TAMPEREEN
AMMATTIKORKEAKOULU

HTML-SOVELLUSKEHYKSEN KEHITYS

Saara Pekkala

Opinnäytetyö
Huhtikuu 2018
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

SAARA PEKKALA:
HTML-sovelluskehyksen kehitys

Opinnäytetyö 51 sivua, joista liitteitä 1 sivua
Huhtikuu 2018

Opinnäytetyön aiheena oli kehittää HTML-sovelluskehys messutapahtumien ohjelmanäytöksi. HTML-sovelluskehys oli tarkoitus kehittää moderneilla web-tekniikoilla sekä hyvän käytettävyyden ja asiakkaan toiveiden pohjalta. Tarve kehitykselle syntyi työelämän tarpeista. Modernit web-tekniikat määräsivät käytettäviksi tekniikoiksi perus HTML-, CSS- ja JavaScript-teknologiat.

Kehityksen tuloksena oli kaksi erillistä versiota, joista toista kehitettiin ensimmäisen version ongelmakohtien pohjalta. Kokonaisuudessaan HTML-sovelluskehystä on käytetty yli viidessä eri tapahtumassa.

Mikäli sovelluskehyksen kehitystä jatkettaisiin, olisi sille suunniteltuna muutama kehityskohde ja ominaisuus. Kehityskohteita olisivat graafinen käyttöliittymä tietojen syöttämiselle ja CMS-serverin tietokantaan relaatio näiden tietojen tallentamiselle. Lisättäviä ominaisuuksia olisivat animaatioiden hallinta ajan ja tyylin suhteen, sekä monikielisyys.

Työssä onnistuttiin asiakkaiden suhteen ensimmäisen kanssa tyydyttävästi ja toisen asiakkaan kanssa kiitettävästi. Työn tuloksena oli suht helpoin muokkauksin uudelleen käytettävä sovelluskehys messutapahtumiin, ja sitä on käytetty myös kehitystyön jälkeen muissa tapahtumissa.

Asiasanat: JavaScript, HTML, CSS, käytettävyys, DOM

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

SAARA PEKKALA:
HTML-framework development

Bachelor's thesis 51 pages, appendices 1 pages
April 2018

The aim of this thesis was to develop a HTML application framework that provides easily customizable schedule for events and fairs. The need for development arose from the needs of working life.

The subject of the thesis was to develop an HTML-framework with modern web technologies. Other goals for the thesis were to make so that the framework would conform to good usability and to the wishes of the customer. The use of modern web technologies led the used technologies to be basic HTML-, CSS- and JavaScript-technologies.

The development was influenced by the customers' needs concerning the events that were meant to use the framework and the restrictions set by the final runtime environment. As a product of this development were two separate versions of the frameworks, of which the second was developed based on the first versions issues. During the development the framework was used in five different events.

If the development would continue there would be few development areas and features planned. These would be a graphical user interface for user and data input, and a database relation to save this data for the current Content Management Systems server. Some of the features could be animation control considering the duration and style of the animations, and support for multiple languages.

The development succeeded satisfyingly with the first customer and commendably with the second customer. As a product of the development was with somewhat easy modification reusable application framework, and has been used outside the development period.

Key words: JavaScript, HTML, CSS, usability, DOM

SISÄLLYS

1 JOHDANTO 7
2 KÄYTETYT OHJELMOINTIKIELET JA TEKNIIKAT 8
2.1 HTML-merkintäkieli 8
2.2 CSS-tyyliohje 11
2.3 Sovelletut ohjelmointitekniikat 12
2.3.1 DOM-rajapinta ja JavaScript-ohjelmointikieli 13
2.3.2 jQuery-kirjasto 16
2.3.3 JSON-syntaksi 17
2.4 Kehitystyökalut 20
3 KEHITYSPROSESSI 21
3.1 Toimeksianto 21
3.2 Kehitysprosessista 23
3.2.1 Asiakkaan rooli kehitystyössä 23
3.2.2 Muut kehitykseen vaikuttaneet tekijät 24
3.2.3 Testaus 24
3.3 Ensimmäinen versio 24
3.3.1 Visuaalinen suunnittelu ja animointi 25
3.3.2 Käytettävyyden huomioiminen 27
3.3.3 Värit näkymässä 28
3.3.4 Sovelluskehyksen toiminnallisuus 29
3.4 Toinen versio 33
3.4.1 Refaktorointi 33
3.4.2 Muutokset JSON-tiedostoon 34
3.4.3 Ulkoasun eli käyttöliittymän muutokset 35
3.4.4 Virheilmoitukset 38
3.4.5 Rakenteelliset muutokset 39
4 JATKOKEHITYS 42
4.1 Graafinen käyttöliittymä 42
4.2 Tietojen tallennus tietokantaan 42
4.3 Muut mahdolliset kehitysalueet 43
5 POHDINTA 44
LÄHTEET 48
LIITTEET 51

LYHENTEET JA TERMIT

API	(Engl. Application Programming Interface) Ohjelmointirajapinta eli kokoelma aliohjelmia, protokollia ja työkaluja ohjelmistokehityksessä.
ECMAScript	JavaScriptin standardoimiseksi luotu skriptikielimäärittelmä.
CMS	(Engl. Content Management System) Sisällönhallintajärjestelmä.
CSS	(Engl. Cascading Style Sheet) Internet-sivun tyyliohje, joka kertoo, millaisena sivu näytetään.
DOM	(Engl. Document Object Model) Dokumenttioliomalli. Ohjelmointirajapinta HTML-ja XML-dokumenteille.
Framework	Sovelluskehys, määrittelee geneerisen mutta muunneltavan toiminnallisuuden.
GET-kutsu	HTTP-protokollan metodi, jolla voidaan lukea yksittäinen Internet-sivu tai muu resurssi.
HTML	(Engl. Hyper Text Mark-up Language) Merkintäkieli, jota käytetään Internet-sivujen luomiseen.
HTTP	(Engl. HyperText Transfer Protocol) Tiedonsiirtoprotokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.
JavaScript	Löyhästi tyypitetty ja tulkittu pääosin web-ympäristössä käytettävä ohjelmointi kieli.
JSON	(Engl. JavaScript Object Notation) JavaScrip:in olionotaatiota noudattava syntaksi tiedon säilyttämiseen ja välittämiseen.
jQuery	Selainriippumaton JavaScript-kirjasto, joka on suunniteltu helpottamaan asiakaspuolen ohjelmointia.
MIME-tyyppi	Multipurpose Internet Mail Extension -tyyppi. Standardoitu tapa ilmoittaa dokumentin formaatti ja luonne.
UTF-8	Merkistön koodaustapa.
URL	(Engl. Uniform Resource Locator) Internet-osoite.
XML	(Engl. Extensible Markup Language) Tekniikka tekstimuotoisten dokumenttien merkkaamiseen ja käsittelyyn.
XMLHTTPRequest	JavaScript API, jonka avulla voidaan luoda kyselyjä serverille.

W3C	World Wide Web Consortium. Kansainvälinen web-standardien kehittäjä.

1 JOHDANTO

Tämän työn tilaajana on Jidoka Technologies Oy (myöhemmin vain Jidoka), ja sen tavoitteena on luoda HTML-sovelluskehys. Jidoka tarjoaa monenlaisia palveluita, joista muun muassa yksi on heidän erilaiset näyttölaitteensa. Näyttölaitteita myydään ja vuokrataan, ja niille voidaan asettaa video- tai kuvasisältöä. Näyttölaitteelle voidaan asettaa myös verkkosisältöä reitittämällä laitteelle halutun verkkosivun Internet-osoite eli URL.

Tarve HTML-sovelluskehyksen kehittämiselle syntyi, kun Jidoka myi sitä käyttävän palvelun Easyfairs-tapahtumajärjestäjälle. Easyfairs halusi neljään messutapahtumaansa Jidokan näyttölaitteita. Sen sijaan, että näytöille olisi tehty perinteinen staattinen messuohjelmanäkymä, haluttiin kehittää dynaamisempi ratkaisu HTML-tekniikoilla. Tämän kehitystyön tulos on tämän työn HTML-sovelluskehys. Sovelluskehys muistuttaa rakenteeltaan verkkosivua. Valmis sovelluskehys ladataan näyttölaitteelle sen oman sisällönhallintajärjestelmän (Content Management System, CMS) avulla. Näyttölaitteella sovelluskehys ajetaan Chromium-selaimessa.

Sovelluskehystä käytettiin Easyfairsin tapahtumien lisäksi Suomen Kädentaidot tapahtumassa. Tässä työssä esiteltävä ensimmäinen versio kuvaa Easyfairsin käyttöön kehitettyä kokonaisuutta ja toinen versio Tampereen Messuille kehitettyä versiota. Toiseen versioon pyrittiin korjaamaan ensimmäisen version puutteellisuuksia. Sovelluskehyksen ensisijaisena kehitysperiaatteena toimi kunkin tapahtumajärjestäjän tarpeiden huomiointi tapahtumakohtaisesti. Lisäksi ulkoasun ja ominaisuuksien kehityksessä huomioitiin hyvän käytettävyyden edellytykset.

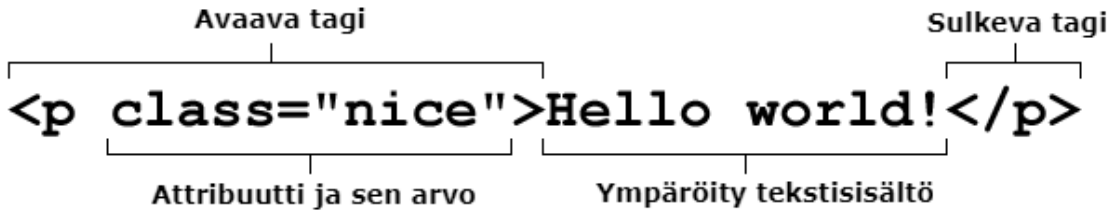
Tässä opinnäytetyössä tarkastellaan sovelluskehyksen kehitykseen liittyneitä valintoja ja niiden perusteita. Lopuksi käydään läpi jatkokehitysmahdollisuuksia ja arvioidaan, miten työssä ja sen tavoitteissa onnistuttiin.

2 KÄYTETYT OHJELMOINTIKIELET JA TEKNIIKAT

Tässä kappaleessa esitellään lyhyesti toteutuksessa käytetyt ohjelmointikielet ja teknologiat. Kunkin teknologian yhteydessä selvennetään myös sen roolia ja käyttöä itse työssä.

2.1 HTML-merkintäkieli

HTML on kuvaileva kieli, joka määrittelee nettisivun rakenteen. HTML-dokumentti on selkokielinen dokumentti, joka on jäsennelty elementeillä. Elementit on ympäröity avaavilla ja sulkevilla tageilla. Kukin tagi alkaa ja loppuu kulmasulkeisiin (< >). On olemassa joitakin tyhjiä tai *void* -tageja, jotka eivät voi sisältää mitään tekstiä, esimerkiksi <img> -tagi. (Mozilla Developer Network 2018).



KUVIO 1. Elementin anatomia (Mozilla Developer Network 2018, muokattu)

Elementin tagit kertovat, minkälaista sisältöä HTML-sivun tulee näyttää. Esimerkiksi kuvion 1 elementin tagi on p, joka kertoo HTML-sivulle, että kyseessä on tekstikappale (Mozilla Developer Network 2018).

Elementeillä voi olla myös erilaisia attribuutteja. Attribuutit sisältävät lisätietoa elementistä, mitä ei haluta näkyvän varsinaisessa sisällössä (Mozilla Developer Network 2018). Kuvion 1 elementille on annettu attribuutti *class*, ja sen arvoksi teksti *nice*. Class-attribuutin arvolla voidaan myöhemmin kohdentaa elementtiin ulkoasun määrityksiä ja muita ominaisuuksia, kuten käyttäytymistä sekä interaktiivisuutta ja dynaamisuutta.

HTML-dokumenttiin kuuluu muutama sille ominainen elementti, joilla viestitään selaimelle mitä näytetään ja millä lailla se näytetään. Seuraavaksi käydään läpi yksinkertaisen HTML-dokumentin malli ja sen sisältävät ja sille tyypilliset elementit.

KUVIO 2. HTML-dokumentti.

Kuviossa 2 HTML-dokumentissa määritellään ensimmäiseksi (rivillä 1) dokumentin tyyppi ”<!DOCTYPE html>” -esittelyllä. Tyyppimääre ei ole HTML -tagi vaan ohje selaimelle siitä, millä HTML-versiolla dokumentti on kirjoitettu (W3Schools n.d.) Kuvion tyyppimääre kertoo dokumentin olevan HTML5-muotoista. Rivillä 2 alkava ”html”-elementti sisältää koko sivun sisällön, ja se kertoo selaimelle, että kaikki sen sisällä tulee tulkita HTML-syntaksiksi. Rivillä 3 alkaa ”head”-elementti, joka toimii säiliönä sivun metatiedoille. Metatietoja ovat muun muassa tiedot avainsanoista, tyyliohjeistus ja linkit muihin resursseihin. Esimerkki metatiedosta on rivillä 4, jossa määritellään sivun merkistökoodaukseksi utf-8. Metatietoja ei näytetä selaimessa ollenkaan, vaan näkyvä sisältö sijoitetaan ”body”-elementin sisälle. Jos kuitenkin halutaan piilottaa elementtejä tai kommentoida HTML-dokumenttia, voidaan piilotettavat elementit tai kommentti kirjoittaa rivin 7 osoittamalla tavalla merkkijonojen ”<!--” ja ”-->” väliin. (Mozilla Developer Network 2018).

HTML:llä muodostettiin työssä sovelluskehyksen runkorakenne eli se, mistä osioista näkymä koostuu. Tähän runkoon lisättiin skriptillä sen ajonaikainen sisältö. HTML:stä käytettiin HTML5:sta, joka on viimeisin versio HTML:stä (W3Schools n.d.) HTML5:n tuo mukanaan uusia elementtejä HTML-syntaksiin, ja näistä työssä käytettiin semanttisia elementtejä ja graafisia elementtejä. Lisäksi HTML5 on esimerkiksi dokumentin tyyppimääre yksinkertaistunut.

