä sovellusta pidellen laitetta käsissä vai laitteen ollessa kiinteästi pöydällä.

Käyttäjiltä kerättyä aineistoa ei vielä tässä sovellusversiossa käytetty mitenkään hyväksi. Tiedot kerättiin ensisijaisesti käytettävyystutkimusta varten. Seuraavissa sovellusversioissa tätä aineistoa käytetään hyväksi sovelluksen parantamiseksi ja uusien ominaisuuksien tuomiseksi sovellukseen.

6.7.6 Käyttöliittymä

Toisen virallisen julkaisun käyttöliittymä oli perusrakenteeltaan samanlainen kuin ensimmäisessä versiossa lukuun ottamatta lataussivua. Avattuaan sovelluksen käyttäjä näkee ensimmäisenä lataussivun, sitten siirrytään kotinäkymään, jossa ovat uutisosastot laatikoina. Uutisosastot ovat omia sivuja. Lisäksi sovelluksessa on valikko ylhäällä. Kuvassa 6.12 on sovelluksen lataussivu, jolle sovellukseen saapuja menee ensin. Lataussivulla ladataan kaikki uutismateriaali sovellukselle ja näytetään latauksen edistyminen palkissa. Kuvassa 6.13 on sovelluksen alkunäkymä, jossa ovat kaikki julkaisussa olevat uutisosastot laatikoissa. Kuvassa 6.14 on avattuna *Etusivu*-uutisosasto. Osastosivulla ovat kaikki osastoon kuuluvat uutiset aseteltuna sivulle dynaamisesti laatikoihin. Kuvassa 6.15 on avattuna *Etusivu*-osastosta uutinen, jossa on myös linkitys pääjuttuun. Uutinen avautuu lukunäkymään käyttöliittymän päälle ikkunaan.

Lukunäkymä on sovelluksen päälle avautuva uutisikkuna, jossa uutisen voi kokonaisuudessaan lukea. Kuvassa 6.16 on sovelluksen valikko avattuna. Valikko liukuu ylhäältä piilosta alas näkyviin. Valikko sisältää kaikki uutisosastot laatikoina, sekä kaiken muun sovellukseen tehdyn toiminnallisuuden. Näitä ovat sivun päivitys, siirtyminen kotinäkymään, julkaisun avaaminen arkistosta pudotusvalikosta ja luetuimmat uutiset. Luetuimmat uutiset avautuvat omaan ikkunaan, jossa ne ovat listassa.

Etusivu-uutisosastossa avatussa uutiskkunassa oli linkitys uutisen pääjuttuun

Kuva 6.12: Toisen virallinen julkaisun lataussivu.

Kuva 6.13: Toisen virallinen julkaisun alkunäkymä.

Kuva 6.14: Toisen virallinen julkaisun uutisosasto.

Kuva 6.15: Toisen virallinen julkaisun uutisen lukunäkymä.

Kuva 6.16: Toisen virallinen julkaisun valikko.

muualla julkaisussa. Uutisikkunaan lisättiin lisäksi vielä monen kuvan käsittely. Joissakin uutisissa oli useita kuvia. Avattuun uutisikkunaan lisättiin tekstin viereen sarakkeeseen uutisessa olevat toiset kuvat, mikäli niitä oli. Pikkukuvaa pystyi napauttamaan, jolloin se näkyi uutisen pääkuvan paikalla suurena. Uutisen pääkuvan ylänurkassa oli läpinäkyvä painike, joka oli pyöreä rengas ja jonka keskellä oli plusmerkki. Painikkeesta pystyi avaamaan jutun pääkuvan koko näyttöön tarkastelua varten.

7 Yhteenveto

HTML on yksinkertainen www:ssä käytettävä merkkauskieli, jolla kuvataan HTML-dokumentin rakennetta [57]. HTML-dokumentteihin voidaan luoda linkkejä toisiin HTML-dokumentteihin. Dokumentit voivat olla fyysisesti eri puolilla maailmaa. DHTML:n myötä webin kaupallisuus lähti liikkeelle 1990-luvun loppupuolella, kun sivustoilla alettiin näyttää mainontaa. 2000-luvun alussa käyttöön otetun Ajaxin myötä oli mahdollista tehdä asynkronisia kyselyitä palvelimen ja selaimen välillä. Nykyään voidaan tehdä HTML5:tä ja CSS3-tyylejä hyödyntäviä *Javascript* web-sovelluksia, jotka häivyttävät rajaa natiivien ja web-sovellusten välillä.

