Joni Eronen

HTML5 mobiilisovellusten kehityksessä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Ohjelmistotekniikka
Insinöörityö
7.4.2013

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Joni Eronen HTML5 mobiilisovellusten kehityksessä 37 sivua 7.4.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Juha Kämäri Liiketoimintayksikön johtaja Timo Laaksonen

Työn aiheena on tarkastella HTML5:n tarjoamia uusia teknologioita mobiilikehitykseen. Työn alussa käsitellään yleisesti, mitä HTML5 on ja minkälaisia uusia ominaisuuksia se tarjoaa.

Työn keskiosassa vertaillaan verkkosovellusten tehokkuutta verrattuna laitteiden natiivisovelluksiin ja perehdytään tarkemmin, miten HTML5:n avulla voidaan toteuttaa natiivisovellukselle ominaisia toimintoja verkkosovelluksiin. Käsiteltäviä aihepiirejä ovat offline-käytettävyys, verkkosivun skaalautuminen pienille näytöille ja sivusiirtymien luominen.

Työn lopussa esitellään demo, jossa käytetään teoriaosuudessa esiteltyjä HTML5-ominaisuuksia. Käydään läpi demototeutus, jossa on tehty offline-tilassa käytettävä uutistenlukija. Demossa hyödynnetään responsiivistä suunnittelua ja toteutetaan sivusiitymä CSS3-animaatioiden avulla.

Lopputulosten perusteella voi päätellä, että HTML5 on käyttökelpoinen joskin osittain vielä keskeneräinen. Useita uusia ominaisuuksia on turvallista ottaa käyttöön selaimille, jotka niitä tukevat, rikkomatta vanhojen selaimia. Etenkin kevyemmät verkkosovellukset, jotka eivät vaadi koko mobiililaitteen suorituskykyä, toimivat erittäin hyvin.

Avainsanat	HTML5, CSS3, Mobiilisovellus, Verkkosovellus

Metropolia

Abstract

Author(s) Title Number of Pages Date	Joni Eronen Creating mobile applications using HTML5 37 pages 7 April 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Juha Kämäri, Senior Lecturer Timo Laaksonen, Director

The purpose of the thesis is to find out what kind of new features HTML5 offers for mobile development. First it was found out what exactly HTML5 is and what kind of features it offers.

The middle part of the thesis focuses on the performance of HTML5 mobile applications in comparison to their native mobile application counterparts. In addition, an introduction to how to create web applications using similar functionality as native applications is provided. Topics discussed are offline usability, scalable websites and page transitions.

In the end a working demo using the HTML5 functionality is presented. The demo is an offline usable news reader application featuring the responsive web design and page transitions.

Based on the results it can be concluded that HTML5 is usable but still slightly unfinished. It is safe to start using several of the HTML5 features for the browsers that support them without breaking the functionality of the older browsers. Especially lightweight web applications that do not require full processing power of the mobile device work very well.

Keywords	HTML5, CSS3, Mobile application, Web application

Sisällys

Lyhenteet

Ajax	Asynchronous JavaScript and XML. Teknologia joka mahdollistaa tiedon lähetyksen ja vastaanoton ilman selainsivun päivitystä.
CodeIgniter	Avoimen lähdekoodin php-sovelluskehys.
CSS	Cascading Style Sheet. Kieli, joka kuvaa verkkosivun ulkoasua.
DOM	Document Object Model on JavaScriptin rajapinta, jota käytetään verkkosivun rakenteen muokkaamiseen.
HTML	HyperText Markup Language on verkkosivujen rakennetta kuvaava merkkauskieli.
HTML5	HTML-spesifikaation viides versio.
IE	Internet Explorer. Microsoftin kehittämä verkkoselain.
JavaScript	Selaimessa toimiva skriptikieli, joka mahdollistaa selaimen dynaamisen toiminnan.
JSON	JavaScript Object Notation. Tiedonkuljetukseen käytettävä datarakenne.
jQuery	JavaScriptin ympärille rakennettu kirjasto, joka lyhentää, helpottaa ja täydentää useiden JavaScriptin ominaisuuksien käyttöä.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri joka pyrkii erottamaan käyttöliittymän ja tietosisällön toisistaan.
Natiivisovellus	Laitteen omilla rajapinnoilla ohjelmoitu ohjelma.
PHP	Hypertext Preprocessor. Palvelinpään skriptikieli, jolla tuotetaan dynaamisia verkkosivuja.

RSS	Really Simple Syndication eli uutissyöte. XML-pohjainen tiedon kuljetus tyyppi.
SVG	Scalable Vector Graphics. XML-pohjainen vektorikuva.
SQL	Structured Query Language. Tietokannan kyselykieli.
userAgent	Selaimen merkkijono, joka sisältää tietoja kuten selaimen nimen ja version.
XHTML	Extensible HyperText Markup Language. XML-rakennetta noudattava HTML-sivu.
XML	Extensible Markup Language. Ihmis- ja koneluettava datarakenne.

1 Johdanto

Opinnäytetyöni tavoitteena on tutustua HTML5-teknologian tarjoamiin ominaisuuksiin ja selvittää, miten niitä voi hyödyntää mobiilisovellusten kehittämisessä. Selvitystyön tilanneen Cybercom Finland Oy:n tavoitteena on kouluttaa lisää osaajia käyttämään HTML5:n tarjoamia ominaisuuksia ja pysyä kilpailukykyisenä jatkuvasti mobilisoituvassa yhteiskunnassa.

Työn alussa käsittelen yleisesti, mitä HTML5 on ja minkälaisia uusia ominaisuuksia se tarjoaa, miten ne saa käyttöön ja mitkä laitteet sekä selaimet tukevat kyseisiä ominaisuuksia. Selvitetään, mitä ongelmia uudesta teknologiasta on ja miten sen saa toimimaan myös vanhemmissa selaimissa.

Opinnäytetyöni ei pyri selvittämään kaikkia HTML5:n ominaisuuksia vaan keskittyy tiukasti mobiilikehityksen kannalta oleellisiin asioihin. Selvitetään, miten HTML5 tuo verkkosivujen toiminnallisuuden lähemmäksi mobiililaitteden natiiveja ominaisuuksia. Tutkitaan, mihin HTML5:sta kannattaa käyttää ja mitkä ovat sen edut ja haitat verrattuna mobiililaitteiden natiivisovelluksiin.

Käsiteltäviä ominaisuuksia ovat offline-käytettävyys, Ajax-pohjaiset verkkosivut, kosketuseleet ja laitteen koosta riippumattomat sivurakenteet sekä animaatiot. Työn alkuosassa on pääasiassa teoriaa, miten eri ominaisuuksia käytetään ja lopuksi esitellään toimiva demo, jossa käytetään käytännössä työn alkuosassa esiteltyjä teknologioita.

2 HTML5

2.1 HTML5:n lyhyt historia

HTML (HyperText Markup Language) on verkkosivujen kuvauskieli, joka otettiin ensimmäisen kertaa käyttöön vuonna 1990. Muutaman ensimmäisen vuoden aikana HTML-standardia muutettiin useaan kertaan, kunnes vuonna 1997 julkaistiin HTML4-standardi. Vuoden 1997 jälkeen HTML-standardia ei ole kehitetty eteenpäin. [1.] Ajan kuluessa laitteet ja ohjelmistot kuitenkin kehittyivät eikä HTML enää täyttänyt uusien verkkosivujen tarpeita.

Vuonna 2004 Mozilla ja Opera ehdottivat HTML-standardin kehittämistä W3C:n (World Wide Web Consortium) työpajassa. Ehdotus kuitenkin hylättiin, sillä se oli ristiriidassa aiemmin valitun XHTML-pohjaisen kehityssuunnitelman kanssa. [1.]

Myöhemmin samana vuonna Mozilla, Apple ja Opera esittivät huolensa W3C:n kiinnostumattomuudesta HMTL-kieltä kohtaan ja loivat WHATWG-ryhmän (Web Hypertext Application Technology Working Group), jonka tarkoitus on kehittää HTML:ää eteenpäin. [2.] Vuonna 2006 W3C muutti mielensä ja ilmoitti kiinnostuksensa HTML5:n kehitystä kohtaan. Vuodesta 2007 lähtien ryhmät ovat työskennelleet yhdessä yleisen standardin kehityksessä. [1.]

2.2 HTML5 yleisesti

HTML5 ei ole pelkästään viimeisin versio HTML-kuvauskielestä, kuten nimestä voisi päätellä. HTML5 on yleinen termi, jolla kuvataan nykyaikaisten verkkosivujen käyttämiä teknologioita. HTML5:n kolme suurinta teknologiaa ovat HTML, CSS (Cascading Style Sheet) ja JavaScript. [3.]

HTML kuvaa käytössä olevat elementit ja sivun rakenteen. CSS kertoo verkkoselaimelle, miltä elementtien tulisi näyttää. JavaScript mahdollistaa HTML-rakenteen muokkaamisen ja välittömän palautteen käyttäjälle. Jotkut näkevät HTML5:n vain uusina elementteinä, mutta nykypäiväisten verkkosivujen luonne on muuttunut niin paljon, että kaikkien kolmen teknologian ymmärtäminen on välttämätöntä. [3.]

WHATWG suunnitteli HTML5:sta neljän periaatteen varaan. Nämä periaatteet ovat Compatibility (yhteensopivuus), Utility (hyödyllisyys), Interoperability (yhteentoimivuus) ja Universal access (yleismaailmallinen käytettävyys). [4.]

Yhteensopivuudella tarkoitetaan, että HTML5:n tulee toimia kaikkialla. Uudet toiminnallisuudet eivät saa rikkoa vanhoja toimintoja. Mikäli uutta toimintoa ei tueta, sivun pitäisi silti olla käyttökelpoinen. On myös tärkeää, että kaikki vanha HTML-koodi on edelleen käyttökelpoista. [4.]

Hyödyllisyydellä tarkoitetaan, että käyttäjän kokemus on tärkeämpi kuin se, että HTML-koodi täysin puhtaiden standardien mukainen. Tämä tarkoittaa esimerkiksi sitä, että

isoilla ja pienillä kirjoitetut tagit ja attribuuttien nimet ovat samanarvoisia. On tärkeämpää, että loppukäyttäjän ei tarvitse kärsiä pienestä syntaksivirheestä kuin että sivu on täysin standardien mukainen. [4.]

Hyödyllisyydellä kuvataan myös sitä, että kaikki toiminnallisuudet suunnitellaan alusta pitäen turvallisiksi. Uusien turvallisuuskäytäntöjen ansiosta ennen mahdottomia toimintoja voidaan toteuttaa ilman turvallisuusriskejä. [4.]

Yhteentoimivuudella tarkoitetaan, että yksinkertaisuus on parempi. Sen sijaan, että jokaiselle sivulle rakennettaan pitkiä ja vaikeita JavaScript-toteutuksia, tehdään yleisesti käytetyistä ominaisuuksista osa selaimen omia ominaisuuksia. Kaikki osat myös määritellään ja dokumentoidaan huomattavasti tarkemmin, jotta toiminnallisuudet voisivat olla mahdollisimman lähellä toisiaan. [4.]