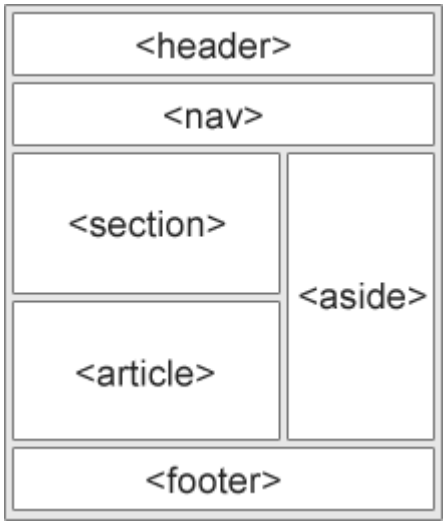
Kuviossa 2 esiteltiin HTML5 dokumentin tyyppimääreen kirjoitusmuoto. Se on yksinkertaisempi kuin edeltävän version, HTML4:n tyyppimääre. HTML4:ssä tyyppimääre alkaa myös ”<!doctype html” -sanoilla mutta vaatii seuraavaksi vielä lisätietoa dokumentissa käytetystä HTML-määrittelystä.

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

KUVIO 3. HTML4-muotoisen dokumentin tyyppimäärittely.

Kuviossa 3 on esimerkki yhdestä HTML4-tyyppimääreestä. HTML4:ssä tyyppimääreitä on kolmea erilaista, ja tyyppimääre määritellään kuvion sitaateissa olevan URL;n avulla. Tämä URL määritteli sen, kuinka ankarasti dokumentin HTML-syntaksi tulkitaan. Esimerkiksi kuviossa 3 on määritelty tulkinnaksi ”strict”, joka hyväksyy dokumentissa kaikki ne HTML-elementit, jotka eivät ole vanhentuneet. (Quackit n.d.).

HTML4-versiolla web-sivun eri osiot merkataan geneeristen ”div”-elementtien avulla, ja niille määriteltiin tunnus ”id”- tai ”class”-attribuutin avulla. Tässä tavassa ei sinänsä ole mitään vikaa mutta selain ei erota eri ”div”-elementtejä toisistaan pelkän attribuutin avulla. (W3C 2014). HTML5:n semanttiset elementit mahdollistavat hakukoneiden tunnistaa web-sivun oikean sisällön (W3Schools n.d.).



KUVIO 4. HTML5 version semanttisia elementtejä (W3Schools n.d.)

Kuviossa 4 on esitettynä joitakin HTML5:n semanttisista elementeistä web-sivun muotoon sovitettuina. Semanttisilla elementeillä on niille suositeltuja käyttötarkoituksia, esi-

merkiksi ”header”-elementin tulisi sisältää sivustoa esittelevää sisältöä, esimerkiksi logon. Elementin ”nav” tulisi sisältää sivuston navigaatioon liittyviä linkkejä tai muita rakenteita, kuten esimerkiksi etsi-lomake. Elementti ”section” on tarkoitettu sisältämään eri toiminnallisuus- tai aihealueita, ja tämän sukulaiselementti ”article” on tarkoitettu sisältämään yksittäisiä, sivuun liittyviä, sisältöjä. Elementin ”aside” tulisi sisältää sen ympärillä olevaan liittyvää sisältöä. Lopuksi ”footer”-elementin tulisi määritellä dokumentin tai osion alatunnisteen, ja se tyypillisesti sisältää tiedot dokumentin kirjoittajasta, yhteystiedot jne. (W3C 2014).

HTML5:den graafisia elementtejä ovat ”canvas”- ja ”svg”-elementit, jotka mahdollistavat grafiikan piirtämisen HTML-dokumenttiin (W3Schools n.d.). Näistä kahdesta työssä käytettiin ”canvas”-elementtiä. Tämä elementti toimii vain säiliönä grafiikalle, itse grafiikka täytyy piirtää skriptin, kuten JavaScriptin, avulla. (W3Schools n.d.).

2.2 CSS-tyyliohje

CSS on merkintäjärjestelmä, jolla voi esittää selaimille dokumenttien ulkoasua koskevia ehdotuksia. Yhtä ehdotusten kokonaisuutta sanotaan **tyyliohjeeksi** eli tyylisäännöstöksi, englanniksi *style sheet.* (Korpela, 2008, 2.)

CSS määrittää web-sivun tyylin ja elementtien ulkoasun, esimerkiksi sillä voidaan määrittää käytetty fontti, fontin väri ja taustan väri. Kuten HTML, CSS ei ole oikea ohjelmointikieli. Se ei ole myöskään merkintäkieli vaan tyyliohjekieli. Sen avulla voi HTML-dokumentin elementteihin osoittaa haluttuja tyylejä selektiivisesti. (Mozilla Developer Network 2017)

KUVIO 5. CSS-ohjeistus, jossa kaikkien ”p”-elementtien fontin väriksi määritellään punainen (Mozilla Developer Network n.d., muokattu.)

HTML-dokumentissa voidaan ottaa käyttöön CSS-ohjeistus kahdella eri tavalla: tyyliohje voi olla kokonaan erillinen tiedosto tai se voidaan kirjoittaa suoraan HTML-dokumenttiin, joko ”style”-tagien sisälle tai suoraan elementtiin ”style”-attribuutin avulla. Kuvion 5 osoittamaa kirjoitustapaa voidaan käyttää, kun CSS-ohjeistus sijaitsee joko erillisessä tiedostossa tai ”style”-tagien sisällä. Suoraan elementtiin kirjoitettuna syntaksi on hieman erilainen mutta sillä voidaan määrittää suoraan tietyssä elementissä käytetty tyyli.

Työssä sovelluskehyksen tyyliohje sijaitsi niin erillisessä tiedostossa kuin HMTL-dokumentissakin. Erillisessä tiedostossa määriteltiin sovelluksen runkoelementtien ominaisuudet niille osoitettujen attribuuttien ja luokkien avulla sekä joitain valmiita tyyliluokkia, joita tultaisiin myöhemmin sovelluksen ajonaikana HTML-elementeille lisäämään. Lisäksi HTML-dokumentin ”head”-elementtiin injektoitiin sovelluskehyksen suorituksen alussa JSON-tiedoston määreiden perusteella lisää tyylillisiä ominaisuuksia sovelluskehyksen näkymälle.

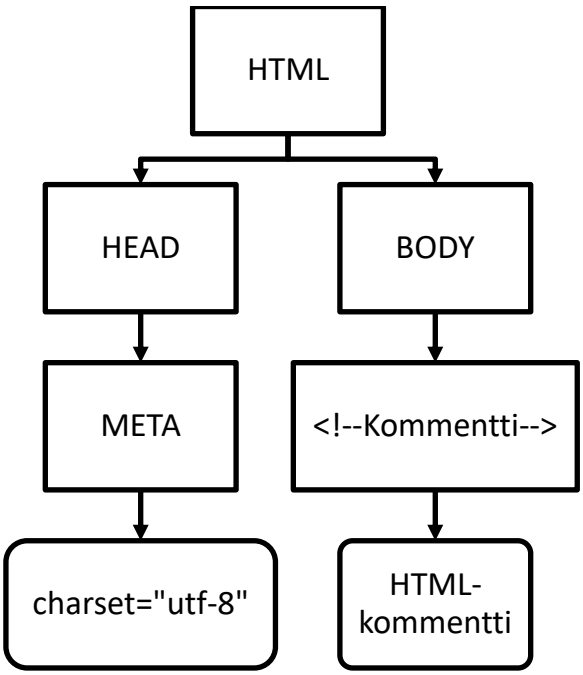
2.3 Sovelletut ohjelmointitekniikat

Tässä luvussa käydään läpi sovelluskehyksessä sovelletut ohjelmointitekniikat. Luvussa 2.3.1 tutustutaan tarkemmin DOM-rajapintaan ja JavaScriptiin, luvussa 2.3.2 JavaScript-kirjastoon jQuery ja lopuksi luvussa 2.3.3 JSON-merkintäkieleen. Lisäksi luvussa 2.3.4 mainitaan kehitystyön tärkeimmän työkalut.

2.3.1 DOM-rajapinta ja JavaScript-ohjelmointikieli

DOM (Document Object Model) on HTML- ja XML-dokumenttien ohjelmointirajapinta. Se tarjoaa rakenteellista tietoa dokumenteista ja määrittelee tavan, jolla tähän rakenteeseen päästään ohjelmasta käsin. Tällöin dokumentti esitetään rakenteellisena **solmuryhmänä** (group of nodes), joka liittää web-dokumentin skriptiin tai ohjelmointikieleen. (Peltomäki & Nykänen 2006, 168).

DOM perustuu oliorakenteeseen, joka muistuttaa läheisesti mallinnettavan dokumentin rakennetta. Sen nimi ”Document Object Model” valittiin, koska se on ”oliomalli” perinteisen olio-ohjelmointityylin mukaan: dokumentit mallinnetaan käyttämällä olioita ja malli käsittää dokumentin rakenteen lisäksi myös dokumentin käyttäytymisen ja oliot, joista se on koostettu. (W3C 2000). DOM sai alkunsa määrittelynä, joka sallisi JavaScript-skriptien ja Java-ohjelmien olla sisällytettyjä web-selaimissa (W3C 2000). Useimmat nykyaikaiset selaimet tukevat W3C DOM- ja WHATWG DOM -standardeja mutta kaikki selaimet käyttävät jotain dokumenttioliomallia tehdäkseen verkkosivuista JavaScriptillä muokattavia (Mozilla Developer Network 2018.)



KAAVIO 1. Kuvassa 1 esitetyn HTML-dokumentin DOM-malli.

Kaaviossa 1 on esimerkki DOM:n muodostamasta rakenteesta. Tämä esittää kuvion 2 HTML-dokumentin DOM:n graafisen esityksen, jossa DOM on esitettynä puurakenteena. Puurakenteen solmut eivät kuvaa tietorakennetta, vaan olioita, joilla on funktioita ja identiteetti. Rakenteen juurena on ”HTML”-tagia kuvaava solmu. Juuresta haarautuvat sen sisäiset tagit ”HEAD” ja ”BODY” omiksi solmuikseen, toisin sanoen ”HTML”-solmun lapsisolmuiksi. Näillä on vielä kummallakin omat lapsisolmunsa, ”HEAD”-tagin sisällä oleva ”META”-tagi ja ”BODY”-tagin sisällä oleva kommentti. ”HEAD”- ja ”BODY”-solmut ovat lapsisolmujensa vanhempia. ”BODY”-tagin sisällä oleva kommentti luetaan mukaan DOM:in omaksi solmuksi, ja tälle haarautuu oma tekstisolmu kommentin tekstisisältöä kuvaamaan. Tekstisisältö on tavallaan kommenttiolion ominaisuus. Samalla tavalla myös ”META”-solmun lapsisolmu, joka kertoo dokumentin merkistökoodauksen (”charset”-attribuutti), on ”META”-solmuolion ominaisuus.

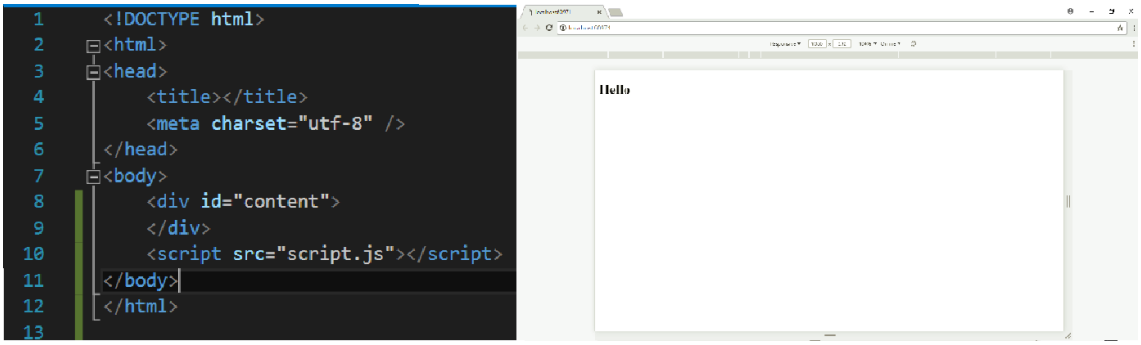
Sovelluskehyksen DOM-mallia manipuloitiin JavaScriptillä. JavaScript on järjestelmäriippumaton, oliokeskeinen skriptikieli. Se on pieni ja kevyt ohjelmointikieli. Ajoympäristössä (esimerkiksi selaimessa) JavaScript voidaan yhdistää sen ympäristön olioihin ja osiin tuottaen niihin ohjelmallista hallintaa (Mozilla Developer Network n.d.). Esimerkiksi kuviossa 6 luodaan erilliselle HTML-dokumentille tekstisisältöä manipuloiden HTML-dokumentin DOM-olioita JavaScriptillä.

```
1  var div = document.getElementById("content");
2
3  var header = document.createElement("H1");
4  var text = document.createTextNode("Hello");
5  header.appendChild(text);
6  div.appendChild(header);
7
```

KUVIO 6. JavaScript-koodi, jossa manipuloidaan HTML-dokumentin DOM:ia.

Kuviossa 6 on skripti-tiedosto script.js. Sen rivillä 1 tallennetaan JavaScript-muuttujaan ”div” HTML-elementtiolio käyttämällä DOM:n ”document”-rajapinnan metodia ”getElementById”. Metodilla noudetaan HTML-dokumentista luodusta DOM-mallista solmu, jonka id-attribuutti on ”content”. Rivillä 3 luodaan HTML-elementti ”h1” ja rivillä 4 puolestaan luodaan tekstientiteetti sisällöltään ”Hello” edelleen ”document”-rajapinnan metodien ”createElement” ja ”createTextNode” avulla. Edellä luotuun ”h1”-elementtiin

määritetään sisällöksi tekstientiteetti liittämällä se tähän ”appendChild”-funktiolla, joka lisää tekstientiteetin ”h1”-elementin lapsisolmuksi. Lopuksi luotu ”h1”-elementti lisätään ”div”-muuttujan lapsisolmuksi.



KUVIO 7. Vasemmalla yksinkertainen HTML-dokumentti, oikealla sama dokumentti selaimessa ajettuna.

Kuviossa 7 on esitettynä vasemmalla HTML-sivuston dokumentti, jota manipuloitiin kuvion 6 script.js-koodissa. HTML-dokumenttiin on sisällytetty rivillä 10 kuvion 6 esimerkin script.js -tiedosto ulkoisena lähteenä ”script”-tagien avulla asettamalla niille ”src”-attribuutiksi skriptin sijainti. Kuvion 7 oikealla puolella on vasemmalla esitetty HTML-dokumentti ja siihen scriptissä luodut elementit selaimessa suoritettuna. Tästä nähdään, että script.js-skripti on lisännyt web-sivulle tekstin ”Hello”, mitä ei aluperäisessä HTML-dokumentissa ollut.