Mobiililaitteelle tehty natiivi sovellus alkaa alustalle sopivasta sovelluskoodista. Koodikieliä on useita eri alustoille, sekä eri valmistajilla on omat sovelluskaupat ja -käytänteet. Web-sovellusta ei tarvitse tehdä kuin kerran ja koska kaikissa laitteissa on www-selain, niin se toimii myös kaikissa laitteissa yhdellä koodilla. Natiiveissa alustoissa olevat valmiit käyttöliittymäkomponentit ja käyttökokemukset eivät ole keskenään samanlaisia. Käyttöliittymän täysi muokattavuus on web-sovellusten yksi vahvuuksista.

Pöytätietokoneiden suosion jälkeen kannettavista tietokoneista tuli suosittuja mukana kulkevia versioita pöytätietokoneesta. Tablet-laitteet vievät kannettavuuden pidemmälle, koska tablettia voi mm. käyttää pidellen sitä käsissä, akku kestää pitkään ja sovellukset ovat välittömästi käytettävissä ilman laitteen käynnistymisen odottamista. Kosketusnäyttö on erilainen käyttöliittymä kuin perinteisen pöytätietokoneen käyttöliittymä. Tablet-laitteella kirjoja ja sanomalehtiä voidaan lukea ja käyttää sormilla. Tabletissa julkaisun sivuja voidaan vaihtaa käsin, uutista voi tarkastella katsomalla lähempää tai kääntämällä laite johonkin haluttuun asentoon.

E-kirja on painetun kirjan sähköinen versio, jota voidaan lukea sille erikseen suunnitellulla kannettavalla laitteella tai tietokoneella [54]. E-kirjalaitteita oli pelkästään e-kirjoja varten ja sitten oli laitteita, joilla pystyi tekemään muutakin. E-kirjalaitteet kehittyivät ajan myötä ja niihin tuotiin vähemmän virtaa kuluttavia ominaisuuksia kuten sähköinen muste, joka mahdollisti lukemisen päivänvalossa. E-kirjoista monipuolisempi tablet-tietokone on kompakti kannettava yleistietokone, joka on koteloitu yhteen näyttöpaneelin kanssa [68]. Vuonna 2010 julkaistiin

Applen *iPad*, jonka julkaisun jälkeen merkittävimmät laitevalmistajat ovat julkaisseet omia kilpailevia tablet-tietokoneita ja niiden suosio on kasvanut melkeinpä räjähdysmäisesti.

Vertailtaessa lukemiskokemusta tablet-tietokoneella ja kannettavalla tietokoneella, on näytön koko yksi tärkeä vertailun kohde. Kannettavassa tietokoneessa on yleensä suurempi näyttö ja siihen mahtuu enemmän tietoa yhdelle näytölle. Toisaalta tablet-tietokonetta voi lukea kuten kirjaa ja sitä on helpompi pidellä ilmassa. Suunnitellessa tablet-tietokoneelle sovellusta on huomioitava käyttöliittymässä se, että sovellusta käytetään koskettamalla. Painikkeet eivät saa olla liian pieniä ja sovellus ei saa olla liian monimutkainen. Tablet-laitteissa on huomioitavaa myös se, että niitä ei välttämättä käytä vain yksi henkilö, kuten yleensä matkapuhelimia. Tablettien käyttö jaetaan perheenjäsenten kesken.

Tablet-tietokoneiden käyttäjät lukevat uutisia useammin kuin muut ihmiset. Toisin kuin muilla ihmisillä, tablet-käyttäjien pääasiallinen uutistenlähde on Internet. Suunnitellessa sanomalehdestä tehtävää tablet-uutissovellusta pitää huomioida käyttäjien tarpeet ja lukukokemus. Useat lehdet ja muut mediat ovat tuottaneet monelle eri tablet-alustalle jonkinlaisen version julkaisustaan. Laitteiden ympärillä pyörivää keskustelua käydään enemmän julkaisijoiden kuin kuluttajien näkökulmasta. On selvitettävä, mitä kuluttaja haluaa ja mitä kuluttaja saa, jos hän vaihtaa lukualustansa sähköiseen muotoon. Julkaisijoiden ei tulisi miettiä tehdäänkö tablet-versio, vaan minkälainen siitä tulee.