Yleismaailmallisella käytettävyydellä tarkoitetaan, että HTML5-toiminnallisuudet tulisi toimia mahdollisimman hyvin kaikilla laitteilla ja alustoilla. HTML5 tukee kaikkia kieliä kuten aasialaisia merkkejä uudella ruby-elementeillä. Käytettävyys mahdollistaa myös sen, että näkörajoitteinen henkilö voi helpommin käyttää sivustoa esimerkiksi ruudunlukuohjelman avulla. [4.]

2.3 Multimediaa ilman lisäosia

Verkkosivuilta voi yleisesti löytää tiettyjä toistuvia teemoja. Useimmat sivut sisältävät otsikon, navigaatiopalkin ja alatunnisteen. Nämä on perinteisesti toteutettu div-elementeillä, joille on annettu vastaava id-määre esimerkiksi .

HTML5 tarjoaa uusia semanttisia elementtejä, joita ovat header, nav, section ja footer. Näiden tarkoitus selkeyttää HTML:n rakennetta ja kuvata sivuilla yleisesti käytettäviä osia. Header on yleisesti sivun alussa oleva kenttä, johon kuuluvat otsikko, logot ja hakukentät. Nav sisältää listan linkkejä, joilla sivulla navigoidaan. Section erottelee sivun eri osat toisistaan, esimerkiksi sivulla voisi olla uutis- ja yhteystieto-osiot. Footerin sisälle laitetaan usein tietoa artikkelin kirjoittajasta, tekijänoikeustietoja ja linkkejä. [5.]

On hyvä huomata, että semanttisia elementtejä ei ole luotu korvaamaan div-elementtejä vaan tuomaan dokumentille selkeämmän rakenteen, jotta se olisi parem-

min luettavissa esimerkiksi ruudunlukijoille ja hakuroboteille. Toisin kuin semanttiset elementit, div-elementti ei kerro mitään sen sisällöstä ja sitä tulisi edelleen käyttää apuvälineenä sisällön tyylien kanssa.

Verkkosivujen videot ja äänet on tavallisesti otettu käyttöön erilaisten lisäosien avulla. Lisäosien käyttö aiheuttaa erilaisia ongelmia kuten tietoturvariskejä, jotka eivät ole kehittäjien hallinnassa. Osa käyttäjistä poistaa lisäosat käytöstä, sillä niitä käytetään yleisesti mainostamistarkoituksiin. Tällöin myöskään video- ja äänisisältö ei ole käytettävissä. Lisäosat eivät myöskään pysty kommunikoimaan hyvin muiden sivun elementtien kanssa. [4.]

HTML5 tarjoaa uusia mediaelementtejä, joiden sisältöä ja asetuksia voidaan helposti säätää yhtenäisellä JavaScript-rajapinnalla. Video- ja audio-elementeille voidaan yksinkertaisimmillaan antaa vain src-ominaisuus, joka kertoo, missä toistettava tiedosto on. Media-elementtien alle voi kuitenkin laittaa useampia source-elementtejä, joka mahdollistaa videoiden toistamisen useammissa selaimissa. [4.]

```
<video id="videoPlayer" width="320" controls>
  <source src="video.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src=" video.mp4" type="video/mp4" />
  <source src=" video.ogv" type='video/ogg; codecs="theora, vorbis"' />
  Selaimesi ei tue video tagia.
</video>
<input type="button" id="videoBtn" value="Play/Pause"/>
```

Kuvio 1. Videoelementti

Kuviossa 1 on yksinkertainen videoelementti ja kolme source-elementtiä. Kuvio 2 on esimerkki, miltä kuvion 1 luoma videoelementti näyttää selaimessa.

Kuvio 2. Video-elementti selaimessa

Videon alapuolella on nappula, johon toteutetaan toisto- ja pysäytys-ominaisuudet käyttäen video-rajapintaa.

```
var videoPlay = false;
var video = document.getElementById('videoPlayer');
$("#videoBtn").click(function() {
  videoPlay ? video.pause() : video.play();
  videoPlay = !videoPlay;
});
```

Kuvio 3. Play/Pause-nappula

Kuvio 3:n koodiesimerkissä haetaan sivulta halutut video- ja input-elementit. Kun nappulaa painetaan, jQueryn tapahtumakuuntelijassa joko käynnistetään tai pysäytetään videon toisto video-rajapinnan pause- ja play-metodeilla.

Yksinkertaisen viivan piirtäminen selaimen ruudulle kuulostaa yksinkertaiselta, mutta on itse asiassa melko monimutkaista, sillä HTML4-spesifikaatiossa ei ole yksinkertaista rajapintaa grafiikan piirtämiselle. Käytännössä grafiikkaa vaativat sovellukset on toteutettu käyttäen erilaisia lisäosia kuten Adoben Flashia. [4.] HTML5 tuo mukanaan canvas- ja svg-tagit, jotka mahdollistavat vektorigrafiikan käytön ilman lisäosia ja niiden muokkaamisen suoraan JavaScriptillä.

Canvas-elementti luo sivulle neliönmuotoisen alueen, johon voidaan piirtää viivoja, tekstiä ja kuvia JavaScriptillä samankaltaisella tavalla kuin monen muun kielen grafiikkakirjastoissa. Grafiikoille annetaan alku- ja loppukoordinaatit sekä värit.

Svg-grafiikat on tavallisesti liitetty sivulle img-tagin avulla kuten kaikki muutkin kuvat. Svg-elementti mahdollistaa vektorigrafiikan upotuksen suoraan HTML:n sisään. Tämän ansiosta svg-elementti voidaan valita JavaScriptillä ja sitä voidaan muokata reaaliaikaisesti.

En mene tämän työn puitteissa tarkemmin siihen, miten canvas ja svg-grafiikoita käytetään, vaan keskityn oman demon osalta oleellisimpiin ominaisuuksiin.

2.4 HTML5:n käyttöönotto

Nykyinen HTML5-spesifikaatio on working draft (toimiva luonnos) tilassa. Tämä tarkoittaa, että se on vielä keskeneräinen, eikä kaikki toiminnallisuus ole vielä käytössä tai lopullista. Spesifikaation valmistumisvuodeksi on määritelty 2022. Tämä ei kuitenkaan tarkoita, että HTML5:sta ei tulisi käyttää ennen kyseistä päivämäärää. Selainvalmistajat ovat jo ottaneet käyttöön monia ominaisuuksia ja lisäävät uusia jatkuvasti. [4.]

Tässä luvussa käsittelen, miten HTML5-ominaisuuksia voi ottaa käyttöön siten, että sivusto on edelleen käytössä myös vanhemmissa selaimissa. Ratkaisuina toimivat erilaiset apukirjastot, jotka joko kertovat kehittäjälle kykeneekö selain käyttämään jotain ominaisuutta tai yrittävät tuottaa mahdollisimman samankaltaisen tuloksen myös vanhoille selaimille. Käsitteltäviä apukirjastoja ovat HTML5Shiv, Modernizr ja Selectivizr.

Jotta voisimme ymmärtää, miten HTML5Shiv mahdollistaa uusien elementtien käytön myös vanhoissa selaimissa on ensin ymmärrettävä, miten selain käsittelee tuntemattomia elementtejä. Jokaisella selaimella on sisäänrakennettu lista elementeistä, joita se ymmärtää. Oletuksena kaikki tuntemattomat elementit käyttävät tiettyä tyyliä. [5.]

Vanhemmat selaimet eivät tunnista esimerkiksi HTML5:n article-tagia. Tämä tarkoittaa, että kaikki CSS-määreet, jotka on asetettu HTML5-elementeille, jätetään huomioimatta. Tämän voi kiertää yksinkertaisella JavaScriptillä, joka luo sivun alussa tyhjän HTML-elementin kuten kuviossa 4. [5.]

```
<script>document.createElement("article");</script>
```

Kuvio 4. Tyhjän article-elementin luominen

HTML5Shiv on skripti, joka tekee vastaavan tempun kaikille uusille HTML5-elementeille. Tämän jälkeen sivulla voidaan vapaasti käyttää kaikkia uusia semanttisia elementtejä ilman, että pitää huolehtia vanhemmista selaimista. HTML5Shiv ympäröidään ehdollisilla kommenteilla, joilla tarkistetaan onko käytössä vanha Internet Explorer-selain kuten kuviossa 5. [5.]

```
<!--[if lt IE 9]>
  <script src="html5shiv.js"></script>
<![endif]-->
```

Kuvio 5. HTML5Shivin käyttöönotto

Modernizr on avoimen lähdekoodin JavaScript-kirjasto, joka pystyy tunnistamaan selaimen HTML5- ja CSS3-tuen. Modernizr pyrkii tunnistamaan laitteen todelliset ominaisuudet, eikä luota pelkästään selaimen userAgent-määreeseen. UserAgent-määre on selaimen sisältämä merkkijono, joka kertoo, mikä selain on kyseessä. Käyttäjät voivat kuitenkin halutessaan vaihtaa userAgent-määrettä, joten se ei ole erityisen luotettava tapa tutkia selainominaisuuksia. [6.]

```
Modernizr.load({
 test: Modernizr. localstorage,
 yep : 'localstorage-implementation.js',
 nope : 'localstorage-polyfill.js'
});
```

Kuvio 6. Modernizerin Yepnope.js

Kuviossa 6 on yksinkertainen tapa tarkastella selaimen tukemia ominaisuuksia. Riippuen tukeeko selain haluttua ominaisuutta esimerkkikoodissa localStorage, haetaan joko yep- tai nope-haaran JavaScript toteutus. [6.]

Selaimen tukemia ominaisuuksia voi testata myös perinteisillä JavaScriptin if-lauseilla kuten kuviossa 7.

```
if (Modernizr.touch){ // Mobiililaite
  document.addEventListener('touchstart', function(e){
  ...
} else { // Tietokone
  document.addEventListener('mousedown', function(e){
  ...
}
```

Kuvio 7. Yksinkertainen ominaisuuden tunnistus

Kuviossa 7 tarkastellaan, onko kyseessä kosketuspohjainen laite kuten älypuhelin tai tabletti. Riippuen käytettävästä laitteesta koodi asetetaan tunnistamaan joko kosketuseleitä tai hiirenpainalluksia.

Tunnistustapojen suurin ero on, että Yepnope.js lataa vain toisen skripteistä. [6.] Tämä mahdollistaa hieman nopeamman sivun lataamisen, joka voi olla erittäin hyödyllistä esimerkiksi hitailla mobiiliyhteyksillä.

```
.glowy { /* Kummitusmaiset kirjaimet */
  color: transparent;
  text-shadow: 0 0 10px black;
}
.no-textshadow .glowy { /* Selaimille jotka eivät tunnista text-shadow-määrettä */
  color: black;
}
```

Kuvio 8. CSS-ominaisuuksien tunnistus

Kuviossa 8 on CSS:n vastaava ominaisuuden tunnistus. Selain, joka osaa käyttää teksteille asetettuja varjoja, näyttää läpinäkyvät kirjaimet ja kummitusmaisen varjon, kun taas vanhemmat selaimet näyttävät vain tavallisen mustan tekstin.

Selectivizr on toinen yleisesti käytetty apukirjasto. Se toimii samalla ideologialla kuin HTML5Shiv, sillä sen tarkoitus on saada käyttöön ominaisuuksia, jotka eivät ole tavallisesti käytettävissä IE6-IE8-selaimissa.