Toisin kuin useissa ohjelmointikielissä, JavaScript ei sisällä syöte- tai tulostustoimintoja. Se on suunniteltu ajettavaksi skriptikielenä jossain ajoympäristössä tai palvelimella, ja tämän ympäristön vastuulla on tarjota mekanismit kommunikointiin ajoympäristön ulkopuolelle. Yleisin ajoympäristö on selain mutta JavaScript-tulkkeja voi löytää useista muistakin paikoista. (Mozilla Developer Network 2018).

```
function Terve(nimi) {
    if (nimi.length > 0) {
        alert("Moikka " + nimi)
    }
}

Terve("Saara");
```

KUVIO 8. JavaScript -funktio.

Kuviossa 8 on esimerkki JavaScript-funktiosta nimeltä ”Terve”. Funktiolle välitetään parametri ”nimi”. Jos annetun parametrin pituus merkkeinä on suurempi kuin 0, selaimeen avautuu ponnahdusikkuna, missä tervehditään parametrina välitettyä nimeä. Kuvan alareunassa kutsutaan funktiota.

Selaimessa skriptiä voidaan ajaa joko HTML-sivulla ”script”-tagien välissä tai määrittämällä ”script”-tagiin lähde erilliseen JavaScript-tiedostoon. Kuviossa 6 on JavaScript-tiedosto liitetty HTML-dokumenttiin erillisenä tiedostona.

JavaScriptillä koodattiin sovelluksen toiminnallisuus, eli ajonaikaiset tapahtumien käsittelyt ja DOM-olioiden manipulaatio. Manipulaatiolla tarkoitetaan muun muassa DOM-olioiden ja niiden attribuuttien lisäämistä ja poistamista sekä joissain tapauksissa muokkaamista. Sovellus ohjelmoitiin proseduraalisesti, eli ohjelman suoritus jäsennettiin pienempiin, itsenäisiin kokonaisuuksiin, aliohjelmiin (Wikipedia 2017.)

2.3.2 jQuery-kirjasto

jQuery on nopea, pieni ja monipuolinen JavaScript-kirjasto. Se tekee useasta asiasta, kuten HTML-dokumentin manipuloinnista, tapahtumien käsittelystä ja animoinnista yksinkertaisempaa helppokäyttöisellä ohjelmointirajapinnalla, joka toimii useissa selaimissa (jQuery n.d.) Periaatteessa jQuery pakkaa JavaScriptin yksinkertaisempaan muotoon.

jQueryn syntaksi on periaatteessa seuraava: $(selektori).toiminto(), missä

- $-merkillä määritellään/otetaan käyttöön jQuery
- selektorilla haetaan HTML-elementti
- jQuery toiminto() -komennolla määritellään selektorilla valittuun elementtiin kohdistuva funktionaalisuus.

(W3Schools n.d.).

```
$("#content").append("<h1>Hello</h1>");
```

KUVIO 9. jQuery esimerkki.

Kuviossa 9 on samanlainen HTML-elementin luominen ja sivulle lisääminen kuin JavaScript-esimerkissä (**Virhe. Viitteen lähdettä ei löytynyt.**). Selektorin ”content” edessä olevalla #-merkillä määritellään, että hakusanana on HTML-elementin id-attribuutti.

Sovelluksen toisessa versiossa käytettiin jQueryä JavaScriptin sijasta sen yksinkertaisemman syntaksin vuoksi. Skriptin toiminnallisuudet pysyivät teknologian vaihdosta ja koodin uudelleen kirjoittamisesta huolimatta kutakuinkin samoina. Käytetty versio jQuerystä oli 3.2.1.

2.3.3 JSON-syntaksi

JSON on kevyt, tekstipohjainen ja kieliriippumaton syntaksi tiedonsiirtoformaatin määrittelyyn. Se johdettiin ECMAScript -ohjelmointikielestä, mutta on ohjelmointikielestä riippumaton. JSON määrittelee pienen erän rakenteellisia sääntöjä jäsennellyn datan esittämiseen. (Ecma International 2017). JSON on lyhenne sanoista JavaScript Object Notation, jonka perusteella voidaan sanoa sen olevan JavaScriptiä kirjoitettuna olionotaatiolla.

JSON on rakennettu kahden universaalin rakenteen pohjalle: nimi/arvo -parien eli olioiden ja jäsenneltyjen arvolistojen eli listojen varaan. JSON:ssa olio on järjestämätön määrä nimi/arvo -pareja, ja se määritellään aaltosulkeiden ({}) sisälle. Jokaista nimeä seuraa kaksoispiste ja nimi/arvo -parit eritellään pilkulla. Listan sisältö määritellään hakasulkeiden ([]) väliin, ja sen sisältämät arvot erotetaan toisistaan pilkulla. (Json.org n.d.)

```
1  {
2    "Objekti": {
3      "attr1": "tekstiattr",
4      "attr2": 123,
5      "attr3": "\"sitaatit\""
6    },
7    "Lista": [
8      "eka", 2
9    ]
10 }
```

KUVIO 10. Yksinkertainen JSON-tiedosto.

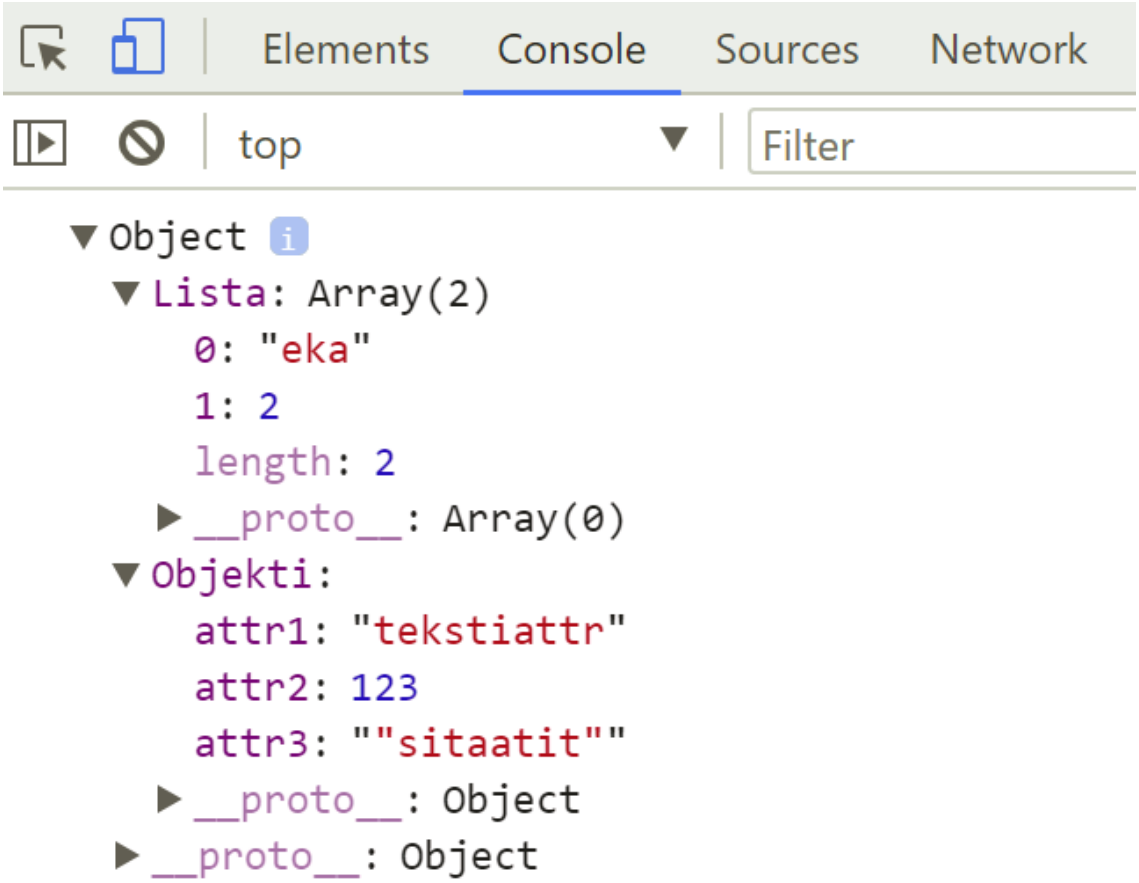
Kuviossa 10 on esitetty esimerkiksi JSON-tiedoston sisältöesimerkki, jossa on olio ”Objekti” ja lista ”Lista”. ”Objekti”-oliolle on määritetty kolme nimi/arvoparia, joista rivin 3 nimellä ”attr1” on tekstiarvo ”tekstiattr” ja rivin 4 nimellä ”attr2” on numeroarvo 123. Rivin 5 nimellä ”attr3” on tekstiarvo, jonka ympärille on sisällytetty sitaattimerkit \”-merkinnällä. Tällä \”-merkinnällä voidaan sisällyttää Unicode-merkkejä arvoihin. (Json.org n.d.). Rivillä 8 on määritelty listalle ”Lista” kaksi arvoa, teksti ”eka” ja numero 2.

```
1  function readJSON(filename, callback) {
2      var req = new XMLHttpRequest();
3      req.overrideMimeType("application/json");
4      req.open("GET", filename, true);
5      req.onreadystatechange = function () {
6          if (req.readyState === 4 && req.status == "200") {
7              callback(req.responseText);
8          }
9      }
10     req.send(null);
11 }
12
13 function parseJSON(data) {
14     console.log(JSON.parse(data));
15 }
16
17 readJSON("json.json", parseJSON);
18
```

KUVIO 11. JSON-tiedoston käsittely JavaScriptillä.

JSON-tiedosto luetaan työssä skriptitiedostossa kuvion 11 riveillä 1-12 esitellyn funktion ”readJSON” mukaisesti. Funktiolle välitetään parametreinä luettavan tiedoston ja callback-funktion nimet, joista callback-funktiota kutsutaan onnistuneen lukemisen jälkeen. Kuviossa kutsutaan readJSON-funktiota rivillä 17, ja sillä on parametrina teksti ”json.json” ja funktionimi ”parseJSON”. Tiedoston lukeminen suoritetaan luomalla uusi XMLHttpRequest-olio (rivi 2), joka mahdollistaa HTTP-protokollan käytön. Rivillä 3 määritellään luettavan tiedoston MIME-tyypiksi JSON ja rivillä 4 aloitetaan HTTP-protokollan GET-pyyntö luettavaan, paikalliseen tiedostoon nimeltä ”json.json”. Rivillä 5 määritellään funktio, joka suoritetaan, kun XMLHttpRequestin ”readyState”-tila muuttuu. Attribuutti ”readyState” kertoo XMLHttpRequestin client:n tilan eli missä vaiheessa tiedoston lukemista ollaan. Lisäksi halutaan tietää XMLHttpRequestin tila (status-attribuutti). Mikäli pyyntö on suoritettu onnistuneesti eli status on 200 (W3 n.d.) ja mikäli client:n tila on valmis eli 4 (XMLHttpRequest Standard 2018), kutsutaan välitettyä callback-funktiota, jolle annetaan parametrinä luetun tiedoston paluuviesti ”responseText”.

Callback-funktio ”parseJSON” on määritelty riveillä 13-15. Funktiossa ”JSON.parse”-metodi parsii JSON-merkkijonon ja muodostaa siitä JavaScript arvon tai olion JSON-merkkijonon määrityksestä riippuen (Mozilla Developer Network 2018).



KUVIO 12. Parsitun JSON-tiedoston sisältö selaimen konsoliin tulostettuna.

Kuviossa 12 on ruudunsieppaus selaimen konsolitulosteesta sen jälkeen, kun kuviossa 10 suoritettu JSON-tiedoston lukeminen on onnistuneesti suoritettu. Konsolitulosteen suorittava funktio on ”console.log”, jota kutsutaan kuviossa 11 rivillä 14. Kuten nähdään, selain tulkitsee JSON:n olioksi, joka sisältää listan ”Lista” ja olion ”Objekti”.

JSON:a käytettiin työssä resurssitietojen lähteenä. Sillä kirjoitettiin tiedosto, joka sisälsi tiedot näkymään halutusta ohjelmasta ja ohjelmalle halutut tyylilliset määreet, kuten värit eri elementeille.

2.4 Kehitystyökalut

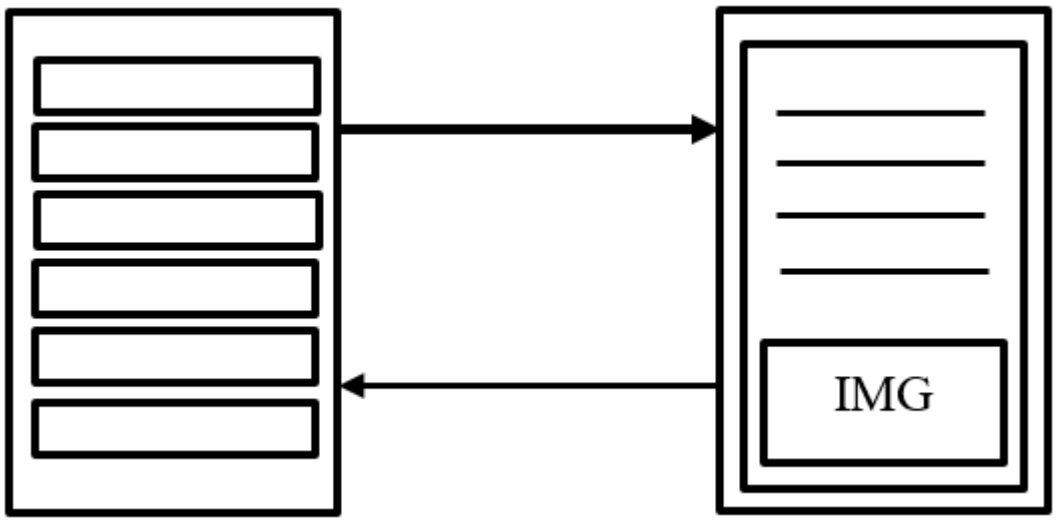
Kehitys tehtiin käyttämällä Microsoftin Visual Studio 2017 -ohjelman ilmaisversiota. Sovelluskehystä testattiin tämän ohjelman tarjoaman käännöstyökalun avulla. Sovelluskehys käännettiin Google Chrome -selaimelle, jonka DevTools-työkalu toimivat yhtenä kehitystyökaluna. Lisäksi kehitystä tehtiin suorittamalla sovelluskehystä yhdellä näyttölaitteella. Näyttölaitteella suoritettiin sovellukselle myös erinäisiä testitapauksia, joilla pyrittiin simuloimaan sovelluksen reaalimaailman käyttötilanteita.

3 KEHITYSPROSESSI

Tämä luku kuvaa kehityksen etenemistä toimeksiannosta lähtien aina toiseen versioon asti. Seuraavissa luvuissa esitetään kehitykseen liittyneet ja vaikuttaneet prosessit ja asiat.