Tablet-tietokoneelle suunnitellun sovelluksen käytön pitäisi olla helppoa ja yksinkertaista. Ikonien ja painikkeiden koko, elementtien pituudet, jokaisen näkymän asettelu ja muut käyttöliittymään liittyvät asiat tulisivat olla helposti käytettäviä sormilla. Tablet-laitteissa on yleensä myös käännettävä näyttö, jolloin kuvasuhde vaihtuu samalla erilaiseksi. Käyttöliittymä voidaan asetella dynaamisesti näytön koon mukaan, jolloin käyttöliittymä mukautuu erikokoiseen näyttöön.

Työn tarkoituksena oli selvittää, miten toteutetaan web-pohjainen tablet-uutissovellus. Tämän työn alkuosa koostuu kirjallisuuskatsauksesta, jossa tutkittiin webin kehitystä, HTML5-standardia ja web-teknologioita. Tutkittiin myös e-kirjalaitteita sekä tablet-laitteita ja niiden käytettävyyttä. Tutkittiin kirjallisuutta, Internet-lähteitä, artikkeleita sekä sitä, miten sanomalehdestä voidaan muodostaa tablet-tietokoneella käytettävän versio. Sovellusta tehdessä tutkittiin, mitä *Javascript*-kirjastoja voisi käyttää ja miten kannattaa edetä sovelluksen kehittämisessä.

Työn loppuosa koostuu konstruktiivisen tutkimuksen raportoinnista. Keski-Poh-

janmaan Kirjapaino Oyj:lle toteutettiin tämän työn yhteydessä Keskipohjanmaa-lehdestä digitaalinen webissä toimiva HTML5 tablet-uutissovellus, johon tuotiin sanomalehden uutisaineisto. Uutissovelluksella pystyi lukemaan päivittäisen Keskipohjanmaa-lehden uutiset sekä KP24.fi-verkkosivuilla julkaistuja uutisia. Kehitystyön aikan toteutettiin kolme prototyyppiä sekä kaksi virallista julkaistua sovellusversiota. Sovelluksen suunnittelussa oli mukana Keskipohjanmaa-lehden toimitus. Tekninen toteutus tehtiin Keskipohjanmaa-konserniin kuuluvassa it-yksikössä Kosila Digimediassa. Sovelluksen kehityksen aikana tehtiin yhteistyötä myös VTT:n tutkimusprojektin kanssa ja mukana olivat yhteyshenkilöt Aalto-yliopistosta, Metropolia Ammattikorkeakoulusta ja Arena Partners Oy:stä. Uutisdatan tuonti uutistoimituksen käyttämästä *Doris*-toimitusjärjestelmästä ja palvelimen tekninen toteutus tehtiin Kosila Digimedian kehitystiimin toimesta. Itse lukusovellus oli pääosin tämän tutkimuksen tutkijan tekemä. Ensimmäisen version koodaamiseen osallistui myös muita Kosila Digimedian tiimistä. Toisen version käyttöliittymä ja tekninen toteutus olivat pääosin kokonaan tämän tutkimuksen tukijan tekemiä.

Toteutettujen kahden eri virallisen sovellusversion rakenne oli yksinkertainen. Sovellus koostui alkunäkymästä, jossa olivat kaikki julkaisun uutisosastot laatikoissa. Jokainen osasto oli oma sivunsa sovelluksessa. Osastosivulla oli aseteltuna kaikki osaston uutiset uutislaatikoissa. Uutisen sai auki lukunäkymään napauttamalla uutislaatikkoa. Ensimmäisessä versiossa sovellusta ja dataa ei eroteltu ja sovelluksen pystyi lataamaan yhteydettömään tilaan offline-lukemista varten. Toisessa versiossa käyttöliittymän rakenne oli samankaltainen, mutta sovellus ja data eroteltiin sekä sovellus toteutettiin täysin *Javascriptilla*. Käyttöliittymä luotiin dynaamisesti datan perusteella.