Selectivizr emuloi CSS3:n tarjoamia pseudo-luokkia ja attribuutti-valitsimia. Selectivizrin voi ottaa käyttöön yksinkertaisesti kuten kuviossa 9. Käyttöönoton jälkeen monet CSS3-valitsimet toimivat myös vanhemmilla IE-selaimilla. Selectivizr on yhteensopiva muiden yleisesti käytettyjen apukirjastojen kuten jQueryn kanssa. [7.]

```
<!--[if (gte IE 6)&(lte IE 8)]>
  <script type="text/javascript" src="selectivizr.js"></script>
<![endif]-->
```

Kuvio 9. Selectivizrin käyttöönotto

3 Natiivisovellus vs. HTML5-websovellus

HTML5:n ja natiivisovelluksien välillä on jo pitkään käyty väittelyä. Kumpi on parempi, tehokkaampi ja käytettävämpi? Kummalla tavalla mobiilisovellus kannattaa kehittää? Näihin kysymyksiin ei ole yhtä oikeaa vastausta tai ratkaisua, mutta pyrin tässä luvussa käsittelemään molempien tapojen hyviä ja huonoja puolia. Väittely jakautuu kuuteen pääkohtaan: toiminnallisuuksien määrään, tehokkuuteen, kehittäjien kokemukseen, käyttökokemukseen, löydettävyyteen ja rahoittamiseen.

Natiivisovellukset käyttävät suoraan laitteen omia rajapintoja ja pystyvät täten käyttämään kaikkia laitteen toiminnallisuuksia kuten kosketuseleitä, GPS:ää ja kameraa. Verkkosovellus on aina rajattu eikä voi kehittäjän kyvykkyydestä huolimatta ottaa kuvia käyttäen laitteen sisäänrakennettua kameraa. Jos verkkosovelluksen on kuitenkin pakko kyetä ottamaan valokuvia, on mahdollista tehdä hybridisovellus, jolloin tietyt ominaisuudet voidaan toteuttaa natiivisti. Hybridisovelluksen tekeminen ei kuitenkaan ole erityisen ideaalista ja tekee sovelluksesta monimutkaisen. Natiivisovellukset pystyvät kieltämättä toteuttamaan enemmän toimintoja, mutta verkkosovelluksien toiminnot kasvavat hurjaa vauhtia ja monet teknologiat kuten offlinekäytettävyys toimivat jo suurimmassa osassa nykyaikaisista mobiililaitteista. [8.]

Natiivisovellukset saavat täydet edut laitteiden suorituskyvystä. Ne kykenevät käyttämään näytönohjaimen täyttä tehoa ja ajamaan useampia säikeitä samaan aikaan. Verkkosovellukset käyttävät JavaScriptiä, joka ei voi suoraan hyödyntää laitteiden kaikkia tehoja. Selaimet ja JavaScriptin suoritusnopeudet ovat kuitenkin kehittyneet paljon viimeisten vuosien aikana, ja HTML5 tuo mukanaan ominaisuuksia, jotka tulevat korjaamaan nykyisiä puutteita. Ennen kaikkea on hyvä muistaa, että suurin osa mobiilisovelluksista ei ole upouusia 3D-pelejä tai grafiikkasovelluksia vaan esimerkiksi uutislukijoita ja sähköpostiohjelmia, jotka eivät edes tarvitse kaikkea laitteen tehoa toimiakseen. [8.]

Natiivisovellukset ohjelmoidaan käyttäen vankkoja ohjelmointikieliä kuten Javaa, Objective C:tä ja C++:aa, jotka on suunniteltu monimutkaisten sovellusten kehittämiseen. Käytössä olevat ohjelmointirajapinnat on usein suunniteltu alusta pitäen tukemaan tiettyä laitetyyppiä. Täten niitä on helppo testata ja löytää virheet. Laitteiden ja niiden käyttöjärjestelmien kirjo on kuitenkin hyvin suuri, ja voi olla vaikea sanoa, onko joku toiminto käytössä juuri tietyssä laitteessa. [8.]

Verkkosovellusten hyödyt tulevat esille erityisesti, kun pyritään tekemään sovellus, joka toimii useammalla laitetyypillä samaan aikaan. Jos halutaan tehdä natiivisovellukset, jotka toimivat iOS- ja Android-laitteilla, on ensin kirjoitettava iOS-sovellus Objective C:llä ja sitten koko sovellus uudelleen Javalla. Tästä aiheutuu lisätyötä, ja koodin ylläpito vaikeutuu. Yksi verkkosovellus ei toimi pelkästään sekä Android- että iOS-laittella vaan myös kaikilla muilla laitteilla kuten Windows- ja BlackBerry-puhelimilla. Tulevaisuudessa yhä useampi laite kykenee käyttämään verkkosovelluksia kuten älytelevisiot. [8.]

Jokaisella laitteella on oma tuntuma, miten käyttäjät olettavat sen toimivan. Kun käyttäjä painaa nappulaa pitkään tai pyyhkäisee ruutua laitteesta riippuen, tapahtuu hieman eri asioita. Yksittäinen verkkosovellus ei voi täten täyttää kaikkia käyttäjän olettamia ominaisuuksia. Natiivisovellusten kohdalla haluttu tuntuma saadaan automaattisesti tai vähällä vaivalla. On kuitenkin mahdollista rakentaa ensin yksinkertainen verkkosovellus ja sitten tarkentaa sovelluksen toimintaa ja ulkoasua tärkeimmille laitteille. Täten sama verkkosovellus voi tuntua enemmän natiivisovelluksen kaltaiselta. [8.]

Natiivisovelluksille on saatavilla omat markkinapaikat, josta niitä voi helposti selata. Kehittäjät voivat vapaasti julkaista sovelluksiaan, ja käyttäjät voivat kirjoittaa niille arvosteluja. Käyttäjät voivat täten keskitetysti löytää kiinnostavia sovelluksia, ja hyvät arvostelut saavat ihmiset lataamaan ja kokeilemaan sovelluksia. Verkkosovelluksilla ei ole vastaavaa markkinapaikkaa, joten miten käyttäjät löytävät ne? Verkkosovelluksia voi löytää verkosta kuten muutakin verkkosisältöä: osoitteen tai hakukoneen avulla. Linkkejä voidaan jakaa sähköpostitse, tekstiviestillä tai sosiaalisen median välityksellä ja sen avaaminen käynnistää sovelluksen ilman, että käyttäjän tulee asentaa mitään. [8.]

Uutiset ovat täynnä juttuja, miten kuusivuotias teki miljoonia myyvän mobiilisovelluksen. Ei ole siis mikään ihme, että useat kehittäjät haluavat rakentaa natiivisovelluksia niiden markkinapaikoille. Jos kerran verkkosovelluksia ei saa markkinapaikoille niin miten niitä voidaan rahoittaa? Sovelluksen ostaminen ei ole ainut tapa tehdä rahaa. Mainostaminen ja sponsorit ovat rahoittaneet verkkosivuja jo pitkään ja sama periaate toimii hyvin myös verkkosovelluksille. On myös mahdollista tehdä niin sanottu kehyssovellus, joka ei tee mitään muuta kuin avaa verkkosovelluksen, jokaiselle markkinapaikalle. Täten verkkosovellus on helpommin löydettävissä ja arvosteltavissa kuten muutkin sovellukset. [8.]

Tähän väittelyyn ei ole yhtä voittajaa. Kummallakin sovellustyypillä on omat hyödyt ja haitat. Päätös kehitysalustasta tulee tehdä sovelluksen vaatimien ominaisuuksien mukaan. Aiheeseen tutustuminen helpottaa ratkaisun tekemistä ja nostaa esille asiat, joita tulisi miettiä ennen sovelluksen kehityksen aloittamista.

4 Selailu ilman verkkoyhteyttä

Verkkosovellukset ovat yleisesti aina vaatineet verkkoyhteyden toimiakseen. HTML5:n myötä verkkosovelluksien käyttö ilman internetyhteyttä on mahdollista. Sovellus voi olla joko kokonaan offline-käytettävä tai sen tärkeimmät osat on viety selaimen muistiin, jotta sovellus toimii nopeasti.

Cache manifest (välimuisti luettelo) on tiedosto, joka mahdollistaa tiedostojen tallentamisen offline-käyttöön. Cache manifest on tavallinen tekstitiedosto, joka voidaan luoda millä tahansa tekstieditorilla. [9.]

```
CACHE MANIFEST:
/index.html
/main.css
/scripts.js
```

Kuvio 10. Yksinkertainen cache manifest

Kuviossa 10 on erittäin yksinkertainen cache manifest -toteutus. CACHE MANIFEST-avainsanan jälkeen on listattu tiedostot, jotka tallennettaan selaimen muistiin. Cache manifest otetaan käyttöön merkitsemällä html-elementtiin manifest-määreeseen viite kuten kuviossa 11.

```
<html manifest="offline.manifest">
```

Kuvio 11. Manifestin määritys html-elementissä

Manifestiin voidaan määritellä kolme osiota: CACHE, FALLBACK ja NETWORK. CACHE-osio kertoo, mitkä tiedostot tallennettaan offline-käyttöä varten. Mikäli osioita ei ole määritelty kuten kuviossa 10, tiedostot oletetaan olevan CACHE-osion alla. Verkkosivuilla on usein paljon sisältöä, eikä ole oleellista tallentaa jokaista sivua tai kuvaa offline-tilaan. FALLBACK-osio kertoo, mitä tehdään pyynnöille, jotka osoittavat sivuille, joita ei ole määritelty CACHE-osiossa. NETWORK-osio kertoo, mitkä tiedostot tarvitsevat aina internetyhteyden toimiakseen eikä pitäisi ikinä tallentaa välimuistiin. On hyvä huomata, että kaikki sivut, jotka sisältävät manifest-määreen, tallennetaan selaimen muistiin huolimatta, mitä manifestissa lukee. [9.]

```
CACHE MANIFEST:
#Tämä on kommenttirivi

CACHE:
/index.html
/main.css
/scripts.js

FALLBACK:
/img/ /offline.png

NETWORK:
/post.php
```

Kuvio 12. Osioitu manifest-tiedosto

Kuviossa 12 on manifest-tiedosto, johon on lisätty FALLBACK- ja NETWORK-osiot. FALLBACK-osiossa kerrotaan, että kaikkien img-kansion sisältävien tiedostojen tilalla näytetään offline.png. Tässä esimerkkinä voisi olla keskustelupalsta, jossa käyttäjillä on henkilökohtaiset profiilikuvat eikä niitä haluta tallentaa selaimen muistiin offline-käytössä. NETWORK-osiossa on palvelimella sijaitseva php-skritpi, joka ottaa vastaan käyttäjän viestejä. Tämä ei voi toimia oikein, jos verkkoyhteyttä ei ole saatavilla.

Jos sivut sisältävät dynaamista sisältöä kuten keskustelupalstaesimerkissä uusia viestejä, ne eivät tule näkyviin. Jos selain ei näe muutoksia manifest-tiedostossa, se tarjoaa aina välimuistissa olevaa tiedostoa, vaikka palvelimella olisi päivitetty tiedosto. Jos uusin tietosisältö halutaan selaimen käyttöön, manifest-tiedostoon on tehtävä muutos. Ei ole oleellista, mitä osioita tiedostossa muutetaan. Esimerkiksi manifestin alussa olevan kommenttirivin perään voi päivittää versionumeron. Mikäli tiedostoihin tulee paljon päivityksiä, on mahdollista luoda dynaaminen manifest-tiedosto, jossa generoidaan joka pyynnön yhteydessä uusi aikaleima esimerkiksi php:lla. Tällöin aina kun selain pyytää manifestia, se huomaa muutoksen ja hakee tiedostot uudelleen. Jos käyttäjä on offline-tilassa eikä saa yhteyttä manifest-tiedostoon, käytetään selaimen muistissa olevia tiedostoja. [9.]