3.1 Toimeksianto

Alkuperäisenä toimeksiantona toimi Jidokan asiakkaalle myyty palvelu, jolla näytetiin näyttötornilla messuohjelma. Tällaisia messuohjelmia oli tehty asiakkaalle jo aikaisemmin mutta ne ovat olleet yksittäisiä digitaalisia julisteita, jossa on näkynyt staattisesti koko tapahtuman ohjelma. Tästä mallista pyrittiin kehittämään dynaamisempi versio, joka näyttäisi koko tapahtuman ohjelman lisäksi erillisellä näkymällä sen hetkisen aktiivisen ohjelmanumeron sekä lisätietoa tästä ohjelmasta.



KUVIO 13. Sovelluskehyksen luonnos.

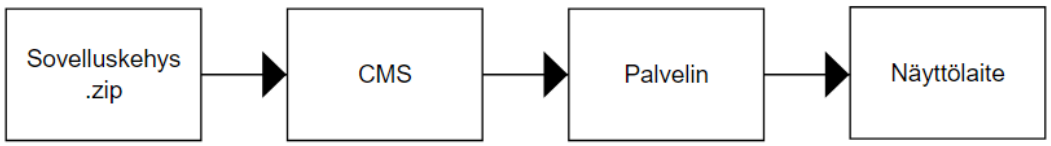
Kuviossa 13 on esitetty sovelluskehyksen luonnos. Alkuun sovelluskehykseltä odotettiin kahta erillistä näkymää, jotka kuviossa on piirretty kahdeksi isoksi suorakulmioksi. Nuolet suorakulmioiden välillä osoittavat näkymien vaihtumista: näkymästä siirrytään toiseen ja takaisin, kun näkymä on ollut ”aktiivinen” eli näkynyt näytöllä tietyn ajan. Vasemmanpuoleinen suorakulmio, jonka sisällä on pienempiä suorakulmioita, kuvaa koko ohjelmiston sisältämää näkymää, jossa pienemmät suorakulmiot esittävät yksittäisiä ohjelmanu-

meroita. Oikeanpuoleinen iso suorakulmio kuvaa sen hetkistä ohjelmanumeroa korostavaa näkymää, joka sisältäisi lisätietoa ohjelmasta (useat viivat kuvaavat leipätekstiä) ja mahdollisesti kuvan (IMG-kuvio).

Sovelluskehys tulisi ajettavaksi Jidokan 50-tuumaisella näyttötornilla (KUVA 1). Näyttötornin LCD-näytön resoluutio on 1080 x 1920 pikseliä, ja tästä saatiin dimensiot, jotka sovelluskehyksen tulisi täyttää.

KUVA 1. Jidokan 50" Näyttötorni.

Jidokan näytöille asetetaan sisältö sen oman sisällönhallintajärjestelmän avulla. Näyttölaitteessa oleva tietokoneratkaisu lataa sisällönhallintajärjestelmän palvelimelta omalle paikalliselle tietokoneelleen sille osoitetun sisällön, ja suorittaa sen sille annettujen määreiden mukaisesti. Sovelluskehys ladataan .zip-pakettina sisällönhallintajärjestelmän kautta palvelimelle.



KUVIO 14. Sovelluskehyksen lisääminen näyttölaitteelle.

Näyttölaitteen tietokoneena toimii Raspberry PI, jossa ajetaan Rasbpian käyttöjärjestelmää. Raspbian on Debian-pohjainen käyttöjärjestelmä Raspberry PI:lle (Wikipedia 2018.) Rasbpian suorittaa X-ikkunointijärjestelmää, joka mahdollistaa yksinkertaisen sovelluskehyksen graafiselle käyttöliittymäympäristölle (Wikipedia 2018.) X-ikkunointijärjestelmässä vuorostaan ajetaan Chromium-selainta, joka toimii HTML-sovelluskehyksen ajoympäristönä.

3.2 Kehitysprosessista

Kehitystä lähdettiin tekemään kuvion 13 esittämän rakenteen, teknologisten ja laitteiston asettamien reunaehtojen perusteella. Teknologiset reunaehdot olivat seuraavat: tuote tulisi toteuttaa HTML5 -tekniikoita käyttäen. Laitteisto asetti sovelluskehykselle vaatimukseksi, että sen tulisi toimia Google Chrome-selaimella, sillä lopullisella laitteella se tulisi ajetuksi Chromium-selaimessa. Tämä siksi, että Chromium ja Chrome selaimet koostuvat suurimmaksi osaksi samasta lähdekoodista (Wikipedia 2018.)

Lisäksi oli tarkoitus kehittää palvelu, joka olisi helposti ja vähin muokkauksin käytettävissä myös muiden asiakkaiden kohdalla. Tämän perusteella päätettiin sovelluskehyksen sisältö lukea erillisestä tiedostosta sen sijaan, että se olisi kovakoodattu näkymään eli HTML-tiedostoon suoraan. Tähän sopi JSON-formaatilla kirjoitettu tiedosto, sen kehittäminen voitiin aloittaa nopeasti.

3.2.1 Asiakkaan rooli kehitystyössä

Asiakas otettiin mahdollisimman paljon mukaan kehitystyöhön, ja tällä pyrittiin asiakaslähtöiseen kehitykseen. Asiakas toimitti haluamansa ohjelmatiedot sekä yleensä ilmoitti tapahtuman värit tai muut teemaan liittyvät muotoiluohjeet. Tapauksissa, joissa asiakkaalta saadut muotoiluun liittyvät yksityiskohdat olivat niukkoja, yritettiin löytää muualta, kuten tapahtumien kotisivuilta, inspiraatiota ohjelmanäkymien muotoiluun. Ensimmäisen asiakkaan kohdalla löydettiinkin tällaista apua; muun muassa kotisivuilta löydettyä fonttia käytettiin myös ohjelmanäkymässä. Tällä pyrittiin varmistamaan asiakkaan tyytyväisyys tuotteeseen. Kun näiden tietojen perusteella oli saatu ensimmäinen versio valmiiksi, lähetettiin tästä linkki asiakkaan hyväksyttäväksi ja arvioitavaksi. Asiakasta

kannustettiin kertomaan kehitysehdotuksia ja antamaan palautetta. Muutoksia sovelluskehykseen tehtiin saadun palautteen perusteella, mikäli ne oli mahdollista toteuttaa ajan puitteissa.

3.2.2 Muut kehitykseen vaikuttaneet tekijät

Vaikka kehitystyötä tehtiin yhden kehittäjän voimin, kysyttiin muilta yrityksen työntekijöiltä palautetta ja kehitysehdotuksia oikealla näyttölaitteella testaamisen yhteydessä. Tällä pyrittiin löytämään ongelma- ja kehityskohtia, jotka kehittäjältä olivat jääneet huomaamatta. Tämä osoittautui hedelmälliseksi sovelluksen ominaisuuksien kehittämisen suhteen, ja jatkokehitystä varten saatiin muutamia kehitysideoita ja lisättäviä ominaisuuksia.

3.2.3 Testaus

Sovelluskehystä testattiin kahdella erilaisella metodilla. Ensinnäkin testausta suoritettiin ns. passiivisesti ennalta määrättyjen testitapausten perusteella. Yksi tällainen testitapaus kesti kahden päivän ajan, ja sillä haluttiin selvittää, osaako sovelluskehys reagoida muuttuvaan ohjelmasisältöön. Sovelluskehykseen ladattiin testin vaatima sisältö, tässä tapauksessa kahden päivän ohjelma, missä vain toisella päivällä oli ohjelmaa. Testeille merkittiin ennen testausta odotetut tulokset päivien kohdalle. Suoritetuista testeistä ilmoitettiin testiaika ja täyttyivätkö odotetut tulokset sekä vielä lopputulema eli oliko näyttölaite suoriutunut testistä.

Testausta suoritettiin myös aktiivisemmin ja aggressiivisemmin. Tällaista testausta oli näyttölaitteen tietokoneen kellon ja päivämäärän manipulointi, millä pyrittiin varmistamaan sovelluskehyksen toiminnallisuuden eheys eli reagointi ajan muutoksiin.

3.3 Ensimmäinen versio

Ensimmäinen versio kehitettiin Easyfairsin käyttöön. Sitä hyödyntäviä tapahtumia oli yhteensä neljä, joiden välillä kehitystyötä jatkettiin ilmenneiden ongelmien ja asiakkaan tapahtumakohtaisten toiveiden mukaan. Ensimmäisessä tapahtumassa käytettyyn versioon oli käytettävissä yksi kuukausi, mikä tarkoitti sitä, että toimiva ja asiakkaan tarpeisiin

sopiva versio pitäisi saada tässä ajassa valmiiksi. Tämä alkuperäisen kehitysajan niukkuus rajasi käytettävän JavaScript-kirjaston valinnan Vanilla JavaScriptiin, koska kehittäjällä oli jo jonkin verran kokemusta tästä ohjelmointikielestä, ja kehitys sillä voitiin aloittaa heti.

3.3.1 Visuaalinen suunnittelu ja animointi

Näkymien elementtien muotoilua ja sisältöä lähdettiin hahmottamaan asiakkaan tarjoamien tietojen perusteella sekä aiempien messuohjelmien pohjalta. Ensimmäisen asiakkaan kohdalla tiedot koostuivat yleensä satunnaisesta määrästä ohjelmanumeroita, joilla useimmilla oli vain yksi esiintyjä. Esiintyjästä oli yleensä kerrottu nimi, ammatti, työnantaja ja hänestä saattoi olla kuva. Lisäksi puheenvuorolla oli jokin otsikko sekä alkamis- ja loppumisaika. Joillain saattoi olla myös puheenvuoroa selventävää kuvausta mutta tällaisia oli Easyfairsin tapahtumissa vähän. Paneelikeskustelut olivat muista poikkeava ohjelmanumero, missä oli useampi henkilö osallistujana.

Aikaisempien messuohjelmien mukaan tehtiin koko päivän ohjelman näyttämästä näkymästä kaikki ohjelmanumerot listaava näkymä, jossa kustakin puheenvuorosta kerrottiin alkamis- ja loppumisaika, puheenvuoron nimi ja esiintyjä. Tarkoitus oli näyttää erikseen aktiivisen puheenvuoron näyttämällä sivulla jo edellä mainittujen tietojen lisäksi tiedot puhujan työantajasta ja nimikkeestä sekä puhujan kuva. Näin pyrittiin pitämään kunkin näkymän tarjoama tieto suhteellisen pienenä. Asiakkaan toiveesta päädyttiin kuitenkin lisäämään ohjelmanäkymäänkin tiedot puhujan työnantajasta ja nimikkeestä, koska asiakas koki näiden olevan katsojia kiinnostava tieto, mikä pitäisi olla näkyvillä mahdollisimman paljon.

Sovelluskehyksessä haluttiin hyödyntää mahdollisuutta animoida HTML-elementtejä. CSS mahdollistaa HTML-elementtien siirtymien animoinnin. Animaatiot koostuvat kahdesta komponentista: CSS-animaatiota kuvaavasta tyyliluokasta ja joukosta ”avainkehyksiä”, jotka ilmoittavat animaation tyylin alku- ja lopputilan, sekä mahdolliset tilat tältä väliltä (Mozilla Developer Network 2018). Tyyliluokalle voidaan antaa useampia attribuutteja riippuen siitä, miten animaation halutaan käyttäytyvän. Nämä attribuutit ovat nimi, kesto, viive, toistokerrat, toistotapa, nopeuskäyrä ja täyttötapa (W3Schools n.d.).

```
.show-item {
  animation: show 1s ease;
}

@keyframes show {
  0% {
    visibility: hidden;
    opacity: 0;
  }

  1% {
    visibility: visible;
    opacity: 0;
  }

  100% {
    visibility: visible;
    opacity: 1;
  }
}
```

KUVIO 15. CSS-tiedostossa määritelty animaatio, joka näyttää elementin vähitellen.

Kuviossa 15 on esitettynä ensimmäisen version elementin näkymään tuovan animaation toteuttava luokka ”show-item”. Luokalle määritellään animaatio ”animation”-attribuutilla, ja tälle attribuutille voidaan antaa arvoja riippuen siitä, miten animaation halutaan toistuvan. Kuvion esimerkissä arvoina ovat animaation nimi (show), sen kesto (1 sekunti) ja nopeuskäyrä, jolla määritetään animaation nopeus toiston eri vaiheissa (ease). Animaation ”avainkehykset” määritellään ”@keyframes”-säännöllä, ja tämän jälkeen ilmoitetaan määritettävän animaation nimi. Kuviossa animaatiolle ”show” on määritelty kolme avainkehystä riippuen siitä, missä vaiheessa suoritusta ollaan prosentuaalisesti. Kuviossa asetetaan 0 %:ssa eli animaation alussa elementti olemattomaksi määräämällä ”visible”-attribuurille arvo ”hidden”. Heti tämän jälkeen 1 %:n kohdalla tuodaan elementti läpinäkyväksi asettamalla edelliselle arvolle arvoksi ”visible” mutta jättämällä läpikuultamattomuus eli ”opacity”-arvo nolliin. 100 %:n kohdalla eli animaation lopputilassa asetetaan elementti täysin näkyväksi määräämällä läpikuultamattomuus maksimiinsa eli yhteen.

```
document.getElementById(name).classList.add("show-item");
setTimeout(function () {
    document.getElementById(name).style.visibility = "visible";
    document.getElementById(name).classList.remove("show-item");
}, 1000);
```

KUVIO 16. CSS-animaation suorittaminen JavaScriptissä.

Kuviossa 16 on esitettynä edellä määritellyn CSS-animaation käyttö JavaScriptissä. Ensinnäkin animoitavaan DOM-solmuun, jonka id on kuviossa tallennettu muuttujaan ”name”, lisätään kuviossa 14 määritelty tyyliluokka ”show-item”. Tyyliluokan lisääminen solmuun käynnistää selaimessa animaation. Koska animaation kestoksi oli määrätty CSS-tiedostossa yksi sekunti, piti edellä lisätty tyyliluokka poistaa animoimiseen kuluneen ajan jälkeen. Poistaminen täytyy tehdä, ettei sama animaatio toistuisi uudestaan. Tässä hyödynnettiin JavaScriptin ”setTimeout”-funktiota, jolla voidaan määrittää haluttu koodi suoritettavaksi tietyn millisekunteina annetun ajan kuluttua. Ajaksi annettiin siis 1000 millisekuntia eli 1 sekunti. Ajan kulumisen jälkeen suoritetaan koodi, jossa animoitu solmu asetetaan näkyväksi manipuloimalla DOM-solmun tyylin näkyvyys-attribuuttia ja tyyliluokka ”show-item” poistetaan saman DOM-solmun tyyliluokkalistasta.