Sovellusta tehdessä selvisi, että tablet-laitteiden erilaiset Internet-selaimet saattavat renderöidä web-sovelluksen HTML-sivut ja jotkin sen CSS-tyylit toisistaan poikkeavalla tavalla. Tablet-laitteissa on sen lisäksi suorituskykyongelmia ja toiminnallisuuksia, joita pitää ottaa huomioon kehitettäessä web-sovellusta. Yleisesti ajatellaan, että HTML5 web-sovellusta ei tarvitse tehdä kuin kerran ja se toimii kaikilla laitteilla. Todellisuudessa ei ole kuitenkaan ihan näin yksinkertainen vielä. Vaikka web-sovellus toimii kaikilla alustoilla, niin silti joudutaan testaamaan web-sovellusta eri laitteilla ja korjaamaan ilmenneet eroavaisuudet. Ei riitä, että testataan perinteisellä tietokoneella, koska suorituskyky ja tuetut ominaisuudet ovat erilaisia tablet-laitteilla. On kuitenkin mahdollista rakentaa kaikilla laitteilla toimiva web-sovellus. Siihen joutuu tekemään vain hiukan enemmän työtä, kuin vain yhden so-

velluksen tekemiseen menisi. Tämä työmäärä on kuitenkin murto-osa siitä työmäärästä, joka pitäisi tehdä, jos toteutettaisiin jokaiselle laitealustalle oma natiivi sovellus, julkaista se sovelluskaupassa ja ylläpitää jokaista sovellusta erikseen. HTML5 standardin valmistuttua ja kaikkien valmistajien toteutettua standardin mukaiset ominaisuudet selaimiinsa, web-sovellusten kehitys helpottuu.

Työn aikana selvisi paljon uusia asioita, ongelmia ja niiden ratkaisuita web-pohjaisen tablet-uutissovelluksen tekemisestä. Työn lopputuloksena saatiin kerättyä paljon teoriaa ja tietämystä aiheesta, huomioon otettavia asioita tablet web-sovelluksen tekemisestä ja kaksi rakenteeltaan erilaista onnistunutta ja toimivaa tablet-uutissovellusta. Sovellusta kehitetään edelleen ja odotettavissa on vielä kehittyneempiä versioita. Julkaistu käytössä oleva web-uutissovellus on osoitteessa `http://tablet.kp24.fi`.

Jatkokehityksenä voitaisiin analysoida käyttäjiltä kerätty käytettävyys-aineisto ja hyödyntää tuloksia. Teknisiä kehityskohteita olisivat suorituskyvyn tutkiminen, uudenlaiset lukutavat, dynaaminen käyttöliittymä ja yhteydettömän tilan käyttö. Uutissisältöä voitaisiin rikastaa siten, että lisättäisiin lehdessä oleviin uutisiin esimerkiksi videoita, animaatioita, käyttäjien kommentteja, uutisten arviointia, sosiaalinen media ja *aiheesta lisää* -linkkejä. Toimittajat voisivat kirjoittaa lisäjuttuja sovellukseen. Sovelluksessa ei tarvitse näyttää pelkästään uutisia vaan niiden lisäksi voitaisiin näyttää esimerkiksi paikallisia tapahtumia, mainontaa, rivi-ilmoituksia, sarjakuvia, kuvagallerioita ja videoita. Lisäksi käyttäjiä varten voitaisiin luoda ominaisuuksia, joiden takia käyttäjä palaisi sovellukseen uudestaan päivän aikana, eikä vain lukemaan aamun lehteä. Käyttäjille voitaisiin antaa mahdollisuus valita käyttöliittymässä mieleisensä tapa lukea lehteä, kuten selaamalla kaikki uutiset yksitellen tai selaamalla uutiset esimerkiksi allekkain listana. Käyttäjiä voitaisiin houkutella käyttämään sovellusta myös siten, että annettaisiin käyttäjän luoda oma uutisosasto omalla nimellään. Käyttäjä voisi kirjoittaa sinne omia uutisia tai jakaa uutisia muualta. Käyttäjä voisi tallentaa lehdestä haluamansa uutiset omaan arkistoonsa. Muut käyttäjät voisivat selata toisten käyttäjien uutisosastoja. Käyttäjien sijaintia voitaisiin hyödyntää korostamalla lähellä olevia uutisia. Mainonnassa voitaisiin tarjota paikallista mainontaa sekä käyttäjän avaamien uutisten perusteella voitaisiin tarjota kohdennettua mainontaa esimerkiksi urheilu-uutisista pitäville. Jatkokehitys ja -tutkimusmahdollisuudet ovat melkein rajattomat. Kaiken kaikkiaan web sovellusalustana on erittäin mielenkiintoinen.

Lähteet

[1] ADAMS, C., BOULTON, M., CLARKE, A., COLLISON, S., CROFT, J., FEATHERSTONE, D., LLOYD, I., MARCOTTE, E., RUBIN, D., JA WEYCHERT, R. *Web Standards Creativity: Innovations in Web Design with XHTML, CSS, and DOM Scripting*. Apress, 2007.