Cache-manifestin jatkuva päivitys ja tiedostojen jatkuva uudelleenlataus tuntuu jotenkin pakotetulta ratkaisulta, jos käytössä ei ole mitään muita offline-teknologioita. Jatkuva tiedostojen päivittäminen saattaa myös aiheuttaa turhan pitkiä latausaikoja mobiilioptimoituihin sovelluksiin. Cache-manifesti tuntuu kuitenkin erityisen sopivalta tavalta tallentaa staattisia tiedostoja kuten kuvia sekä JavaScript- ja CSS-tiedostoja, joiden sisältö muuttuu vain harvoin.

4.1 Selaimen tietojärjestelmät

Piparit (cookies) on pitkään käytetty nimitys verkkoliikenteen mukana liikkuvalla tiedolle. Se perustuu vanhaan ohjelmointitekniikkaan, jossa kaikki sivujen välillä säilyvä tieto lähetetään edestakaisin selaimen ja palvelimen välillä jokaisen pyynnön yhteydessä. Esimerkkitapauksena voi toimia yksinkertainen verkkokauppa, jossa käyttäjän valitsemat tuotteet tallennetaan pipareihin. [4.]

Piparit eivät kuitenkaan ole täydellinen vaihtoehto tiedon tallentamiseen. Piparille varattu koko on yleisesti 4 kilotavua, eivätkä ne täten sovi suurempien dokumenttien välitykseen. Toinen suuri ongelma on, että kakki piparit lähetetään edestakaisin jokaisen sivun latauksen yhteydessä. [4.] Jatkuva tiedon pallottelu selaimen ja palvelimen välillä lisää tiedonsiirron käyttöä, joka on muutenkin hyvin rajattu mobiiliyhteyksillä.

Suurinta osaa verkkosivujen välillä tarvittavista tiedoista ei kuitenkaan olisi tarpeen lähettää palvelimelle joka pyynnön yhteydessä. HTML5:n WebStorage mahdollistaa tiedon tallentamisen ja hakemisen selaimen muistista JavaScriptillä. Täten esimerkiksi verkkokaupan palvelimen ei tarvitse tietää käyttäjän ostoksista ennen varsinaista ostostapahtumaa. WebStorage mahdollistaa myös suuremman tietomäärän käytön, joka vaihtelee selaimittain muutamissa megatavuissa. [4.]

WebStorage koostuu kahdesta osasta: localStoragesta (paikallinen tietovarasto) ja sessionStoragesta (istuntokohtainen tietovarasto). Kuten nimestä voi päätellä, localStorage tarjoaa pidempiaikaisen tallennuspaikan, joka säilyy selaimen muistissa, vaikka selain sammutettaisiin välissä. SessionStorage pitää tiedot muistissa vain niin kauan kuin käyttäjän selainikkuna pysyy auki. [4.]

Sekä local- että sessionStorage käyttävät samaa rajapintaa ja käyttäytyvät samalla tavalla lukuun ottamatta niiden säilymisaikaa selaimen muistissa. WebStorageen voi tallentaa merkkijonotyyppisiä avain-arvopareja. [4.]

```
localStorage.setItem('avain','arvo');
```

Kuvio 13. Avain-arvoparin asettaminen WebStorgeen

Kuviossa 13 on pitkän kaavan esimerkki, miten WebStorageen asetetaan arvo. Arvoja voidaan asettaa myös lyhyemmin käyttäen olionotaatiota tai taulukkomerkintää kuten kuviossa 14. [4.]

```
localStorage.avain = 'arvo';
localStorage['avain'] = 'arvo';
```

Kuvio 14. Eri tavat asettaa arvoja WebStorageen

Tiedon hakeminen WebStoragesta on yhtä suoraviivaista kuin sen tallentaminen kuten kuviosta 15 ilmenee.

```
localStorage.getItem('avain');
localStorage.avain;
localStorage['avain'];
```

Kuvio 15. Tiedon hakeminen WebStoragesta

Tiedon hakemisen ja lisäämisen lisäksi rajapinta kertoo, kuinka monta arvoa sinne on tallennettu sekä tarjoaa funktiot yksittäisen tai kaiken tiedon poistamiseen kuten kuviossa 16. [4.]

```
localStorage.length; //kertoo tallennettujen arvojen määrän
localStorage.removeItem('avain'); //poistaa tietyn tiedon
localStorage.clear(); //tyhjentää localStoragen
```

Kuvio 16. WebStoragen muut funktiot

WebStoragen vahvuus piilee sen yksinkertaisuudessa ja laajassa selaintuessa. Sitä tuetaan lähes kaikissa selaimissa Internet Explorer 8:sta lähtien. [4.] WebStoragen heikkouksiin voi luetella sen, että se tarjoaa vain avain-arvoparien tallentamisen ja täten esimerkiksi tietojen järjestäminen metadatan kuten päivämäärän mukaan voi olla työlästä tai tehotonta.

4.2 Selaimen tietokannat

WebStorage sopii hyvin yksinkertaisten avain-arvoparien tallentamiseen, mutta usein tietoa pitää voida käsitellä, vertailla tai lajitella jonkin metadatan mukaan. Tähän tarkoitukseen käytetään usein tietokantoja.

HTML5:n ehdotuksessa on käsiteltävänä kaksi erilaista tietokanta ratkaisua: WebSQL ja IndexedDB. WebSQL käyttää useille kehittäjille jo tuttua SQL-syntaksia, joka mahdollistaa kyselyiden tekemisen tietokannalle. IndexedDB on hieman erilainen tietokanta, jossa ei käytetä tauluja samalla tavalla kuin SQL:ssä. [4.] Valitettavasti selainvalmistajien erimielisyyksien takia kumpaakaan tietokantamallia ei tueta kaikissa selaimissa. [10.] Tästä johtuen tietokannan käyttäminen verkkosovelluksessa edellyttäisi molempien tapojen käyttämistä rinnakkain, eikä siltikään toimisi vanhemmilla selaimilla.

4.3 JSON:in hyödyntäminen

Yksi tapa tallentaa metatietoja selaimen muistiin on JSON:in käyttö. JSON on JavaScriptin oma notaatio, joka voidaan helposti muuttaa tekstiksi ja täten tallentaa WebStorageen. [4.]

```
{
  "id" : "54",
  "content" : "Jotain sisältöä"
  "time" : "1359556020"
}
```

Kuvio 17. Esimerkki tekstimuotoisesta JSON:ista

Kuviossa 17 on JSON esitettynä tekstimuodossa. Tekstimuotoisen JSON:in voi muuttaa JavaScript olioksi JSON.parse-funktiolla, jonka jälkeen sen kenttiin pääsee käsiksi kuten tavallisissa JavaScript-olioissa. [4.]

```
var JSONobject = JSON.parse(jsonText); //muuttaa tekstin JavaScript olioksi
console.log(JSONobject.content); //tulostaa kentän content arvon "Jotain sisältöä"
```

Kuvio 18. Tekstimuotoisen JSON:in muutos JavaScript-olioksi

Kuviossa 18 on tekstimuotoisen JSON:in muunnos JavaScriot-olioksi. Vastaavasti JavaScript olio voidaan muuttaa tekstiksi JSON.stringify-funktiolla ja tallentaa WebStorageen. Latausvaiheessa teksti muutetaan taas takaisin JavaScript-olioksi. [4.]

5 Visuaalisempia websivuja

5.1 Mobiililaitteelle sopivat näkymät

Jotta verkkosivut näyttäisivät järkeviltä ja luettavilta, myös pienemmillä näytöillä kuten älypuhelimilla on otettava käyttöön erityismäärityksiä. Oletuksena selaimet yrittävät zoomata sivun mahdollisimman kauas ja pyrkivät näyttämään koko sivun, jolloin käyttäjä joutuu itse zoomaamaan ja vierittämään ennen kuin teksti on luettavaa. [9.]

Näkyvän alueen (viewport) kokoa voi säätää määrittelemällä sivun alkuun metamääreen. Tämän metamääreen avulla voi määritellä, miten sivun sisältö piirtyy ruudulle. [9.]

```
<meta name="viewport" content="width=device-width">
```

Kuvio 19. Viewport metamääre

Kuvion 19 metamääre kertoo selaimelle, että näkyvän alueen leveys on oletuksena laitteen leveys. Täten käyttäjän ei tarvitse zoomata nähdäkseen sivun tekstiä, mutta osa sivusta on ruudun ulkopuolella.

Näkymän leveyden lisäksi on mahdollista määritellä sen korkeus, zoomausaste ja joko estää tai sallia käyttäjän zoomauseleet. [9.]

Pelkän zoomauksen ja näkyvän alueen hallinta ei riitä, jos sivusto halutaan oikeasti mobiililaitteen ruudulle sopivaksi. Sivun tulisi osata asetella elementit siten, että käyttäjän ei tarvitse vierittää näkymää vasemmalta oikealle lukeakseen koko tekstisisällön tai nähdäkseen koko kuvan. Tällaista sivustoa kutsutaan responsiiviseksi verkkosivuksi.

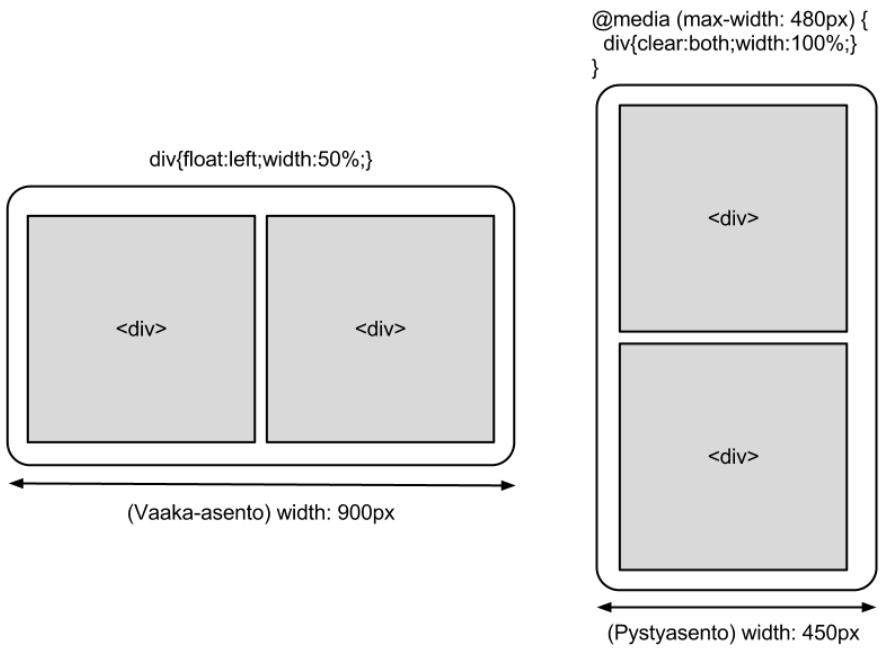
Responsiivisuuden voi saavuttaa käyttämällä media queryitä. Media queryillä voi tunnistaa laitteen ominaisuuksia kuten sen leveyden tai asennon (orientation). Media queryn voi määrittää joko suoraan link-tagissa, joka viittaa CSS-tiedostoon tai itse CSS-tiedostossa kuten kuviossa 20. [9.]

```
<link rel="stylesheet" media="screen and (max-width: 480px)" href="small.css">

@media (max-width: 480px){
  /* toteutettavat CSS määreet jos laitteen leveys on pienempi kuin 480px */
}
```

Kuvio 20. Media queryiden määritystavat

Kuvio 21 pyrkii selventämään responsiivisen sivuston toimintaa. Suurella näytöllä 2 div-elementtiä ovat 50 % näytön koosta ja vierekkäin. Media query kuitenkin määrittelee, että näytön leveyden ollessa pienempi kuin 480 pikseliä div-elementit eivät salli muita elementtejä vierelleen ja vievät 100 % ruudun leveydestä. Täten sivun elementit sopivat paremmin pienemmälle näytölle siten, ettei teksti litisty liian pieniin laatikoihin tai menee ruudun ulkopuolelle.