Animointi onnistuttiin toteuttamaan yksittäisten ohjelmanumeroiden kohdalla koko ohjelmanäkymässä. Yritettäessä animoida yksittäisen ohjelmanumeron näyttämää näkymää samalla tavalla huomattiin, ettei näyttölaitteen tietokoneen suorituskyky riittänyt niin suuren alueen animoinnin sulavaan suorittamiseen. Tästä johtuen jätettiin ensimmäisessä versiossa yksittäisen ohjelmanumeron animoiminen kokonaan pois, ja vain koko ohjelman näyttämän näkymän ohjelmanumerolista animoitiin.

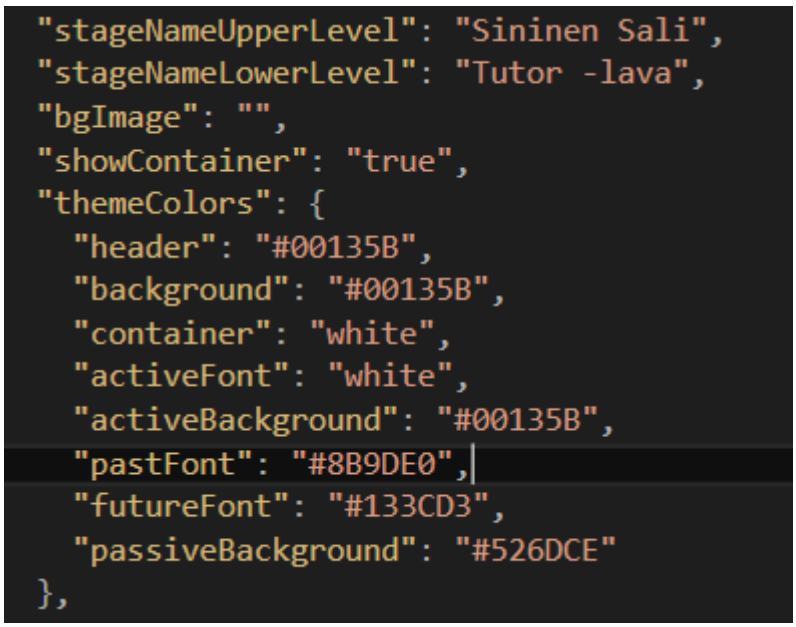
3.3.2 Käytettävyyden huomioiminen

Vaikka työhön ei tullut interaktiivista käyttöliittymää kehityksen tässä vaiheessa, pyrittiin hyvään käytettävyyteen ja luettavuuteen suunniteltaessa ja kehittäessä visuaalista ilmettä tuotteelle. Hyvän käytettävyyden saavuttamiseksi perehdyttiin muun muassa alan asiantuntijoiden kirjallisuuteen. Yksi näistä oli Jakob Nielsen, joka on keksinyt useita käytettävyyden arvioimiseen tarkoitettuja metodeja (Nielsen Norman Group, 2018). Nielsenin 10 heuristiikkaa käytettävyydelle (Nielsen Norman Group, 1995) olivat eräs hyödynnetyistä metodeista.

3.3.3 Värit näkymässä

Asiakas voi määrittää tietyille elementeille haluamansa värit. Värien määrää pyrittiin kuitenkin rajoittamaan, sillä Nielsenin (1993, 119) mukaan on hyvä rajoittaa värien määrää käyttöliittymässä korkeintaan 5-7 eri väriin. Mikäli asiakkaalta oli tullut tapahtumaan liittyen teemaväri, määritettiin sovellukseen monokromaattinen väriteema annetun värin perusteella. Monokromaattisessa väriteemassa annetusta väristä johdettaan useita eri sävyjä, muuttamalla vain sen valoisuutta ja saturaatiota. (W3Schools n.d.).

Eri väreillä pyrittiin tuomaan esille elementtien ominaisuuksia. Esimerkiksi ohjelmanäkymään ohjelmanumeron tilan mukaan pystyi asettamaan tälle määrittelytiedostossa oman värikoodinsa. Näin aktiiviselle eli käynnissä olevan ohjelmanumeron elementille oli mahdollisuus määrittää eri värit, niin fontille kuin taustalle, kuten myös passiiviselle elementillekin. Passiivisia elementtejä olivat menneet ja tulevat ohjelmanumerot sisältävät elementit. Näille olisi ollut mahdollista määrittää eri taustavärit. Samaa taustaväriä päädyttiin käyttämään siksi, että käytettyjen värien määrä pysyisi suhteellisen matalana.



KUVIO 17. Osa ensimmäisessä versiossa käytetystä JSON-tiedostosta.

Kuviossa 17 näkyy ”themeColors”-olion attribuutteina elementit, joille pystyi määrittämään värit ensimmäisessä versiossa. Näitä olivat tausta (background), säiliö-elementti (container), aktiivinen ja passiivinen tausta (active- ja passiveBackground). Lisäksi voidaan määrittää kolme eri fonttiväriä, jos halutaan korostaa ohjelmien tilaa. Ohjelmanu-

meroiden fonttien värit määritellään attribuuteilla ”activeFont”, ”pastFont” ja ”futureFont”. Attribuutti ”activeFont” määrää arvollaan aktiivisen ohjelmanumeron fontin värin, ”pastFont”-attribuutti määrittää menneen tapahtuman fontin värin ja ”futureFont”-attribuutti määrittää tulevan tapahtuman fontin värin. Attribuutilla ”header” puolestaan määritetään tapahtuman otsikon väri. Näin ollen käyttöliittymän väreiksi voi tulla 8 eri väriä, joka on edellä mainitusta reunaehdosta suurempi. Käytännössä jotkin näistä elementeistä kuitenkin olivat samanvärisiä, joten reunaehtoa ei suoranaisesti rikottu.

3.3.4 Sovelluskehyksen toiminnallisuus

Sovellusta suoritettaessa ladataan sen esittämä verkkosivusto eli sovelluskehyksen HTML- ja CSS -osa selaimeen, ja liitetyn skriptin lukeminen aloitetaan näiden kanssa yhtäaikaisesti. Kun HTML-dokumentti ajetaan selaimessa, selain muodostaa siitä DOM-mallin.

```
...<!doctype html> == $0
  <html>
    <head>
      <meta charset="utf-8">
      <title>Template - Schedule</title>
      <link rel="stylesheet" type="text/css" href="styles.css">
    </head>
    <body>
      <main id="mainPage">
        <div id="mainContainer">
          <div id="stageHeader"></div>
          <div class="content">
            <article id="dayAndTime"></article>
            <section id="schedule"></section>
            <section id="specificInformation"></section>
            <section id="comingUp"></section>
          </div>
        </div>
      </main>
      <script src="script_collection.js"></script>
    </body>
  </html>
```

KUVIO 18. Ensimmäisen version DOM-malli suorituksen alussa.

Kuviossa 18 on ruudunkaappaus Chromen DevTools-työkalun Elements-osiosta, jossa näkyy ensimmäisen version DOM-malli sovelluskehyksen ajon alussa. Tässä vaiheessa DOM:iin ei ole vielä lisätty sisältöä JSON:in perusteella, joten mallista nähdään, miltä

HTML-runko-osa näyttää. Rungon HTML-elementit voidaan jaotella staattisiin ja dynaamisiin osiin. Staattisia osia ovat osat, joihin ei tehdä ajon aikana muutoksia tai päivityksiä. Dynaamisia osia taas päivitetään. Niitä ovat kuvion ”div”-elementti, jonka id on ”stageHeader”, ja johon luetaan lavan otsikkotiedot, ja ”div”-elementin, jonka luokka attribuuttina on ”content”, lapsielementit. Lapsielementtiin ”article” määritellään ja päivitetään sekunnin välein tieto ajon aikaisesta päivämäärästä, päivästä ja kellonajasta. Loput kolme ”section”-lapsielementtiä toimivat säiliöinä ohjelmanäkymille: elementtiin, jonka id on ”schedule”, generoidaan ohjelmallisesti koko ohjelman näyttämä näkymä, elementti id:llä ”specificInformation” sisältää nykyisen ja sitä seuraavan ohjelman tiedot ja elementti ”comingUp” id:llä sisältää tästä vielä seuraavien ohjelmien tiedot.

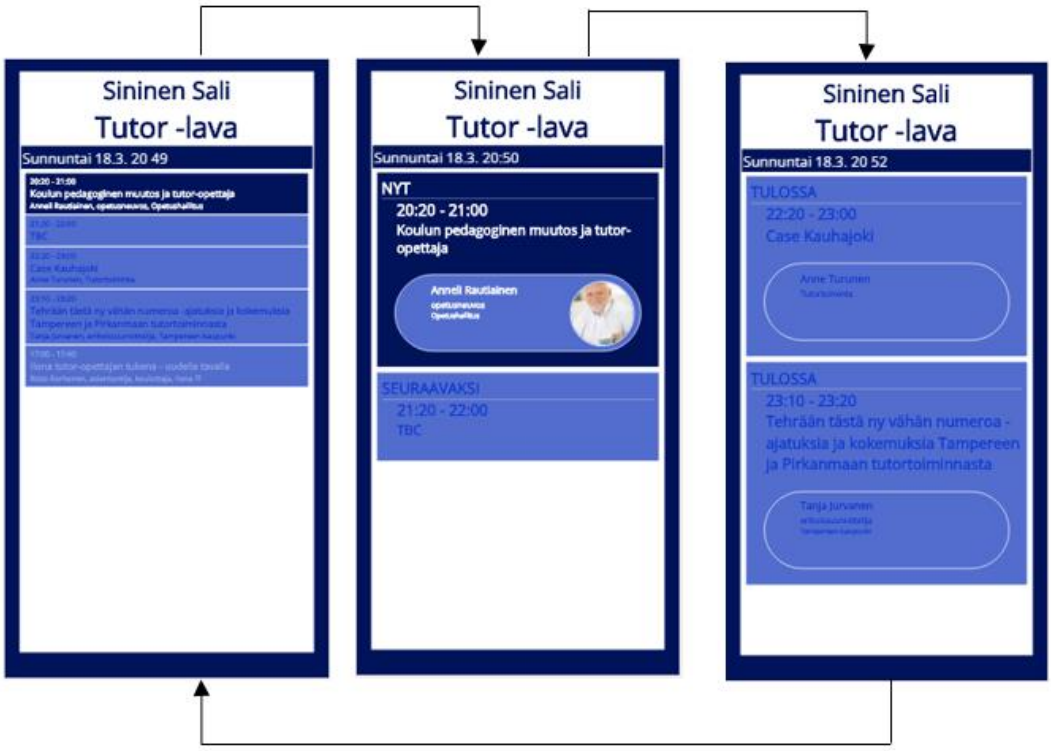
Sovelluskehyksen ohjelmallisesti suorituksesta huolehtii sen skripti. Se toimii sovelluskehyksen ”moottorina”. Ensinnäkin se parsii sille osoitetusta JSON-tiedostosta näkymään halutut ohjelmatiedot sekä sille osoitetut graafiset ominaisuudet. Graafiset ominaisuudet parsitaan ”style”-olioksi ja niiden sisältö lisätään kuvion 17 ”head”-elementin lapseksi, jolloin näkymä päivittyy niiden mukaiseksi. Ohjelmatiedot luodaan samaan tapaan kullekin DOM-mallin sisällöstä vastaavalle solmuelementille.

```
<!doctype html> == $0
<html>
▼<head>
    <meta charset="utf-8">
    <title>Template - Schedule</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
  ▼<style type="text/css">
      .past, .future, .nextProgramNumber, .comingUp1 { background-color:#526DCE; border: 1px
      solid #526DCE;}
      .future, .nextProgramNumber, .comingUp1 { color: #133CD3; }
      .past { color: #8B9DE0; }
      .current, .currentProgramNumber, #dayAndTime { color: white;background-color:#00135B; }
      #stageHeader h2, #stageHeader h3 { color: #00135B; }
      .content, #stageHeader, #mainPage {background-color:white; }
    </style>
  </head>
▼<body style="background-color: rgb(0, 19, 91);">
  ▼<main id="mainPage">
    ▼<div id="mainContainer">
      ▼<div id="stageHeader">
        ▼<article>
            <h3>Sininen Sali</h3>
            <h2>Tutor -lava</h2>
          </article>
        </div>
      ▼<div class="content">
          <article id="dayAndTime">Sunnuntai 25.3. 17:48</article>
        ▶<section id="schedule" style="display: block;">…</section>
          <section id="specificInformation" style="display: none;"></section>
        ▶<section id="comingUp" style="display: none;">…</section>
        </div>
      </div>
    </main>
    <script src="script_collection.js"></script>
  </body>
</html>
```

KUVIO 19. Ensimmäisen version suorituksen aikainen DOM-malli.

Kuviossa 19 on taas ruudunkaappaus selaimen DevTools-työkalun Elements-osiosta. Kaappaushetkellä on sovelluskehyksen suorituksessa päästy jo ylläpitovaiheeseen, jossa dynaamisia osiota päivitetään tietyn aikavälein. Kuviosta nähdään, että DOM:in on tullut lisää elementtejä, muun muassa ”head”-elementtiin on lisätty JSON-tiedoston perusteella tyyliluokkia ja -ohjeita. Näkymien vaihtotavan voi nähdä ohjelmanäkymien ”section”-elementeissä. Kaikkiin näihin on luotu ohjelmatietojen mukaan sisältöä skriptissä mutta vain yksi niistä on kerrallaan määrätty näkyväksi tyyliohjeistuksen ”display” attribuutilla. Mikäli näkymä on aktiivinen, on attribuutilla arvo ”block”.

Jos sisältöä kyseiselle päivälle ei JSON-tiedostossa ollut, lisätään näkymään tapahtuneesta kertova virheviesti. Skripti ylläpitää sovelluksen toimivuutta: se pitää näkymän ajan tasalla. Se siis tarkistaa tasaisin väliajoin, että näkymällä on aktiivisena oikea ohjelmanumero. Ohjelmanumeroiden vaihtuessa se päivittää näytölle sen hetkiselle ohjelmalle aktiivisen ohjelman tyyliluokan. Se myös suorittaa näkymien vaihtamisen.



KUVIO 20. Ensimmäisen version toiminnan kuvaus.