[2] ADOBE SYSTEMS INCORPORATED. Adobe air, 2013. URL: `http://www.adobe.com/products/air.html`, viitattu 16.4.2013.

[3] AHLROTH, J. The nine commandments for newspapers on tablet devices. *Reuters Institute Fellowship Paper* (2011).

[4] AMAZON.COM, INC. Kindle paperwhite, 2012.

[5] ANYGRAAF. Doris-toimitusjärjestelmä, 2013. URL: `http://www.anygraaf.fi/main/products_fin/doristoimitusj%E4rjestelm%E4_70.html`, viitattu 26.6.2013.

[6] APPLE INC. Apple ipad 3 retina, 2012.

[7] APPLE INC. Configuring the viewport, 2012. URL: `http://developer.apple.com/library/ios/#DOCUMENTATION/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html#//apple_ref/doc/uid/TP40006509-SW1`, viitattu 1.7.2013.

[8] APPLE INC. Configuring web applications, 2012. URL: `http://developer.apple.com/library/ios/#DOCUMENTATION/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3-SW3`, viitattu 1.7.2013.

[9] APPLE INC. Developing web content for safari, 2012. URL: `http://developer.apple.com/library/ios/#DOCUMENTATION/AppleApplications/Reference/SafariWebContent/Introduction/Introduction.html#//apple_ref/doc/uid/TP40002079-SW1`, viitattu 1.7.2013.

[10] APPLE INC. ipad, 2012. URL: http://www.apple.com/fi/ipad/, viitattu 7.11.2012.

[11] APPLE INC. Creating compatible web content, 2013. URL: http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/CreatingContentforSafarioniPhone/CreatingContentforSafarioniPhone.html, viitattu 17.7.2013.

[12] APPLE INC. Custom icon and image creation guidelines, 2013. URL: https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/IconsImages/IconsImages.html, viitattu 28.8.2013.

[13] BAKHARIA, A. *C# Fast & Easy Web Development*. Course Technology / Cengage Learning, 2002.

[14] BIRDSALL, B. Swipe, 2013. URL: https://github.com/bradbirdsall/Swipe, viitattu 28.8.2013.

[15] BOUVIER, D. Versions and standards of html. *ACM SIGAPP Applied Computing Review 3*, 2 (1995), 9–15.

[16] BOWERS, M., SYNODINOS, D., JA SUMNER, V. *Pro HTML5 and CSS3 Design Patterns*. Apress, 2011.

[17] BRANT, T. Flipboard: What is the default font used in flipboard iphone app?, 2011. URL: http://www.quora.com/Flipboard/What-is-the-default-font-used-in-Flipboard-iPhone-app, viitattu 9.7.2013.

[18] BRYSON, M. Touchswipe a jquery plugin for touch devices, 2013. URL: http://labs.skinkers.com/touchSwipe/, viitattu 23.7.2013.

[19] BUDIU, R., JA NIELSEN, J. Usability of ipad apps and websites. *Nielsen Norman Group 2* (2011).

[20] CHARLAND, A., JA LEROUX, B. Mobile application development: Web vs. native. *Queue 9*, 4 (2011), 1–8.

[21] CHISAKI, Y., IJTIHADIE, R., JA USAGAWA, T. Offline web application and quiz synchronization for e-learning activity for mobile browser. *TENCON 2010 - 2010 IEEE Region 10 Conference* (2010), 2402–2405.

[22] COLBURN, G. Rosetta tablet trends study: Q&A on key findings, 2012. URL: `http://currents.rosetta.com/index.php/2012/02/rosetta-tablet-trends-study-qa-on-key-findings/`, viitattu 16.4.2013.

[23] COUNTE, S. L. *Going Mobile : Developing Apps for Your Library Using Basic HTML Programming*. ALA Editions, 2011.

[24] DEARNLEY, J., JA MCKNIGHT, C. The revolution starts next week: the findings of two studies concerning electronic books. *Information Services and Use 21* (2001), 65–78.

[25] DESANDRO, D. Masonry cascading grid layout library, 2012. URL: `http://masonry.desandro.com/`, viitattu 15.11.2012.

[26] DESANDRO, D. Metafizzy blog, 2013. URL: `http://metafizzy.co/blog/`, viitattu 26.6.2013.

[27] DUCKETT, J. *HTML & CSS, Design and Build Websites*. Jon Wiley & Sons, Inc, 2011.

[28] FIDLER, R. 2012 rji-dpa mobile media news consumption national survey. *Reynolds Journalism Institute* (2012).