Kuvio 21. Media queryn havainnollistaminen

5.2 Sivusiirtymät ja animaatiot

Verkkosivuilla on jo pitkään näkynyt erilaisia animaatioita ja erikoistehosteita, joita on tehty erilaisilla JavaScript-kirjastoilla. CSS3 pyrkii tuomaan mukanaan uusia työkaluja verkkosivujen animointiin. Nämä työkalut ovat siirtymät (transition), animaatiot (animations) ja muunnokset (transforms). [3.]

Kun elementteihin määritellään CSS-tyylejä, selain muuttaa elementtien tyylit välittömästi. Siirtymien tarkoitus on tasoittaa tyylien muuttumista. Esimerkiksi kun hiiri viedään linkin päälle, linkin väri muuttuu. Siirtymän avulla on mahdollista päättää miten nopeasti tekstin väri muuttuu kuten kuviossa 22. [3.]

```
a:hover{
  color: red;
  transition-delay: 100ms; /* viive ennen siirtymää */
  transition-property: color; /* lista, mihin ominaisuuksiin siirtymä vaikuttaa  */
  transition-duration: 500ms; /* aika, miten nopeasti ominaisuus muuttuu toiseksi */
}
```

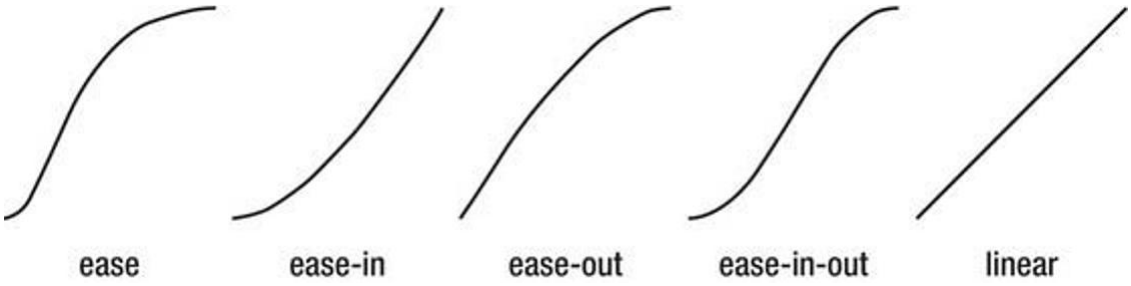
Kuvio 22. Esimerkki linkin värin muuttumisesta

Siirtymille on myös mahdollista asettaa ajoitus. Ajoituksia on viittä eri tyyppiä: ease (oletus), linear, ease-in, ease-out ja ease-in-out. [3.] Esimerkiksi kuvion 22 tapauksessa väri halutaan muuttuvan tasaisesti alusta loppuun, joten ajoitusfunktioksi asetetaan linear kuten kuviossa 23.

```
a:hover{
  transition-timing-function: linear;
}
```

Kuvio 23. Ajoitusfunktion käyttö siirtymissä

Kuvion 24 käyrät kuvaavat eri ajoitusfunktioiden aikajakaumia.



Kuvio 24. Ajoitusfunktion käyrät [3.]

Animaatio on idealtaan vastaava kuin siirtymä, mutta se tarjoaa mahdollisuuden asettaa useampia välipisteitä (keyframes). Animaatiolle voi myös määrätä toistokerrat, ja se palaa oletuksena alkuasemaansa toiston jälkeen. [3.]

```
a:hover{
  animation-delay: 100ms;
  animation-duration: 500ms;
  animation-timing-function: linear;
  animation-iteration-count: infinite; /* animaatiota toistetaan ikuisesti */
  animation-name: 'turnRed'; /* käytettävät välipisteet */
}
@keyframes turnRed{
  to{
    color: red;
  }
}
```

Kuvio 25. Linkin värjääminen punaiseksi ja takaisin

Kuviossa 25 on animaation määritys, jolla on yksi välipiste. Animaatiota toistetaan loputtomasti, eli linkki muuttuu vuorotellen punaiseksi ja takaisin oletusväriksi 500 ms välein.

Animaatiolle on mahdollista määritellä enemmän välipisteitä. Kuviossa määritelty to-välipiste vastaa prosenttilukua 100. [3.] Jos teksti halutaan muuttaa vihreäksi ennen kuin se muuttuu punaiseksi, määriteltäisiin 50 % välipiste kuten kuviossa 26.

```
@keyframes turnGreenRed{
  50%{
    color: green;
  }
  to{
    color: red;
  }
}
```

Kuvio 26. Linkin värjääminen vihreäksi ja sitten punaiseksi

Muunnokset mahdollistavat elementtien siirron (translate), pyörityksen (rotate), skaalauksen (scale) ja kääntämisen (skew). [3.]

```
.upsidedown{
  transform: rotate(180deg);
}
```

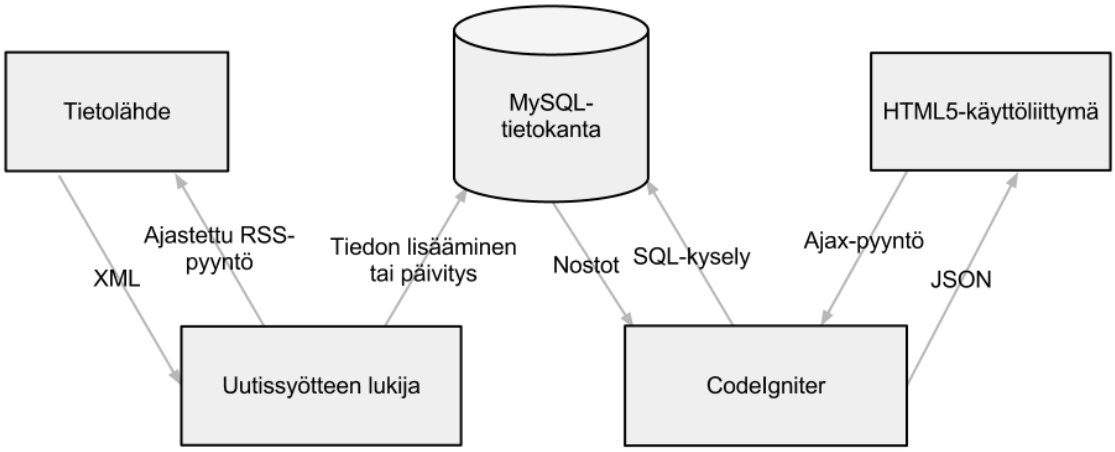
Kuvio 27. CSS-luokka joka kääntää sisällön ylösalaisin

Kuviossa 27 on esimerkki CSS-muunnoksesta, joka kääntää elementin ylösalaisin. Muunnoksia voi käyttää myös siirtymien ja animaatioiden yhteydessä.

6 Oman HTML5-demon toteutus

Tavoitteena on tehdä demo käyttäen oppimiani HTML5-teknologioita. Demon teema on uutisten lukija, joka poimii tiedot jostain verkkosivulta ja tuo ne käyttäjän luettavaksi mobiililaitteille. Demon on tarkoitus vastata näköislehteä, eli sen voi ottaa kätevästi mukaan junaan tai bussiin ilman, että internetyhteyttä on koko ajan saatavilla. Yhteyden ollessa saatavilla artikkelien tulee päivittyvä automaattisesti. Lehti tulee olla luontevasti luettavissa erilaisilla mobiililaitteilla ja sen tulee skaalautua erikokoisille näytöille. Myös mobiilille oleelliset toiminnot kuten kosketuseleet on otettava huomioon. Artikkelista voi hypätä toiseen pyyhkäisemällä ruutua ja sivun sisällä voi olla sisäisiä vierityspalkkeja.

HTML5 ei ole kuitenkaan tarkoitettua massiivisten datamäärien lataamiseen ja parsimiseen vaan keskittyy pääasiassa näkymiin. Tämän takia demo tarvitsee tuekseen taustajärjestelmän. Ennen kun menen varsinaiseen HTML5-osuuteen, selitän lyhyesti, miten sovelluksen taustajärjestelmä rakentuu.



Kuvio 28. Demon arkkitehtuurikuvaus

Kuvio 28 kuvaa demon arkkitehtuuria. Taustajärjestelmä jakautuu kahdeksi osaksi: RSS-uutissyötteen lukijaan ja CodeIgniterin MVC-mallia noudattavaan sovelluskehykseen.

Taustaohjelma on jatkuvasti ajossa oleva php-skripti, joka hakee ja parsii minuutin välein RSS-uutissyötteitä. Parsittua tietoa verrataan tietokantaa vasten, jolloin uudet artikkelit lisätään tietokantaan ja päivitetyt artikkelit korvataan uusilla tiedoilla.

Sovelluksen käynnistyksen yhteydessä HTML5-käyttöliittymä tekee Ajax-pyyntöjä CodeIgniterin controllereille. CodeIgniterin controller kutsuu modelia, joka muodostaa oikeanlaiset SQL-kyselyt ja hakee tietokannasta artikkeleita, kategorioita ja etusivujen nostoja pyynnön tyypistä riippuen. Tietokannasta haetuista tiedoista muodostetaan sopivankokoisia tietopaketteja, jotka välitetään käyttöliittymälle JSON-muodossa aikaleiman kanssa. Jatkossa tehtävien Ajax-pyyntöjen yhteydessä välitetään edellisen pyynnön mukana saatu aikaleima, jota verrataan vastaavasti uutisten aikaleimoihin. Täten ensimmäisen latauskerran jälkeen saatavat tietosisällöt ovat huomattavasti pienempiä kuin ensimmäinen.

Demon käynnistyessä ladataan html-sivu, jossa ei ole JavaScript- ja CSS-tiedostojen lisäksi muuta kuin tyhjiä elementtejä. Tyhjät elementit muodostavat sivun rakenteen, johon tullaan myöhemmin lisäämään sisältöä dynaamisesti. Html-elementtiin on määritelty manifest-tiedosto, jossa on listattu kaikki sovelluksen toimintaan oleellisimmat tiedostot. Sivun latautuessa selain tarkistaa onko tiedostot tallennettu Application Cacheen tai onko manifest-tiedosto muuttunut edellisestä tallennuksesta jälkeen ja

päivittää tiedostot. Tässä vaiheessa sovellus voidaan periaatteessa käynnistää offline-tilassa, vaikka se ei vielä tee mitään.