Kuviossa 20 on esitettynä ensimmäisen version ajonaikaiset näkymät. Vasemmalla puolella on koko ohjelman näyttämä näkymä, keskellä aktiivisen tapahtuman ja sitä seuraavan tapahtuman näyttämä näkymä ja oikealla vielä kaksi tulossa olevaa tapahtumaa. Nuolet kuvaavat siirtymiä näkymien välillä. Koko ohjelman näyttämästä näkymästä siirrytään aktiivisen tapahtuman näkymään ja tästä tulevien tapahtumien näkymään, josta lopulta palataan takaisin koko ohjelman näyttämään näkymään. Ohjelmanäkymässä on eroteltu aktiivinen eli käynnissä oleva ohjelma määrittämällä sille muista erottuva taustaväri (KUVIO 17 ”themeColor”-olion ”activeBackground”-attribuutti). Tämän lisäksi oikean puolen näkymässä nykyisestä ohjelmasta voidaan antaa lisätietoa, kuten puhujan kuva esimerkiksi tai ohjelmasta tarkemmin kertovaa tekstiä. Siirtymät näkymien välillä suoritettiin skriptissä ja niillä tarkoitetaan oikeastaan vain näkymän osion piilottamista ja toisen osion näyttämistä. Nämä vuorotellen näkyvät osiot ovat kuvioiden 18 ja 19 esittämän DOM-mallin ”section”-solmut.

Alkuperäiseen toimeksiantoon verraten (KUVIO 13) näkymien sisältö muuttui hieman. Yksittäisen ohjelman näyttämään näkymään lisättiin myös tieto mahdollisesta seuraa-

vasta ohjelmasta. Tätä ratkaisua päädyttiin käyttämäänkin ensimmäisen asiakkaan muissakin tapahtumissa, sillä ohjelmanumeroin tekstisisällön vähyys jätti näytölle tilaa ylimäärin. Lisäksi näkymien määrä kasvoi yhdellä, eli KUVIO 20 oleva tulevien tapahtumien näkymällä. Tästä seuraavan ohjelman näyttämästä näkymästä tuli sovelluskehyksen pysyvä ominaisuus. Nämä seuraavia ohjelmanumeroita korostavat ominaisuudet kehitettiin asiakkaan pyynnöstä.

3.4 Toinen versio

Toista versiota kehittäessä pyrittiin ottamaan huomioon ja korjaamaan joitain ensimmäisen version vaillinaisuuksia. Näitä olivat muun muassa JavaScript-koodin rakenteen refaktorointi, virheiden käsittely, JSON-tiedostossa olevan tiedon rakenteen ja ominaisuuksien parannukset ja puutteet käyttöliittymässä. Myös sovelluskehyksen HTML-tiedostosta muodostuvaa runkorakennetta kehitettiin edelleen.

3.4.1 Refaktorointi

Koodin refaktoroinnilla tarkoitetaan koodin kirjoittamista uudelleen selkeämmin ja yksinkertaisemmin. Sen tavoitteena on saada koodista muun muassa muille koodaajille helpommin ymmärrettävä ja yleisesti ottaen helpommin ylläpidettävä. (Refactoring Guru n.d.).

Ensimmäisen version skripti oli kirjoitettu Vanilla JavaScriptillä, sillä näyttölaitteen suorituskyvyn ollessa niin huono kuin se oli, tämän todettiin olevan tehokkain ratkaisu. Koodin refaktoroinnin yhteydessä testattiin, miten vaihtaminen helpompisyntaksiseen mutta vähemmän tehokkaaseen jQueryyn vaikutti sovelluksen suorittamiseen. Kun jQueryyn vaihtamisella ei ollut näkyvää vaikutusta näytölle renderöinnin suhteen, ohjelmoitiin toisen version skripti kokonaan jQueryllä. Näin saatiin koodista helpommin luettava ja yksinkertaisemmin kirjoitettava jQueryn syntaksin vuoksi.

```
document.getElementById("content").style.display = "none";
$("#content").hide();
```

KUVIO 21. JavaScript -ja jQuery -syntaksin vertailua.

Kuviossa 21 on demonstroitu esimerkillä syntaksien eroa. Sen ensimmäisellä rivillä piilotetaan DOM-solmu, jonka id on "content" tavallisella JavaScriptillä. Toisella rivillä tehdään sama asia samalle solmulle jQueryllä huomattavasti yksinkertaisemmin.

Ensimmäisen version skriptissä oli huomattu olevan usean sadan rivin funktiokokonaisuuksia. Toista versiota ohjelmoitaessa pyrittiin funktioiden paisumista usean sadan rivin kokonaisuuksiksi tietoisesti välttämään, ja aina kun funktio alkoi lähestyä tai ylitti 50:tä koodiriviä, koitettiin se pilkkoa useammaksi funktioksi, mikäli mahdollista.

3.4.2 Muutokset JSON-tiedostoon

Ensimmäisessä versiossa sovelluksen JSON pystyi säilömään vain yhden tapahtuman tiedot. Tämä tarkoitti käytännössä sitä, että kahden peräkkäisen tapahtuman sattuessa olisi pitänyt kullekin tapahtumalle tehdä oma tiedostonsa ja päivittää tämä näyttölaitteelle tapahtumien välissä. Toiseen versioon kehitettiin JSON:sta usean tapahtuman säilömisen mahdollistava, joten edellä mainitussa kahden tapahtuman esimerkissä molemmat tapahtumat pystyttäisiin säilömään yhteen tiedostoon. Näin ollen tapahtumien välillä ei tarvitse päivittää tiedostoa ja ylläpitämiseen vaadittava työmäärä vähenee.

JSON-tiedostoon lisättiin myös uusia attribuutteja, joilla pystyttiin määräämään muun muassa se, kuinka monta ohjelmaa yhdellä koko ohjelma-näkymällä näytetään. Tämän tiedon perusteella skriptissä laskettiin näytettävien näkymien määrä. Ensimmäisen version kehityksen aikana oltiin huomattu, että ohjelmanumeroiden määrän kasvaessa alkoi ohjelmanäkymästä loppua tila kesken. Ensimmäisen version kehityksen yhteydessä ongelma pystyttiin ohittamaan pienentämällä käytettyä fonttikokoa ja elementtien välisiä etäisyyksiä. Tällainen menettely voisi kuitenkin johtaa riittävän usean ohjelmanumeron kohdalla siihen, että tietojen luettavuus kärsisi. Lisäksi dynaamisesti muuttuvan fonttikoon ohjelmointi olisi vaatinut enemmän kehitystyötä, joka olisi ollut pois muilta tärkeämmiltä kehityskohteilta, kuten toimivan funktionaalisuuden takaamiselta. Toiseen versioon ongelma korjattiin sisällyttämällä mahdollisuus määrittää etukäteen yhdellä ohjelmasivulla näytettävien ohjelmien määrä. Esimerkiksi jos ohjelmanumeroita olisi 12, voitaisiin ohjelmanumerot jakaa kahdelle eri ohjelmasivulle antamalle näytettävien ohjelmien lukumääräksi 6. Sovelluskehyksen suorituksen alussa skripti laskee tämän perusteella, kuinka monelle näkymälle koko ohjelma jakautuu, ja luo riittävästi näkymäelementtejä DOM:in. Näin ei tarvitse uhrata luettavuutta fontin pienentämisellä.

Toinen lisätty attribuutti oli mahdollisesti näytettävän logon tiedostonimi. Tarve tähän tuli asiakkaalta, joka halusi näkymään oman logonsa. Tällä attribuutilla määritetään otsikkotilassa käytettävän logon tiedosto.

3.4.3 Ulkoasun eli käyttöliittymän muutokset

Toinen messuohjelman tilannut asiakas ilmoitti tapahtumansa teemaväriksi oranssin. Lisäksi asiakas halusi näkymän taustakuvassa käytettävän tapahtuman graafisia elementtejä, jotka olivat mustia. Näin ollen päädyttiin oranssilla vain korostamaan tiettyjä elementtejä, kuten otsikkoa ja aktiivisen tapahtuman elementtiä määrittämällä sille oranssit reunat ja otsikon värinä. Muuten näkymän fontit olivat mustalla ja graafisista elementeistä muodostettu taustakuva lähinnä valkoinen (KUVIO 23).

Mahdollisuus vaikuttaa näkymien lukumäärään edellisessä kappaleessa mainitun ohjelman pilkkomisen myötä lisäsi tarvetta viestiä katsojalle, missä vaiheessa sovelluksen suoritusta oltiin. Tämä on yksi hyvään käytettävyyteen liitettävä asia, esimerkiksi Nielsen (1993, 20) listaa palautteen tapahtuvasta prosessista yhdeksi kohdaksi, mikä olisi hyvä ottaa huomioon suoritettaessa käyttöliittymän heuristista arviota.

Sovelluskehyksessä tämä toteutettiin piirtämällä HTML-näkymän alareunaan sijoitettuun ”canvas”-elementtiin näytettävien sivujen lukumäärän verran ympyröitä JavaScriptissä.

```
545  function pager(index) {
546    var canvas = document.getElementById("pager");
547
548    var ctx = canvas.getContext("2d");
549    ctx.clearRect(0, 0, canvas.width, canvas.height);
550    for (var i = 0; i < contentIds.length; i++) {
551
552      ctx.beginPath();
553      var r=30
554      ctx.arc(100 + (i*r), 20, 10, 0, 2 * Math.PI, false);
555      if (i == index) {
556        ctx.fillStyle = "#CF681D";
557      } else {
558        ctx.fillStyle = "#eeeeee";
559      }
560      ctx.fill();
561      ctx.closePath();
562      ctx.stroke();
563
564    }
565  }
```

KUVIO 22. ”Canvas”-elementiin piirtäminen JavaScriptillä.

Kuviossa 22 on esitettynä toisen version ”pager”-funktio, jolla suoritettiin näkymässä ajon tilaa indikoivien ympyröiden piirtäminen. Funktiolle välitetään parametrina aktiivisen näkymän kertovan indeksin numero. Rivillä 546 haetaan ”canvas”-muuttujaan DOM-solmu, jonka id on ”pager”, ja joka tässä tapauksessa käsittää HTML-dokumentin ”canvas”-elementin. Tämän jälkeen rivillä 548 tallennetaan ”ctx”-muuttujaan ”canvas”-olion 2D-renderöintiympäristö funktiolla ”getContext” ja parametrillä ”2d”. Ennen piirtämistä pyyhitään rivillä 549 pois kaikki aiemmin piirretty funktiokutsulla ”clearRect”, jolle annetaan parametreina alkupisteen koordinaatit ja piirtoalan koko. Rivillä 550 aloitetaan ”for”-silmukka, jota suoritetaan näkymien määrän ajan. Yhden silmukan aikana piirretään yksi ympyrä indikoimaan näkymää. Piirtäminen aloitetaan kutsumalla renderöintiympäristön funktiota ”beginPath”, joka aloittaa uuden piirtopolun. Tähän polkuun lisätään ympyräkuvio toisella renderöintiympäristön funktiolla ”arc”, jolle annetaan parametreiksi keskipisteen koordinaatit, ympyrän säde, aloituskulma, lopetuskulma ja vapaaehtoinen totuusarvo määrittämään piirtosuuntaa. Silmukassa tarkistetaan rivillä 555, onko kyseinen indeksi aktiivisen näkymän indeksi, ja jos näin on, annetaan piirrettävälle ympyrälle muista ei-aktiivisista ympyröistä poikkeava väri renderöintiympäristön funk-

tiokutsulla ”fillStyle” ja värin ilmoittamalla parametrilla. Väritys viimeistellään renderöintiympäristön funktiolla ”fill”. Lopuksi aloitettu piirtopolku suljetaan renderöintiympäristön funktiolla ”closePath” ja piirto ”canvas”-elementtiin suoritetaan funktiolla ”stroke”. (Mozilla Developer Network 2018).

Muutoksia tehtiin myös asiakkaan toiveiden perusteella. Toisen asiakkaan kohdalla ilmeni halu saada asiakkaan logo sisällytettyä näkymään. Tälle varattiin tila näkymän alalaidasta. Lisäksi otsikolle varattuun tilaan oli mahdollista asettaa myös kuva tekstin sijaan, kun ei oltu vielä varmoja siitä, mihin kohtaan näkymästä asiakas halusi logonsa sijoittaa. Asiakas halusi näkymistä poistettavaksi myös ohjelmien loppumisajan, joten tämäkin toteutettiin kommentoimalla skriptistä pois lopetusajan määrittelevä koodin pätkä.

KUVIO 23. Toisen version ajonaikaiset näkymät.

KUVIO 23 on esitettynä toisen version ajonaikaiset näkymät, kun koko ohjelmanäkymä on jaettu kuuden ohjelman osissa näytettäviin näkymiin. Kaksi ensimmäistä näkymää luettaessa vasemmalta oikealle näyttävät koko ohjelman, jonka jälkeen näytetään sen hetkistä ohjelmaa. Tämän jälkeen näytetään nykyistä seuraava ohjelma numero, jos sellainen on. Yleisesti näkymässä, mikäli nykyistä ohjelmaa ei ole, näytetään seuraava tulossa oleva ohjelma. Jos ohjelmia ei enää päivän aikana ole, näytetään vain koko ohjelman näyttävä(t) näkymä(t).

Verrattuna ensimmäiseen versioon (KUVIO 20), toisen version näkymien sisällöt ovat myös muiden kuin koko ohjelman osalta muuttuneet. Nykyinen ja sitä seuraava ohjelma olivat ensimmäisessä versiossa aluksi samassa näkymässä. Kahteen erilliseen näkymään päädyttiin toisessa versiossa sen takia, kun asiakkaan aineistoissa alkoi olla enemmän ohjelmanumeroa kuvaavaa tekstiä. Kahta ohjelmanumero ei olisi kannattanut sijoittaa samalle näkymälle, sillä tekstiä ja muita elementtejä olisi pitänyt pienentää liikaa, eikä kaikkea näkymän tekstiä olisi kerennyt lukemaan.