[29] FIORAVANTI, R. Creating fast buttons for mobile web applications, 2011. URL: `https://developers.google.com/mobile/articles/fast_buttons`, viitattu 27.9.2013.

[30] FIRTMAN, M. *Programming the Mobile Web*. O'Reilly Media Inc, 2010.

[31] FIRTMAN, M. Mobile html5 - compatibility on iphone, android, windows phone, blackberry, symbian and other mobile and tablet devices, 2013. URL: `http://mobilehtml5.org/`, viitattu 3.7.2013.

[32] FLANAGAN, D. *JavaScript: the definitive guide*. O'Reilly Media Inc, 2006.

[33] GIBSON, C., JA GIBB, F. An evaluation of second-generation ebook readers. *The Electronic Library 29* (2011), 303–319.

[34] GOOGLE INC. Google web toolkit, 2013. URL: http://www.gwtproject.org/, viitattu 16.4.2013.

[35] GOOGLE INC. Mobile and tablet ads, 2013. URL: https://support.google.com/adwordspolicy/answer/176117?hl=en#, viitattu 13.6.2013.

[36] GROEGER, L. Design principles for news apps & graphics, 2013. URL: http://source.mozillaopennews.org/en-US/learning/design-principles-news-apps-graphics/, viitattu 17.6.2013.

[37] HARJU, A., MÄNNISTÖ, A., JA HEINONEN, A. Debattia tableteista. *Lukulaitejournalismi -nyt -tutkimushankkeen väliraportti* (2011).

[38] HEIKKILÄ, H., HYTÖNEN, K., HELLE, M., KALLINEN, K., RAVAJA, N., JA KALLENBACH, J. ereading media use, experience & adoption. *NextMedia eReading project* (2011).

[39] HEINONEN, A. Tablettidebatissa edelleen kysymyksiä vastausten asemesta. Kirjassa *Tablettijournalismia tutkimassa*, A. Heinonen, Ed. 2012.

[40] HRUBY, P. The long strange trip of dock ellis, 2012. URL: http://sports.espn.go.com/espn/eticket/story?page=Dock-Ellis, viitattu 7.2.2013.

[41] INSTAPAPER, LLC. Instapaper, 2013. URL: http://www.instapaper.com/, viitattu 7.2.2013.

[42] INTEL CORPORATION. Intel web tablet, 2010.

[43] INTERACTIVE ADVERTISING BUREAU. Practical advice for advertising on tablets. *Tablet Buyer's Guide* (2011).

[44] JACOBSON, J., JA COMISKEY, B. Nonemissive displays and piezoelectric power supplies therefor. *Patent Cooperation Treaty (PCT)* (1998).

[45] JINSHENG, L., QING, C., JA XIJING, C. Meego* application design guidelines. *MeeGo Application Design Guidelines* (2011), 14–19.

[46] JINSHENG, L., QING, C., JA XIJING, C. App interface study on how to improve user experience. *Computer Science Education (ICCSE), 2012 7th International Conference* (2012), 726–729.

[47] JOHNSTON, C. Second chance, 2012. URL: http://www.ajr.org/Article.asp?id=5278, viitattu 26.2.2013.

[48] KARLINS, D. *Dreamweaver CS5.5 Mobile and Web Development with HTML5, CSS3, and JQuery*. Packt Publishing Ltd, 2011.

[49] KASTEN, E. Html. *Linux Journal*, 15 (1995).

[50] KELLER, M., JA NUSSBAUMER, M. Css code quality: A metric for abstractness; or why humans beat machines in css coding. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference* (2010), 116–121.

[51] KESKI-POHJANMAAN KIRJAPAINO OYJ. Kp24, 2013. URL: http://www.kp24.fi/, viitattu 26.6.2013.

[52] KORPELA, J. *HTML5 - Uudet ominaisuudet*. WSOYpro Oy, 2011.

[53] LAAKSONEN, K., JA PÄRSSINEN, K. E-kirjan hohto palaa vielä. *mB 1* (2013), 26–32.

[54] LANDONI, M. The active reading task: E-books and their readers. *Proceeding of the 2008 ACM workshop on Research advances in large digital book repositories* (2008), 33–36.

[55] LAVABLAST SOFTWARE INC. Gotcha ipad versus aspnet, 2011. URL: http://blog.lavablast.com/post/2011/05/29/Gotcha-iPad-versus-ASPNET.aspx, viitattu 18.9.2013.