Manifestissa on myös määritelty NETWORK-osio, jossa on pelkkä jokerimerkki (*) kuten kuviossa 16. Tämä tarkoittaa, että kaikki tiedostot ja pyynnöt, joita ei ole listattu CACHE-osioon, ohittavat cache manifestin. Jos tätä määritystä ei ole, Ajax-kutsut eivät pääse manifestin ohi.

```
NETWORK:
*
```

Kuvio 29. Manifestin NETWORK-osio

Kun dokumentin rakenne ja scriptit on ladattu selaimen muistiin, alkaa varsinainen tietosisällön lataus. Tietosisältö ladataan neljässä eri osassa: sivun tiedot, kategorioiden tiedot, kategorioiden artikkelinostot ja itse artikkelit.

Jos selaimen muistissa ei ole entuudestaan mitään tietoa, palvelimelta pyydetään oletusasetuksia. Asetuksissa tulee tieto, mitä sivustoja on saatavilla, ja oletussivun id-numero, jonka avulla voi tehdä lisää pyyntöjä.

```
$.ajax({
 type: 'GET',
 url: '/reader/getSites',
 dataType: 'json',
 cache: false,
 contentType: 'application/json',
 success:function(data) {
   //tallenna JSON tieto ja käytä sitä uusien kutsujen tekemiseen
 }
});
```

Kuvio 30. Oletusasetuksien pyytäminen taustajärjestelmästä Ajaxilla

Kuvassa 30 pyydetään sivuston oletusarvoja Ajax-kutsulla. Ajax-kutsun vastaus tulee JSON-muodossa. JSON on suoraan JavaScript-olio, jolta voidaan pyytää yksittäisiä ominaisuuksia.

```
alert(data.id); //ilmoittaa oletussivun id:n
```

Kuvio 31. Ajax-kutsun vastauksen käyttäminen

Esimerkkikoodissa 31 käytetään Ajax-kutsulta vastauksena saadun data-muuttujan id-arvoa.

JavaScriptin JSON-rajapinta mahdollistaa JSON-datan muuttamisen tekstiksi ja tekstistä takaisin JSON-olioksi. Tämä on erityisen hyödyllistä, sillä localStorageen voi tallentaa vain tekstimuotoista tetoa.

```
localStorage['site'] = JSON.stringify(data);
```

Kuvio 32. JSON-olion muutos tekstiksi ja tallennus localStorageen

Esimerkkikoodissa 32 oletusasetukset tallennetaan tekstinä site-nimiseen localStorage-muuttujaan. Tieto voidaan vastaavasti hakea takaisin JSON-olioksi kuten esimerkissä 33.

```
var JSONsite = JSON.parse(localStorage['site']);
```

Kuvio 33. JSON-olion parsiminen localStoragesta

Kun oletustiedot on ladattu ja tallennettu localStorageen, tehdään id:n perusteella 3 samanaikaista pyyntöä. Ensimmäinen pyyntö lataa näytettävien kategorioiden nimet ja id:t. Toinen lataa id-listan, joka kertoo, mitkä artikkelit tulee näkyä minkäkin kategorian etusivulla, ja kolmas pyyntö lataa varsinaiset artikkelit.

Kategorioiden ja artikkelien tallennus tehdään hieman eri tavalla, sillä ei ole järkeä parsia ja etsiä isoa tietomäärää joka kerta, kun käyttäjä vaihtaa artikkelia. Tämän takia

JSON-taulukko iteroidaan läpi ja jokaiselle artikkelille tehdään oma localStorage-muuttuja.

```
var JSONsite = JSON.parse(localStorage['site']); //ladataan oletusasetukset
var JSONarticles = data; //sijoitetaan artikkelit kuvaavaan muutujanimeen
for(var i = 0;i < JSONarticles.articles.length; i++){
 localStorage['article-'+JSONsite.id+'-'+JSONarticles.articles[i].id] =
JSON.stringify(JSONarticles.articles[i]);
}
```

Kuvio 34. JSON-taulukon läpikäynti

Esimerkkikoodi 34 on otettu Ajax-kutsusta, jossa pyydetään taustajärjestelmältä kaikkia artikkeleita. Avaimet nimetään siten, että niissä on mukana sivuston ja artikkelin id-numerot. Tämä mahdollistaa yksittäisen artikkelin lataamisen nopeasti, kun tiedetään sen id-arvo.

LocalStorage ei ole kuitenkaan tietokanta eikö sinne voi tehdä kyselyitä. Tästä aiheutuu erilaisia ongelmia etenkin listojen muodostuksessa. Jos halutaan listata esimerkiksi tietyn kategorian artikkelit, ei voida tehdä "WHERE category = 1" kaltaisia kyselyitä kuten SQL:ssä, vaan on käytävä läpi koko localStorage. Vastaavasti artikkelien listaus aikajärjestykseen on erityisen hidasta. Listaukseen vaadittavien operaatioiden määrä kasvaa suoraan suhteessa localStoragessa olevien avainten määrään.

Tämä on ehdottomasti yksi suurimmista ongelmista localStoragen käytössä ja suurin hidaste demon nopeudessa. Hidastelua voisi mahdollisesti helpottaa rakentamalla jonkinlaisia aputaulukkoja. Taulukko, jossa on kaikkien artikkelien id:t aikajärjestyksessä, säästäisi huomattavasti aikaa sivun piirtovaiheessa, kun listaa ei tarvitsisi järjestää uudelleen joka kerta. Toisaalta tietosisältö on jatkuvasti muuttuvaa, jolloin taulukoiden ajan tasalla pitäminen saattaa osoittautua aivan yhtä työlääksi kun listojen järjestäminen. Tämä ongelma on hyvä pitää mielessä tulevaisuuden projekteja varten ja miettiä tapauskohtaisesti, miten säästyttäisiin jatkuvalta localStoragen iteroinnilta.

Toinen suuri ongelma localStoragen käytössä on käytettävän muistin määrä. Muistin määrä vaihtelee selainkohtaisesti, yleisesti 2,5 ja 5 megatavun välillä. Tämä on kuitenkin yllättävän pieni, jos sinne pitäisi tallentaa isoja määriä tekstiä tai kuvia.

Uutislukijan pitäisi vastata näköislehteä ja täten sisältää artikkeleita sekä niiden kuvia. Taustajärjestelmä poimii RSS-uutissyötteen lisäksi kuvia alkuperäisistä artikkeleista ja tallentaa ne base64-koodattuna levylle. Base64-koodatut kuvat voidaan lähettää sellaisinaan artikkelien JSON:in mukana. Base64-kuvat voi näyttää verkkosivulla asettamalla ne tavallisen kuvaelementin src-kenttään kuten kuviossa 35.

```
<img src="...">
```

Kuvio 35. Data-urlin käyttö img-tagissa.

Artikkeleita tallennetaan kuitenkin kymmeniä ja jokaisessa artikkelissa voi mahdollisesti olla useampia kuvia. Tämä tarkoittaa, että muisti voi täyttyä yllättävän pienestä määrästä artikkeleita. Nykyisessä localStoragen toteutuksessa ei ole mitään tapaa tutkia, miten paljon muistia on tai miten paljon siitä on käytetty. Tästä johtuen ainut tapa havaita tilan loppuminen on siinä vaiheessa, kun vahinko on jo sattunut.

```
try {
 localStorage['avain'] = 'jotain tietoa...';
} catch(e) {
 if(e.name === 'QUOTA_EXCEEDED_ERR') {
    //poistetaan vanhaa tietoa ja yritetään uudelleen
 }
}
```

Kuvio 36. localStoragen tilanhallinta

Kuviossa 36 on localStorage-rajapinnan ainut tapa hallita tilankäyttöä. Jos tila loppuu, esille tulee sama ongelma kuin tiedon listauksessa. Pitää iteroida artikkeleita läpi ja yksi kerrallaan valita vanhimmista artikkeleista, mitä poistetaan, jonka jälkeen toivotaan, että uusi artikkeli mahtuu sen tilalle. Seuraava artikkeli voi mahdollisesti aiheuttaa virheen uudelleen, jolloin joudutaan taas tekemään lisää tilaa.

Paremman ratkaisuvaihtoehdon puitteissa päädyin ratkaisuun, jossa pyrin siirtämään suurimman vastuun taustajärjestelmälle, jolla on käytössä tehokkaammat työkalut. Kuvat pyritään pakkaamaan mahdollisimman hyvin ja yritetään laskea, että artikkeleiden yhteiskoko ei ylittäisi minimimuistin määrää. Jos selaimen artikkelit ovat hyvin vanhoja, ne tyhjennetään kokonaan ja tilalle ladataan uudet.

Tietojen tallennuksen lisäksi tarvitaan oleellisesti logiikkaa, joka osaa tuottaa tiedoista sivuja. Demossa käytetään yhden sivun tekniikan ideologiaa. Tämä tarkoittaa sitä, että sivua ei missään vaiheessa ladata kokonaan uudelleen vaan pelkästään sen sisältö muuttuu. Yhden sivun tekniikassa linkkien klikkaus ei lähetä pyyntöä palvelimelle ja vastaanota kokonaan uutta sivua. Sen sijaan klikkauksella pyydetään esimerkiksi JSON tai XML-tiedostoa, jonka tietosisältö päivitetään ruudulle. Meidän tapauksessa kaikki tieto ladattiin jo aikaisemmin selaimen muistiin, joten siitä pitää vain tuottaa uusia sivuja.

JQueryllä voi asettaa kuuntelijan, joka seuraa linkkien klikkausta kuten kuviossa 37.

```
$('a').live('click', function(event){ //linkin klikkauksen yhteydessä tapahtuva funktio
 event.preventDefault(); //estää selaimen navigoimasta eteenpäin
 var attr = $(this).attr('href'); //poimii linkin osoitteen
 ...
}
```

Kuvio 37. jQueryn klikkausten kuuntelija

Bind-funktion sijasta käytetään live-funktiota, jotta myös kaikki tulevaisuudessa luotavat linkit käyttäisivät kyseistä tapahtumakuuntelijaa. PreventDefault estää selaimen navigoimasta linkin osoittamaan paikkaan, jotta voisimme itse vaihtaa sivun sisällön.

Demossa on kaksi päänäkymää: kategoria- ja artikkelinäkymä. Käyttäjän tilaa pidetään yllä sessionStoragessa. Se sisältää kolme kenttää: sivuston, kategorian ja artikkelin. Jos artikkelikentällä on arvo, ollaan artikkelinäkymässä. Muuten ollaan etusivulla tai tietyn kategorian näkymässä.

```
<a href="?site=1">Etusivulle</a>
<a href="?site=1&category=3">Kategoriaan 3</a>
<a href="?site=1&category=2&article=103">Artikkeliin 103</a>
```

Kuvio 38. Esimerkkilinkkejä

Kuviossa 38 on esimerkki linkeistä. Linkin parametrit kopioidaan sessionStorageen klikkauksen yhteydessä. Parametrien perusteella sovellus osaa hakea tiedot localStoragesta ja täyttää ne html-koodiin.