3.4.4 Virheilmoitukset

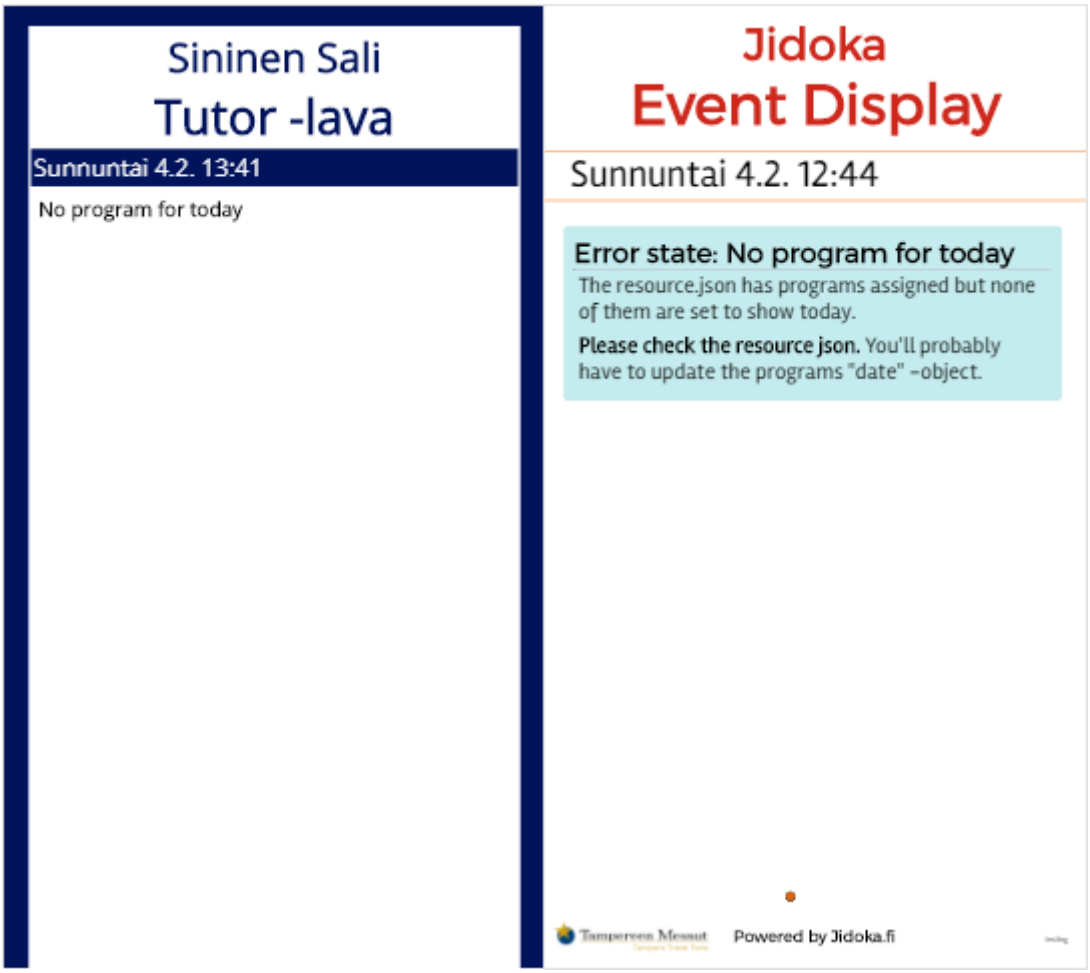
Ensimmäisessä versiossa ei ollut toteutettu varsinaista virheenkäsittelijää eikä virheviestejä lainkaan. Ainoa tällaista tilannetta kuvaava tilaviesti oli sovelluksen ”No program for today”-ilmoitus, joka tarkoitti, että annetusta JSON -tiedostosta ei löytynyt sen hetkiselle päivälle ohjelmaa.

Tätä toiminnallisuutta parannettiin huomattavasti toisen version kohdalla. Skriptiin tehtiin oma virheenkäsittelijä, joka ilmoitti sille välitetystä attribuutista riippuen näytöllä virhetilaan johtaneen tapahtuman selityksen. Esimerkiksi mikäli päivälle ei löytynyt ohjelmaa, saattoi kyseessä olla kaksi eri virhetilaa: 1: JSON:ssa ei ole ohjelmia ollenkaan tai 2: mikään JSON:n ohjelmista ei ole osoitettu kyseiselle päivälle. Näiden tietojen lisäksi ehdotetaan tarkistamaan JSON-tiedosto, sillä sieltä virhetila on alkujaan peräisin. Virhetilan 2 tapauksessa ehdotettiin myös ohjelman päivämäärä-attribuutin päivittämistä.

Muita käsiteltäviä virhetilanteita oli JSON-syntaksivirhe. Määrittelevää JSON-tiedostoa muokattiin tekstieditorista käsin, ja vaikka se on kehittäjälle ymmärrettävää, muokkasivat sitä muutkin Jidokan työntekijät. Vaikka internetistä löytää JSON-syntaksin tarkistavia työkaluja, kuten JSONLint (JSONLint 2018), oli olemassa mahdollisuus, että tarkistamaton ja virheellinen tiedosto ladattaisiin näyttölaitteelle. Tällaiselle virheelle tehtiin oma virheenkäsittelijänsä ja sen aiheuttajasta annetaan oma virheviestinsä.

Näytöllä näkyvistä virheilmoituksista pyrittiin tekemään hyvän käytettävyyden mukaisia toisessa versiossa. Hyvän käytettävyyden mukaiset virheilmoitukset ovat Kuutin (2003, 62) mukaan viestejä, jotka ovat neutraaleja tai kohteliaita, selkokielisiä, rakentavia ja tarkkoja. Kuutti korostaa, että virheilmoituksen sisällön pitäisi olla ymmärrettävä sellaisenaan ilman ohjekirjaa tai muiden pitkien dokumenttien selailua. Ensimmäisen version

”No program…” -ilmoitus ei kertonut mahdollisesta syystä laajasti eikä ehdottanut tilanteeseen sopivaa korjausta. Toisessa versiossa tämä pyrittiin korjaamaan.



KUVIO 24. Molempien versioiden virheviestit samasta virhetapahtumasta.

Kuviossa 24 on esitettynä sovelluskehyksen ensimmäisen ja toisen version virheilmoitus, kun JSON-tiedostosta ei löydy päivälle ohjelmaa. Toisen version virheilmoitukseen on sisällytettynä tilan syy: tiedostossa ei ole ohjelmia tälle päivälle, sekä korjausehdotus: tarkista ohjelmatiedot sisältävä JSON, joka mainitaan tarkkuuden vuoksi nimeltä (resource.json). Tämän lisäksi käyttäjälle annetaan vielä tarkempi korjausehdotus, joka ehdottaa päivittämään ohjelman ”date”-olion, sillä jotain ohjelmaa JSON-tiedostossa on, mutta kyseiselle päivälle niitä ei ole osoitettu.

3.4.5 Rakenteelliset muutokset

Sovelluskehyksen toiminnallisuus pysyi toisessa versiossa kuta kuinkin samanlaisena ensimmäiseen versioon verrattuna. Rakennetta eli HTML-perustaista runkoa sen sijaan

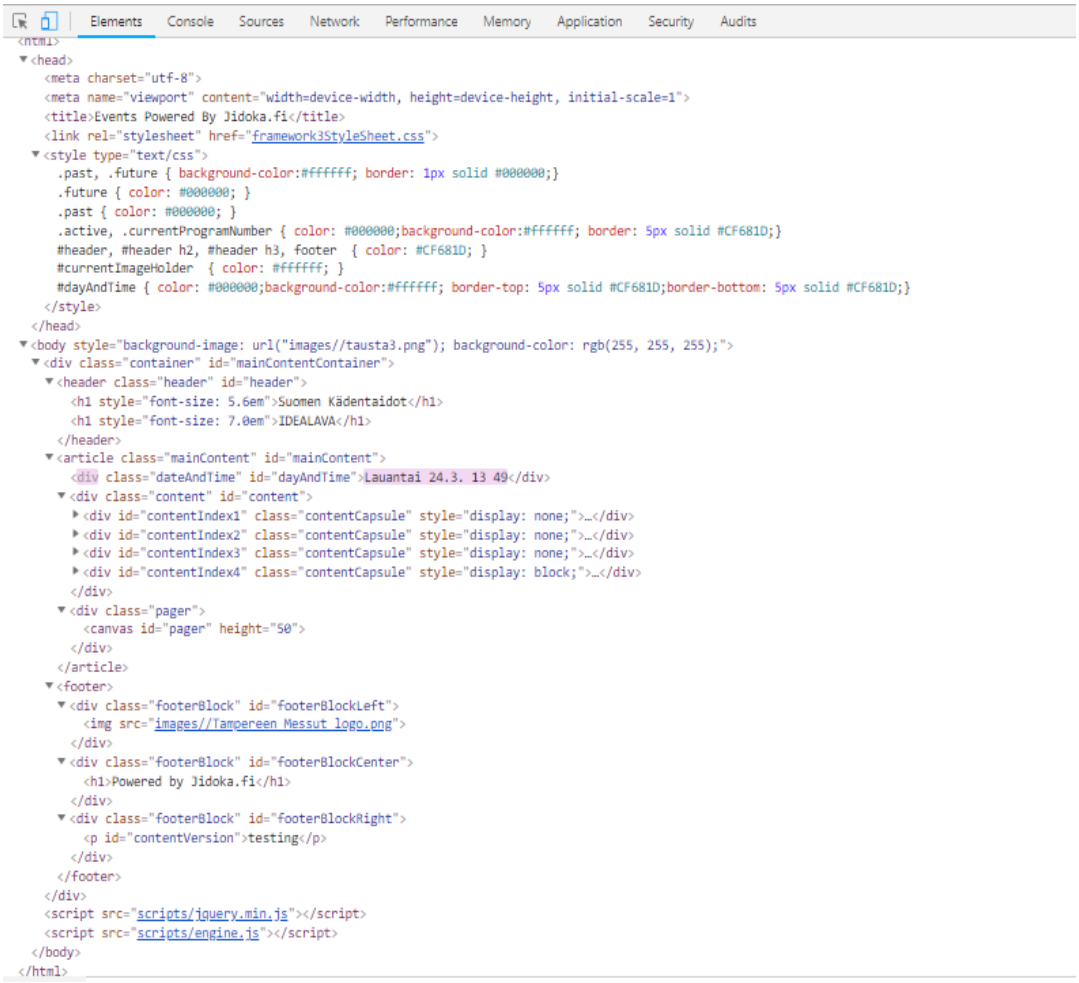
muutettiin jonkin verran. Tämä tarve kumpusi useastakin eri syystä: asiakas halusi lisätä oman logonsa näkymään ja Jidoka lisätä yrityksen nimi näkymään lisätäkseen näkyvyyttään ohjelmistokehittäjänä. Lisäksi uudet ominaisuudet sovelluskehyksen ulkoasussa vaativat rakenteen muokkaamista ja uusien elementtien lisäämistä.

```
<!doctype html> == $0
<html>
▼<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>Events Powered By Jidoka.fi</title>
    <link rel="stylesheet" href="framework3StyleSheet.css">
  </head>
▼<body>
  ▼<div class="container" id="mainContentContainer">
      <header class="header" id="header">
          </header>
    ▼<article class="mainContent" id="mainContent">
        <div class="dateAndTime" id="dayAndTime"></div>
        <div class="content" id="content"></div>
      ▼<div class="pager">
          <canvas id="pager" height="50">
        </div>
      </article>
    ▼<footer>
      ▼<div class="footerBlock" id="footerBlockLeft">
          <img src="images//Tampereen Messut logo.png">
        </div>
      ▼<div class="footerBlock" id="footerBlockCenter">
          <h1>Powered by Jidoka.fi</h1>
        </div>
      ▼<div class="footerBlock" id="footerBlockRight">
          <p id="contentVersion"></p>
        </div>
      </footer>
    </div>
    <script src="scripts/jquery.min.js"></script>
    <script src="scripts/engine.js"></script>
  </body>
</html>
```

KUVIO 25. Toisen version DOM-malli suorituksen alussa.

Kuviossa 25 on ruudunkaappaus selaimen DevTools-näkymästä ennen kuin skriptissä on lisätty mitään HMTL-dokumenttiin. Elementin ”body” sisällä on vain yksi ”div”-elementti id:llä ”mainContent”, joka pitää sisällään näkymän muut osiot. Näkymän osiot on jaettu semanttisiin elementteihin ”header”, ”article” ja ”footer”. Näistä ”header” on varattu otsikolle, oli se sitten kuva tai tekstitieto. Elementti ”article” sisältää tiedot päivämäärästä, ajasta ja päivästä, ohjelmatiedot ja osion sovelluksen tilan indikaattorille. Nämä ovat myös HTML-rungon dynaamiset elementit, joita päivitetään jatkuvasti ajon aikana. Ohjelmatiedot sisältävään ”div”-elementtiin, jonka id on ”content” ei ole ensimmäisen version tavoin valmiiksi koodattu säiliöitä tietyn tyyppisille ohjelmanäkymille. Toisessa versiossa ei voitu etukäteen tietää näkymien lukumäärää, sillä se oli täysin riippuvainen siitä, montako ohjelmaa yhdessä koko ohjelman näyttämässä näkymässä haluttiin näyttää

ja kuinka monta ohjelmaa päivän aikana olisi. Näkymien lukumäärä määritettiin vasta myöhemmin skriptin ajon aikana, ja sen perusteella luotiin myös oikea määrä sopivia näkymäelementtejä DOM:iin. Muita muutoksia olivat tilan indikoiva ”canvas”-elementti ja yksi ”script”-elementti lisää jQueryn sisällyttämisen vuoksi.



KUVIO 26. Toisen version DOM-malli selaimessa suorituksen ylläpitovaiheessa.

Kuviossa 26 on ruudunkaappaus selaimen DevTools-työkalun Elements-osiosta, kun sovelluskehyksen suorituksessa on päästy ylläpitovaiheeseen. Kuviosta nähdään, että kuten ensimmäisen version kohdalla, on dokumentin ”head”-elementtiin lisätty tyyliohjeita. Lisäksi otsikolle on määritelty tekstisisältö ja ohjelmanäkymään luotu neljä ”div”-elementtiä, jotka sisältävät näyttölaitteella vaihtuvat näkymät. Kuviosta 22 näkee ajonaikaiset näkymät sisältöineen.

4 JATKOKEHITYS

Mikäli sovelluskehystä kehitettäisiin edelleen, olisi siihen muutama luonnollinen kehityksen kohde. Näitä ovat sommittelun ja asettelun lisäksi graafinen käyttöliittymä ja tietokantalaajennus. Lisäksi sovelluskehykselle voisi määrittää muita hallittavia ominaisuuksia.

4.1 Graafinen käyttöliittymä

Tämänhetkisessä versiossa sovelluskehyksen sisältöä hallinnoidaan siihen liitetyn JSON:in avulla. JSON:in muokkaaminen on haastavaa: se vaatii oikeanlaisen syntaksin toimiakseen oikein, eikä se sisällä syntaksin tarkistusta itsessään. Myös suurien ohjelmakokonaisuuksien hallinnassa JSON-tiedosto on työläs. Lisäksi teeman värien ja sisällön muotoilun testaaminen on hidasta: sovelluskehys tietoineen tulee ladata sisällönhallintajärjestelmään, jonka jälkeen sille osoitettu näyttölaite pitää käynnistää uudelleen. Nämä ovat asioita, joita voidaan helpottaa ja nopeuttaa graafisen käyttöliittymän avulla.