[56] LAWRIE, M. Photo of the sony data discman dd8, 2011.

[57] LOWERY, J. W., JA FLETCHER, M. *HTML5 24-Hour Trainer*. Wrox, 2011.

[58] MICROSOFT CORPORATION. Microsoft silverlight, 2013. URL: http://www.microsoft.com/silverlight/, viitattu 16.4.2013.

[59] MIKKONEN, T., JA TAIVALSAARI, A. The web as an application platform: The saga continues. *Software Engineering and Advanced Applications (SEAA), 37th EUROMICRO Conference* (2011), 170–174.

[60] MITCHELL, A., CHRISTIAN, L., JA ROSENSTIEL, T. The tablet revolution and what it means for the future of news. *Pew Research Center* (2011).

[61] MÄNNISTÖ, A. Tablettijulkaisun etusivun visuaalisuus verrattuna printti- ja verkkolehteen. Kirjassa *Tablettijournalismia tutkimassa*, A. Heinonen, Ed. 2012.

[62] MOZILLA DEVELOPER NETWORK. Indexeddb, 2013. URL: https://developer.mozilla.org/en-US/docs/IndexedDB, viitattu 26.8.2013.

[63] MOZILLA DEVELOPER NETWORK. user-select, 2013. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/user-select, viitattu 17.7.2013.

[64] NARASIMHAN, P. Indexeddbshim, a polyfill for indexeddb using websql, 2013. URL: https://github.com/axemclion/IndexedDBShim, viitattu 26.8.2013.

[65] NGUYEN, K. The future of the feature: Breaking out of templates to build customized reading experiences, 2012. URL: http://www.niemanlab.org/2012/11/the-future-of-the-feature-breaking-out-of-templates-to-build-customized-reading-experiences/, viitattu 7.2.2013.

[66] ORACLE CORPORATION. Javafx, 2013. URL: http://www.oracle.com/us/technologies/java/fx/overview/index.html, viitattu 16.4.2013.

[67] OZOL, A., BENSON, D., CHAKRABORTY, J., JA NORCIO, A. A comparative study between tablet and laptop pcs: User satisfaction and preferences. *International Journal of Human-Computer Interaction 24* (2008), 329–352.

[68] PC MAGAZINE. Definition of: tablet computer, 2012. URL: http://www.pcmag.com/encyclopedia_term/0,2542,t=tablet+computer&i=52520,00.asp, viitattu 15.11.2012.

[69] PEKONEN, J.-P. Tablettidebatissa edelleen kysymyksiä vastausten asemesta. Kirjassa *Tablettijournalismia tutkimassa*, A. Heinonen, Ed. 2012.

[70] PICARD, R. Business realities of tablets and e-readers, 2011. URL: http://www.editorandpublisher.com/Features/Article/Business-Realities-of-Tablets-and-E-Readers, viitattu 6.3.2013.

[71] PILGRIM, M. The past, present & future of local storage for web applications, 2011. URL: http://simeonvisser.hubpages.com/hub/HTML5-Tutorial-How-To-Use-Canvas-toDataURL, viitattu 26.8.2013.

[72] POWELL, A. db.js, 2012. URL: http://aaronpowell.github.io/db.js/, viitattu 26.8.2013.

[73] PROFFITT, B. *Windows XP Professional Little Black Book*. Paraglyph Press, 2001.

[74] PROJECT GUTENBERG. Michael s. hart, 2011. URL: http://www.gutenberg.org/wiki/Michael_S._Hart, viitattu 17.1.2013.

[75] QUINN, S. D. Poynter 'eyetrack: Tablet' research shows horizontal swiping instinct for photo galleries, 2012. URL: http://www.poynter.org/how-tos/digital-strategies/171368/poynter-eyetrack-tablet-research-shows-horizontal-swiping-instinct-for-photo-galleries/, viitattu 27.2.2013.

[76] READ IT LATER, INC. pocket, 2013. URL: http://getpocket.com/, viitattu 7.2.2013.

[77] RICHARDSON, J., JA MAHMOOD, K. ebook readers: user satisfaction and usability issues. *Library Hi Tech 30* (2012), 170–185.

[78] RSS ADVISORY BOARD. Rss 2.0 specification, 2009. URL: http://www.rssboard.org/rss-specification, viitattu 1.7.2013.

[79] RUSNAK, P. ipad peek / iphone peek, 2013. URL: http://ipadpeek.com/, viitattu 29.8.2013.