Yhden sivun malli aiheuttaa joitain käytettävyysongelmia, esimerkiksi eteen ja taakse nappulat eivät toimi oletetulla tavalla. Koska kaikki tapahtuu samalla sivulla, takaisinpainallus vie käyttäjän pois koko sivustolta sen sijaan, että siirryttäisiin viimeiseksi käytyyn kategoriaan tai artikkeliin.

HTML5:n History API tarjoaa rajapinnan, jolla voidaan tallentaa sovelluksen tiloja ikään kuin ne olisivat käyttäjän selainhistoriassa. Linkkien painalluksia kuunteleva tapahtumakuuntelija kutsuu pushState-funktiota, joka vie uuden tilan muistiin.

PushState-komennolla on kolme parametria. Ensimmäiseen parametriin voi tallentaa JavaScript-olion. Toiseen parametriin voi asettaa tilan otsikon, joka ei ole tällä hetkellä käytössä selaimissa. Kolmantena voidaan välittää osoite, joka tulee näkymään osoitekentässä.

```
var newState = {site : sessionStorage['site'],
 category : sessionStorage['category'],
 article : sessionStorage['article']};
history.pushState(newState, ''); //tallentaa JSON-tiedon history APIin
```

Kuvio 39. pushState-funktion käyttö

Kuviossa 39 on esimerkki pushState-komennon käytöstä. Demossa käytimme vain ensimmäistä parametria, johon tallennamme uuden tilan.

Selaimen takaisin-nappulan painaminen aiheuttaa popstate-tapahtuman, jota kuuntelemalla voimme päättää, mitä tapahtuu, kun käyttäjä painaa takaisin-nappulaa.

```
window.onpopstate = function(event){ //kutsutaan takaisin-nappulaa painaessa
 var poppedState = event.state; //poimitaan aiemmin tallennettu JSON-tieto
 //muutetaan tila vanhaksi tilaksi ja päivitetään näkymä
};
```

Kuvio 40. popstate-tapahtuman käsittely

Kuviossa 40 käsitellään popstate-tapahtuma. Tapahtuma sisältää state-nimisen muuttujan, jolla pääsee käsiksi pushStatessa asetettuihin arvoihin. Nämä arvot asetetaan takaisin sessionStorageen kuvaamaan käyttäjän uutta tilaa, jonka jälkeen sivu rakennetaan uudelleen. Täten käyttäjä on takaisin samalla sivulla kuin ennen linkin klikkausta.

Demon on suunniteltu toimivan suurilla (tietokoneet, tablettien landscape), keskikokoisilla (tablettien portrait) ja pienillä (älypuhelimet) näytöillä. HTML:n head-osioon on asetettu meta-määreitä, jotka kertovat, miten sivusto skaalautuu ruudun koon suhteen.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0">
```

Kuvio 41. viewport-metatagi

Kuvassa 41 on viewpost-määre, joka kertoo, miten sivusto skaalautuu laitteen koon perusteella. Content-ominaisuudessa määritellään, että sivu on laitteen leveyden kokoinen ja että oletuksena käytetään selaimen omaa zoomausta. Siinä kerrotaan myös, miten paljon sivua voi zoomata, tässä tapauksessa sekä maksimi- että minimiarvot on määritelty samaksi eli sovelluksen kokoa ei voi muuttaa. Tämä on tehty sen takia, ettei käyttäjän tarvitse tehdä zoomausliikkeitä jokaisen orientaatiomuutoksen yhteydessä. Koon muuttamisessa kannattaa miettiä myös käytettävyysasioita. Jos zoomaus estetään, olisi hyvä antaa mahdollisuus muuttaa esimerkiksi fontin kokoa, jotta huononäköisemmät ihmiset voivat myös käyttää sovellusta.

Viewport-määreen lisäksi mobiiliverkkosovelluksille on hyvä antaa Applen määrittelemiä metatietoja, joiden avulla käyttäjät pystyvät lisätä kirjanmerkin esimerkiksi tabletin etusivulle.

```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Kuvio 42. Applen webapp-metatagit

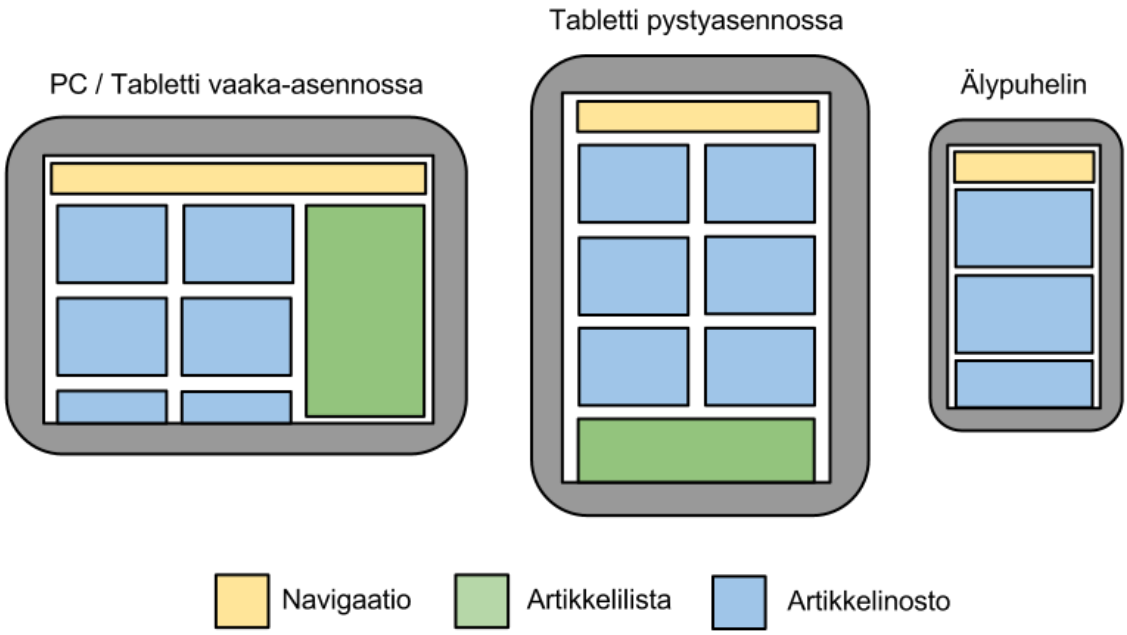
Kuvassa 42 on metamääre, joka mahdollistaa kuvakkeen lisäämisen iOS-laitteen työpöydälle. Toinen määre kertoo, miltä selaimen tilarivi tulisi näyttää, kun verkkosovellus avataan kuvakkeen kautta. On mahdollista asettaa muitakin Applen metatietoja kuten minkälainen kuvake työpöydälle tulee, mutta niitä ei ole toteutettu tässä demossa.

Demossa on karkeasti määritelty, minkä kokoisella näytöllä pääsee mihinkin näkymään.

```
@media all{
  /* Suuren resoluution CSS-määreet */
}
@media only screen and (min-width:550px) and (max-width:900px) {
  /* Keskikokoisen resoluution CSS-määreet */
}
@media only screen and (max-width:550px){
  /* Pienen resoluution CSS-määreet */
}
```

Kuvio 43. Demossa käytetyt media queryt

Kuviossa 43 on demoon määritellyt media queryt. Vain näkymien väliset eroavat on määritelty media query-lohkojen sisällä, kuten miten leveä jokin sivun osa on ja salliiko se muita osia viereensä. Pikselimitat on määritelty hyvin karkeasti siten, että suurin osa nykyisistä mobiililaitteista päätyisi oikeaan näkymään. Oikeissa tuotantoprojekteissa mittojen valinnassa on hyvä ottaa huomioon, millä laitteilla sivustoa oletetaan käytettävän.



Kuvio 44. Sivun osioiden asettelu eri kokoisissa näytöissä.

Kuviossa 44 on pyritty visualisoimaan, miten sivun osiot näkyvät näytön koon mukaan.

Demossa pyrittiin tekemään kosketuslaitteelle luontevia ominaisuuksia kuten sivun sisällä vieritettäviä listoja. Pyrin tekemään artikkelilistan siten, että se pysyisi koko selauksen ajan samassa kohdassa. Lista sisällä olevia artikkelilinkkejä voisi selata ilman, että koko sivua on vieritettävä.

Voisi olettaa, että tällaisen toiminnallisuuden saisi tehtyä suoraan CSS:llä. Tämä ei ole kuitenkaan mahdollista ainakaan vanhemmilla iOS-selaimilla tai vaati epäluontevan kahden sormen eleen, jota laitteiden käyttäjät eivät ole tottuneet käyttämään. Jotta toiminnasta saataisiin yhtenäistä ja luontevaa mahdollisimman monella mobiililaittella, toteutuksen apuna käytettiin valmiiksi tehtyä lisäosaa. Valitsin lisäosaksi iScroll4:n [11.].

```
<div id="wrapper">
 <ul>
   <li></li>
   ...
 </ul>
</div>
```

Kuvio 45. esimerkki iScrollin vaatimasta html-rakenteesta. [11.]

Kuviossa 45 on esimerkki, miltä iScroll4:n vaatima html-rakenne näyttää. On oleellista, että wrapper-luokalle asetetaan tietty koko ja ylimenevät elementit piilotetaan. Tämän jälkeen iScroll pitää ottaa käyttöön JavaScriptissä.

```
var myScroll = new iScroll('wrapper');
```

Kuvio 46. iScrollin käyttöönotto [11.]

Kuvassa 46 on yksinkertaisin mahdollinen iScrollin käyttöönotto. Vaihtoehtoisesti constructorille voi antaa toisena parametrina JSON-konfiguraatiotiedoston, jolla voi määritellä esimerkiksi, mihin suuntaan listaa voi skrollata ja näytetäänkö skrollauksen yhteydessä vierityspalkkia.

Jotta sovellus tuntuisi vielä luontevammalta käyttää mobiililaitteella, on oleellista, että artikkelista toiseen voi liikkua pyyhkäisyeleellä (swipe). Tätä ominaisuutta ei ole vielä määritelty, kun aloitin demon toteutuksen. Valmiiden lisäosien käyttö olisi vaatinut tietynlaisia html-rakennetta ja siitä olisi seurannut erittäin suuria muutoksia muihin jo valmiisiin toimintoihin ja html-rakenteeseen. Tämän takia päätin itse kirjoittaa yksinkertaisen pyyhkäisyn tunnistuksen sekä rakentaa sivunvaihtoanimaation CSS3:n avulla.

Applen dokumentaation [12.] mukaan pyyhkäisyeleen voi tunnistaa seuraavilla ehdoilla:

1. Aloita ele, jos touchstart-tapahtuma sisältää yhden kosketuksen.

2. Keskeytä ele, jos jossain vaiheessa havaitaan enemmän kuin yksi kosketus.

3. Jatka liikettä niin pitkään kun se pystyy pääasiassa x-akselilla.

4. Keskeytä ele, jos se liikkuu pääasiassa y-akselilla.

5. Lopeta ele touchend-tapahtumalla.

Aluksi asetetaan kuuntelija, joka voi tutkia kosketustapahtumia.

```
$(document).bind('touchstart', touchStart)
 .bind('touchmove', touchMove)
 .bind('touchend', touchEnd)
 .bind('touchcancel', touchEnd);
```

Kuvio 47. Kosketustapahtumien sitominen jQueryllä.

Kuvassa 47 sidotaan dokumentaation kuvauksen mukaiset käsittelijäfunktiot kullekin kosketustapahtumalle jQueryllä. Jos päästään touchEnd-funktioon asti ilman, että ele on keskeytetty, tarkistetaan kosketuksen alkukohdan ja loppukohdan välinen erotus x-akselilla. Erotuksen etumerkistä voi päätellä, tapahtuiko pyyhkäisy oikealle vai vasemmalle ja suuruudesta, oliko ele tarpeeksi pitkä.

Sivun rakennuksen yhteydessä sinne on lisätty linkit edelliseen ja seuraavaan artikkeliin tai kategoriaan. Pyyhkäisyeleen tapahtuessa tehdään click-tapahtuma oikealle linkille ja aikaisemmin luotu linkkien klikkausten kuuntelija hoitaa sivun vaihdon. Tässä toteutuksessa ei ole otettu huomioon yksi yhteen liikkumista, joka tapahtuu ennen varsinaista pyyhkäisyelettä ja kertoo käyttäjälle kosketuseletoiminnon olemassaolosta.

Tässä vaiheessa kosketusele vain vaihtaa sivun eikä liu'u nykyisen artikkelin oikealta tai vasemmalta puolelta kuten voisi olettaa. Toteutamme sivusiirtymän CSS-transitiolla, jota varten tarvitsemme kaksi määritystä: ajoituksen sekä koordinaatit. Ajoitus määritellään CSS-tiedostossa.

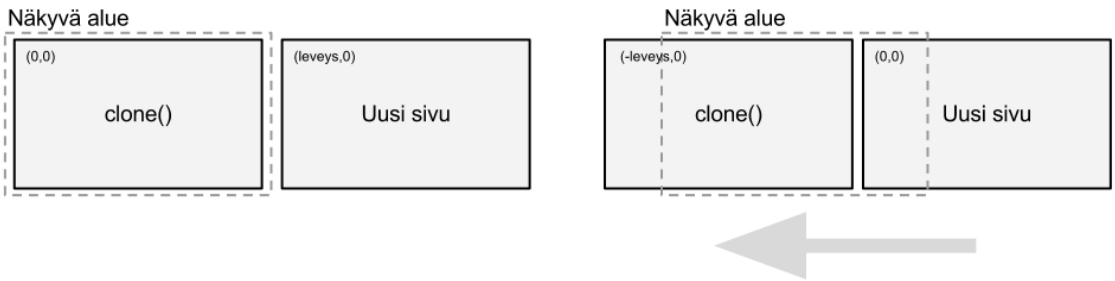
```
.moving{
 -moz-transition: all 0.5s ease-in-out;
 -webkit-transition: all 0.5s ease-in-out;
 -o-transition: all 0.5s ease-in-out;
 transition: all 0.5s ease-in-out;
}
```

Kuvio 48. Ajoituksen määritys CSS-luokassa

Kuviossa 48 on luokka, joka määrittelee liikkuvan kappaleen ajoituksen. Kappale liikkuu 0,5 sekuntia, kiihdyttää vauhtia alussa ja hidastaa ennen loppua. Kyseessä on lyhyt (shorhand) tapa, jolla voi asettaa kaikki transition tarvitsemat tiedot yhdellä rivillä. Transition-määre on edelleen keskeneräinen ja vaatii valmistajien etuliitteet toimiakseen oikein kaikissa selaimissa.

Toinen oleellinen asia animaation kannalta ovat koordinaatit. Näyttöjen koot ja täten liikkuvan alueen suuruus voivat muuttua, joten liikekoordinaatit tulee laskea uudelleen ennen varsinaista animaatiota.

Monet swipe-toimintoa toteuttavat lisäosat näyttivät pitävän muutaman seuraavan sivun piilotettuna DOM:issa ja täten saivat ne nopeasti vaihdettua keskenään eleen tapahtuessa. Demon toteutus oli alun perin suunniteltu pitämään yksi tila kerrallaan, joten animaation yhteydessä joudutaan ensin luomaan kopio nykyisestä sivusta ja rakentamaan uusi sivu ruudun ulkopuolella. Tästä aiheutuu hieman viivettä verrattuna valmiisiin lisäosiin. Hyvänä puolena voi pitää sitä, että laitteen ei tarvitse pitää muistissa kuin yksi sivu kerrallaan.



Kuvio 49. Animaation toteutus

Kuviossa 49 on kuvasarja, joka kuvaa sivusiirtymän tapahtumisen. Aluksi näkyvä alue on nollakoordinaateissa ja piilotettu sivu juuri näkyvän alueen ulkopuolella. Kun näkymille annetaan uudet koordinaatit, ne lähtevät liikkeelle ajoitusluokan määrittelemällä tavalla ja nopeudella.

CSS-animaation etuna verrattuna perinteiseen JavaScript-animaation on se, että se antaa selaimelle mahdollisuuden itse päättää, kuinka moneen näytettävään kuvaan animaatio jakautuu eikä täten jumita hitaampia laitteita. Toinen mahdollisuus toteuttaa vastaava animaatio JavaScriptillä on requestAnimationFrame, mikä oli vielä melko keskeneräinen demon toteutusvaiheessa.

7 Yhteenveto ja pohdinta

Opinnäytetyöni aihepiiri oli erityisen mielenkiintoinen ja ajankohtainen. Mobiililaitteiden suosio on ollut hurjassa kasvussa jo pitkään, ja olen itsekin käyttänyt useita mobiililaitteille suunniteltuja verkkosivuja ja palveluita.

Ennen opinnäytetyötä en tiennyt HTML5:stä paljon mitään, vaan se oli enemmänkin mystinen termi, jonka olin kuullut useaan otteeseen. HTML5 ei kuitenkaan loppujen lopuksi ole vain yksi iso kokonaisuus vaan suuri määrä pieniä parannuksia ja lisäyksiä jo olemassa oleviin työkaluihin. Suurin osa teknologioista oli helppo ymmärtää vanhan tiedon pohjalta, mutta HTML5 sisältää aivan uusiakin toiminnallisuuksia kuten kaksisuuntainen tiedonvälitys palvelimen välillä, canvas-piirtoalustan ja muita aiheita, joihin en varsinaisesti keskittynyt tässä työssä.

Koodin kannalta demoon olisi voinut vielä tehdä reilusti optimointia ja sen olisi voinut saada toimimaan nopeammin. Jos aloittaisin projektin uudelleen nyt osaisin varmasti pienentää koodirivien määrää ja selkeyttää sen rakennetta – sitähän oppiminen on. Pohjalla käyttäisin silti hyväksi havaitsemiani tapoja, jotka esittelin työn demo-osuudessa. Tein demoa käsi kädessä oppimisen kanssa ja sen määrittelyt vaihtuivat hieman ajan kuluessa. Käytännössä tämä tarkoitti sitä, että esimerkiksi sivusiirtymien toteutus tuli vaatimukseksi hieman yllättäen enkä ollut osannut ottaa asiaa tarpeeksi huomioon, kun toteutin demon responsiivista sivupohjaa ja offline-tukea. Tarkka etukäteen määrittely olisi säästänyt hieman päänvaivaa projektin keskivaiheessa.

Teknologiana HTML5 vaikuttaa lupaavalta, mutta se on vielä melko keskeneräinen monen asian suhteen. Esimerkiksi offline-käytettävyydessä huomasi, että HTML5:n kehitysryhmillä oli paljon erimielisyyksiä. Nykyinen localStoragen toteutus ei selvästikään ole tarkoitettu demon kaltaisen datan tallennukseen vaan pienempiin määriin tietoa. Nykyisin työkalujen puitteissa localStorage tarvitsisi ainakin paremmat muistinhallinta työkalut: tiedon käytetystä ja saatavilla olevasta tilasta. IndexedDB-tietokanta ei todennäköisesti tule olemaan saatavilla vielä pitkään aikaan siinä määrin, että sitä voisi käyttää tuotantoprojekteihin etenkin, jos vanhemmat selaimet on oltava tuettuja.

Puutteista huolimatta HTML5:n omaksuminen on hyödyllistä kaikille verkkosovellusten kehittäjille ja sen ominaisuuksia kannattaa ottaa käyttöön siinä määrin kuin mahdollista. Tulevaisuudessa yhä useampi selain tulee tukemaan HTML5:n ominaisuuksia. Vanhempien selainten yhteensopivuusongelmiin löytyy jo nyt suuri määrä erilaisia apukirjastoja. Mielestäni uutta verkkopalvelua suunnitellessa kannattaisi aina ottaa huomioon responsiivinen näkökulma, vaikka sitä ei vielä edellytettäisikään. Pienellä lisävaivalla sivunpohjan rakennusvaiheessa voi säästää suuria määriä aikaa, jos ja kun sivustosta joskus halutaan tehdä responsiivinen.

En näe, että HTML5 tulee korvaamaan natiivisovelluksia, eikä sen tarvitsekaan. Isot laskentaoperaatiot ja graafiset sovellukset toimivat huomattavasti paremmin laitteen natiiveilla kirjastoilla kuin selaimen JavaScript-tulkissa. On mahdollista, että tulevaisuudessa mobiiliselaimet kehittyvät ja pystyvät kilpailemaan natiivisovellusten kanssa, mutta tämä tuskin tapahtuu ihan lähiaikoina. HTML5 soveltuu erityisen vähemmän laskentatehoa vaativiin tehtäviin ja responsiivisten verkkosivujen kehitykseen.

Lähteet

1 HTML5 W3C Working Draft 11 October 2012. 2012. Verkkodokumentti. <http://dev.w3.org/html5/spec/introduction.html>. Luettu 21.10.2012.

2 FAQ - WHATWG Wiki. Verkkodokumentti. <http://wiki.whatwg.org/wiki/FAQ>. Luettu 21.10.2012.

3 Freeman Adam. 2011. The Definitive Guide to HTML5. Apress.

4 Lubbers Peter, Albers Brian, Salim Frank. 2011. Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. Apress.

5 Mark Pilgrim. 2010. Dive Into HTML5. Verkkodokumentti. <http://diveintohtml5.info/>. Luettu 21.10.2012.

6 Modernizr. Verkkodokumentti. <http://modernizr.com/docs/>. Luettu 21.10.2012.

7 Selectivizr. Verkkodokumentti. <http://selectivizr.com/>. Luettu 4.11.2012.

8 Michael Mahemoff. HTML5 vs Native: The Mobile App Debate. 2011. <http://www.html5rocks.com/en/mobile/nativedebate/>. Luettu 27.12.2012.

9 Preston Scott. 2012. Learn HTML5 and JavaScript for iOS. Apress.

10 Arun Ranganathan. Beyond HTML5: Database APIs and the Road to IndexedDB. 2012. Verkkodokumentti. <http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb>. Luettu 31.3.2013.

11 iScroll 4. 2011. Verkkodokumentti. <http://cubiq.org/iscroll-4>. Luettu 27.1.2013.

12 Safari Web Content Guide: Handling Events. 2012. Verkkodokumentti. <http://developer.apple.com/library/ios/#DOCUMENTATION/AppleApplications/Reference/SafariWebContent/HandlingEvents/HandlingEvents.html>. Luettu 27.1.2013.