[80] SAMSUNG ELECTRONICS CO LTD. Galaxy tab, 2012. URL: http://www.samsung.com/fi/consumer/mobile/mobilephones/galaxy-tab, viitattu 7.11.2012.

[81] SAMSUNG ELECTRONICS CO LTD. Samsung galaxy tab 2, 2012.

[82] SCHÖNHERR, M. Third generation amazon kindle, showing text from the novel moby-dick, 2010.

[83] SHELLY, C., JA YOUNG, G. Accessibility for simple to moderate-complexity dhtml web sites. *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)* (2007), 65–73.

[84] SIEGLER, M. Apple pushes past exxon to become the most valuable public company in the world, 2011. URL: http://techcrunch.com/2011/08/09/apple-exxon-valuable-company/, viitattu 15.11.2012.

[85] SPINELLI, M. You shall not flicker!, 2010. URL: http://cubiq.org/you-shall-not-flicker, viitattu 3.7.2013.

[86] STACK OVERFLOW. Safari on ipad occasionally doesnt́ recognize asp.net postback links, 2012. URL: http://stackoverflow.com/questions/12804493/safari-on-ipad-occasionally-doesnt-recognize-asp-net-postback-links, viitattu 17.7.2013.

[87] TASAVALLAN PRESIDENTIN KANSLIA. Edelliset tasavallan presidentit, 2013. URL: http://www.presidentti.fi/public/default.aspx?nodeid=44829&contentlan=1&culture=fi-FI, viitattu 17.6.2013.

[88] THE JQUERY FOUNDATION. jquery mobile: Touch-optimized web framework for smartphones & tablets, 2013. URL: http://jquerymobile.com/, viitattu 9.7.2013.

[89] THE LOWE-MARTIN GROUP. Photograph of the softbook device, 2009.

[90] TIWARI, D., JA SOLIHIN, Y. Architectural characterization and similarity analysis of sunspider and google's v8 javascript benchmarks. *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium* (2012), 221–232.

[91] VINTER, H. Mario garcia: Newspapers need to carve their niche on tablets, 2011. URL: http://www.editorsweblog.org/2011/08/16/mario-garcia-newspapers-need-to-carve-their-niche-on-tablets, viitattu 23.2.2013.

[92] VISSER, S. Html5 tutorial: How to get and set a base64 image on canvas using todataurl, 2011. URL: http://simeonvisser.hubpages.com/hub/HTML5-Tutorial-How-To-Use-Canvas-toDataURL, viitattu 23.7.2013.

[93] VIVAKI, THE POOL. Executive summary. *Tablet Lane* (2013).

[94] VOLKOV, A. ipad2 simulator css3, jquery and html5, 2013. URL: http://alexw.me/ipad2/#!safari, viitattu 29.8.2013.

[95] VOX MEDIA, INC. The verge, 2013. URL: http://www.theverge.com/, viitattu 7.2.2013.

[96] W3SCHOOLS. Css position property, 2013. URL: http://www.w3schools.com/cssref/pr_class_position.asp, viitattu 26.6.2013.

[97] W3SCHOOLS. Css3 @font-face rule, 2013. URL: http://www.w3schools.com/cssref/css3_pr_font-face_rule.asp, viitattu 9.7.2013.

[98] W3SCHOOLS. Css3 transitions, 2013. URL: http://www.w3schools.com/css3/css3_transitions.asp, viitattu 26.6.2013.

[99] WAGNER, R. *Beginning iOS Application Development with HTML and JavaScript*. Wiley, 2012.

[100] WARKENTIN, M., BEKKERING, E., SCHMIDT, M., JA JOHNSTON, A. Proposed study of end-user perceptions regarding tablet pcs. *Proceedings of the Information Resources Management Conference, Idea Group Publishing* (2004), 430–431.

[101] WHITE, G. ios 5 native scrolling-grins & gotchas, 2012. URL: http://cantina.co/2012/03/06/ios-5-native-scrolling-grins-and-gothcas/, viitattu 17.7.2013.

[102] WIKIMEDIA FOUNDATION. Tablet-pc hp tc-1100, 2005.

[103] WILSON, R. Ebook readers in higher education. *Educational Technology and Society 6* (2003), 8–17.

[104] WOLFE, A. Putting pen to screen on tablet pcs. *Spectrum 39* (2002), 16–18.

[105] ZICKUHR, K. Tablet ownership 2013. *Pew Research Center's Internet & American Life Project* (2013).