Graafisella käyttöliittymällä hoidettaisiin niin ohjelmatietojen lisääminen kuin teemavärien määrityskin. Käyttöliittymä toimisi osana jo valmista sisällönhallintajärjestelmää, ja tästä näkisi värit ja sommittelun välittömästi. Kuten useimmissa nykyisissä verkkolomakkeissa, suoritettaisiin syötteen tarkistus ja vaadittujen tietojen oikeellisuuden tarkastaminen jo tietoja syötettäessä. Syötteen tarkistuksen yhteydessä voitaisiin myös paremmin hallita näkymän luettavuutta, esimerkiksi rajoittamalla tiettyjen tekstikenttien pituutta. Tämä estäisi näkymän elementtien ”paisumista” näytön rajojen ulkopuolelle, ja toisaalta rajoittaisi tiedon määrää näyttölaitteella rajatussa ajassa luettavaksi ja omaksuttavaksi riittävän hyvin. Graafinen käyttöliittymä mahdollistaisi sen, että käyttäjän ei tarvitsisi huolehtia syntaksin oikeudesta. Käyttöliittymä huolehtisi myös, että kaikki pakolliset tiedot olisi annettu ennen kuin tiedot voidaan tallentaa järjestelmään. Graafinen käyttöliittymä parantaisi tuotteen käytettävyyttä ja tekisi siitä muutenkin kokonaisvaltaisemman kokonaisuuden tuotekehityksen suhteen.

4.2 Tietojen tallennus tietokantaan

Nykyisessä sisällönhallintajärjestelmässä on kullekin asiakkaalle omat tiedot tietokannassa. Tätä jo olemassa olevaa ominaisuutta voitaisiin jatkaa kehittämällä tietokantaan

uusi relaatio asiakkaalle, jolla on käytössä sovelluskehys. Relaatioon voitaisiin tallettaa asiakkaan omia väriteemoja ja ohjelmatietoja. Tämä helpottaisi esimerkiksi jo kertaalleen näyttölaitteelle asetettujen tietojen muuttamista. Tietokannasta voitaisiin hakea tietyn tapahtuman ohjelma- ja väritiedot päivitystä varten. Kehityksen aikaisten asiakkaiden perusteella on tälle vahvat perusteet: useimpien tapahtumien kohdalla tuli ohjelmiin muutoksia jopa niin myöhäisessä vaiheessa kuin päivää ennen tapahtumaa ja usein oli tarvetta korjata kirjoitusvirheitä.

4.3 Muut mahdolliset kehitysalueet

Sovelluskehyksen elementtien animointimahdollisuuksia voisi kehittää edelleen. Animoinnista itsestään voisi tehdä valittavan ominaisuuden, jota hallittaisiin JSON-tiedostosta käsin, esimerkiksi jollain totuusehdolla. Lisäksi animaatiolle voisi edelleen JSON:ssa määrittää CSS-animaatioiden attribuutteja, kuten KUVIO 15:sta esimerkissä esitetyn animaation ajan ja animaatiotavan.

Toinen attribuutti sovelluksen suorittamisen hallintaan olisi määrittää JSON-tiedossa näkymien haluttu näyttöaika eli kuinka kauan yksittäistä näkymää näytetään. Lisäksi sovellukseen voisi asettaa kielivalintamahdollisuuden. Ensimmäiset mahdolliset valinnat tähän voisivat olla englanti ja suomi. Kehityksen aikaiset asiakkaat molemmat olivat suomenkielisiä mutta englannin kielen mahdollisuuden lisääminen kasvattaisi palvelun sopivuutta suuremmille markkinoille. Kielenvalinta vaikuttaisi näytettävän päivämäärän kirjoitukseen ja virheviestiin.

5 POHDINTA

Työn tavoitteena oli ensisijaisesti kehittää Jidokalle HTML5-sovelluskehys, joka pohjautuisi heidän asiakkaansa tarpeisiin, ja jota he voisivat yksinkertaisin muutoksin käyttää myös muiden asiakkaiden kohdalla. Tälle sovelluskehykselle määriteltiin reunaehdoiksi hyvä käytettävyys ja asiakaslähtöinen kehitys.

Asiakaslähtöinen kehitys pyrittiin varmistamaan ottamalla asiakas mukaan kehitykseen. Tämä tapahtui sähköpostin ja puhelimen välityksellä: kun sovelluskehyksestä oli saatu versio näyttövalmiiksi, lähetettiin se asiakkaalle arvioitavaksi sähköpostilla. Joidenkin tapahtumien osalta saatiin palautetta asiakkaalta myös puhelimitse. Sovelluskehystä muokattiin mikäli asiakkaalla oli joku kehitysehdotus tai toive ominaisuuksien suhteen. Lisäksi toisen asiakkaan kohdalla tehtiin asiakkaan tapahtuman elementeistä oma taustakuva sovelluskehykseen, ja HTML-dokumentin runkorakenne muokattiin sellaiseksi, että siihen pystyttiin sisällyttämään asiakkaan haluama logo. Näissä puitteissa koettiin asiakaslähtöisen kehityksen olleen kokonaisvaltaisesti onnistunutta.

Ensimmäisen version kohdalla asiakaslähtöisyydestä kummunnut tavoite saavutettiin mutta ei täydellisesti. Tapahtuman, jossa sovelluskehystä oli tarkoitus käyttää ensimmäisen kerran, ensimmäisenä päivänä sovellusta käyttöön otettaessa sovellus ei palautunut virhetilasta. Virhetilanteeseen jumiutumisen mahdollisuus oli tiedossa, ja sovellus olisi toipunut tästä, jos laite olisi käynnistetty uudelleen, mutta tieto toipumiseen tarvittavista toimenpiteistä ei ollut käyttäjillä tiedossa. Virhetila saatiin korjattua tunnin sisällä tapahtuman alkamisesta mutta asiakas oli tyytymätön palveluun. Laskutettaessa asiakas pyysi sattuneesta johtuen, ja oikeutetusti, palvelusta alennusta. Tästä seurasi Jidokalle siis taloudellista tappiota ja palvelu ei täysin toteuttanut hyvän asiakaspalvelun vaatimuksia.

Toisen version kohdalla asiakaslähtöisen kehityksen tavoite saavutettiin. Asiakas sai haluamansa tuotteen ja oli tyytyväinen Jidokan tarjoamaan palveluun. Toisen version kohdalla ei ollut samanlaista huolta saada toimivaa versiota kehitettyä. Ensimmäisellä versiolla oltiin jo todettu teknologioiden toimivuus ja kokonaisuuden rakenne oli selvä. Toisen version kohdalla piti vain muokata päivitystä tarvitsevia osia koodista ja HTML-runkorakenteesta, ja jotain ensimmäisen version toimintoja voitiin käyttää sellaisenaan.

Kehityksen jälkeen sovelluskehystä on käytetty ainakin kerran erään Jidokan asiakkaan tapahtumassa. Voidaankin siis perustellusti väittää sovelluskehyksen olleen hyödyllinen Jidokalle. Lisäkehityksellä siitä voisi kehittää oman kokonaisvaltaisen tuotteensa muiden Jidokan palveluiden ohelle.

Toisessa versiossa saatiin korjattua ensimmäisen version käytettävyysongelmia hyvin. Toinen versio ei ollut käytettävyyden suhteen kuitenkaan täysin ongelmaton. Yksi käytettävyyden epäkohta liittyi virheviestiin ja siinä käytettyyn kieleen. Nielsenin listan mukaan vuorovaikutuksessa tulee käyttää käyttäjän kieltä (Kuutti, 49). Virheviesti on kuitenkin kirjoitettu englanniksi, vaikka asiakkaan tiedettiin olevan suomalainen ja puhuvan suomen kieltä. Englanniksi virheviesti kirjoitettiin, koska viesti sijaitsi koodin seassa. Koodissa oli käytetty englantia muuttujien ja funktioiden kuvaamiseen, joten oli luonnollista kirjoittaa koodin seassa oleva virheviesti myös englanniksi.

Yksi käytettävyyteen liittyvä ongelma liittyy tapaan, jolla käyttäjä on vuorovaikutuksessa sovelluskehyksen kanssa. Molemmissa versioissa vuorovaikutus tapahtuu lähinnä JSON-tiedoston kautta. Vuorovaikutuksesta mainitaan Nielsenin listassa käyttäjän kielen huomioimisen lisäksi se, että sen tulee olla yksinkertaista ja luonnollista. JSON-tiedoston muokkaamista ei voida kuvailla yksinkertaiseksi tai luonnolliseksi. Syntaksin oikeinkirjoitus vaatii tarkkaavaisuutta sääntöjensä vuoksi eikä ole näin kovin yksinkertaista. Vuorovaikutuksen luonnollisuudella tarkoitetaan sitä, että käyttöliittymän tulisi käyttää normaaleja arkipäiväisestä elämästä tuttuja konsepteja (Kuutti, 51), ja tätä JSON-tiedoston syntaksi ja esimerkiksi värikoodien heksadesimaalimuodot eivät useimmille ihmisille ole.

Jos sovelluskehyksen kehitystä jatkettaisiin, ja päästäisiin tekemään sille graafinen käyttöliittymä, voitaisiin vuorovaikutukseen ja käytettävyyteen yleisestikin kiinnittää enemmän ja syvällisemmin huomiota. Työn aikana käytettävyyttä pyrittiin ensisijaisesti huomioimaan näyttöjen ulkoasussa ja luettavuudessa, joten kritiikki JSON-tiedoston käytettävyyttä kohtaan ei ole niin vakavaa. Asiakkaiden tapahtumien kohdalla tiedostoa käsiteltiin Jidokan työntekijöiden kesken, eikä tapahtumien järjestäjien tarvinnut siis huolehtia siitä. Tosin palvelua ei olisi voinutkaan tarjota sellaisenaan yksin tapahtumien järjestäjille sen monimutkaisuuden ja keskeneräisyyden takia.

Työssä käytetyt teknologiat toimivat moitteettomasti, vaikka ne valittiinkin lähinnä kehitysajan rajallisuuden ja tuttuuden perusteella eikä ominaisuuksiensa ja etujensa perusteella. HTML5:n käytön voidaan sanoa olleen perusteltua sen tarjoaman piirto-ominaisuuden perusteella. Semanttisten elementtien käyttö ei tuonut työhön niiden käytön tarjoamaa täyttä hyötyä, sillä sovelluskehys ei ollut Internetistä löydettävissä. Toisaalta näistä ei koitunut kehitykselle mitään haittaa ja kehittäjän tietämys HTML5:n suhteen kasvoi, joten käyttö ei ollut täysin tuloksetontakaan. Lisäksi voidaan spekuloida semanttisten elementtien tosiaan parantavan HTML-dokumentin luettavuutta, kun osiot eroavat heti tagiensa perusteella toisistaan. Toisessa versiossa käytetty JavaScript-kirjasto jQuery ei ole viimeisin kehitetty kirjasto vaikka onkin yksi Internetissä käytetyimmistä kirjastoista useiden lähteiden mukaan (Envato Tuts+ 2017, BuiltWith n.d., Medium 2017.) Tehokkaamman ja uudemman kirjaston käyttäminen olisi yksi mielenkiintoinen kehityskohde, mikäli kehitystä jatkettaisiin.

Yksi potentiaalinen vaihtoehto jQuerylle olisi React-kirjasto. React on varta vasten käyttöliittymien kehittämiseen erikoistunut JavaScript-kirjasto ja sen Virtual DOM-ominaisuus tekee siitä mielenkiintoisen ehdokkaan jQuerylle. Sovelluskehyksen toiminta nykyisenään pohjautuu selaimen DOM:n suoralle manipulaatiolle. Reactissa tätä manipuloidaan sen Virtual DOM:n kautta. Kuten oikea DOM, Virtal DOM on solmupuu, joka listaa elementit ja niiden attribuutit ja sisällön olioina ja niiden ominaisuuksina. Aina kun React-sovelluksessa muuttuu jotain, luodaan uusi Virtual DOM-malli käyttöliittymässä. Selaimen DOM:n päivitys tapahtuu kuitenkin vasta sitten, kun React on määritellyt tarkalleen ne osat, mitä käyttöliittymässä on muuttunut. Selaimen DOM:iin siis päivitetään vain ne osat, joita on oikeasti muutettu jotenkin. Nykyisessä työssä DOM:n päivittäminen aiheuttaa sen, että selain määrittelee koko sivun sommittelun uudestaan, mikä on hidasta. (React n.d., Minnick 2016). React siis voisi tällä perusteella tehdä sovelluskehyksestä mahdollisesti tehokkaamman.

Toisaalta Reactin käyttöönotto sovelluskehyksessä vaatisi huomattavasti enemmän kehitysaikaa, sillä kehittäjän pitäisi opetella täysin uusi kirjasto. Reactin käyttöönotto palvelisi kehittäjän ammatillista kehitystä, sillä React on myös yksi Internetin suosituimmista JavaScript-kirjastoista, ja siten oiva lisä web-kehittäjän teknologiaosaamiseen (Medium 2017). Mikäli kehittämistä jatkaisi jo Reactia osaava kehittäjä, pitäisi varmistaa, että loppulaite tukee uutta teknologiaa ennen teknologian vaihtamista, ja että laitteen suorituskyky riittäisi Reactille.

Digitaalisella messuohjelmalla on huomattavia etuja perinteiseen paperiseen messuohjelmaan verrattuna. Digitaalisella messuohjelmalla voidaan korostaa juuri käynnissä olevaa ohjelmaa ja väreillä ohjelmanumeroiden tilan viestittäminen sallii katsojan nopeasti erottaa, mikä käynnissä oleva ohjelma on. Tätä ei ole mahdollista tehdä paperisella messuohjelmalla. Paperisella messuohjelmalla on myös puolensa, kuten jokaisen tapahtuman kohdalla erikseen suunniteltavat teemat ja asettelut elementeille ovat neuvoteltavissa graafisessa ilmaisussa vapaammin kuin digitaalisessa messuohjelmassa. Toisaalta web-ohjelmointia tunteva graafinen suunnittelija pystyy tekemään digitaalisella messuohjelmallakin yhtä yksilöllisiä visualisointeja tapahtumiin. Paperisessa ohjelmassa on myös potentiaalisesti saatavilla koko ajan ohjelmien lisätiedot, jotka digitaalisessa messuohjelmassa ovat näkyvillä vain aktiivisen ja seuraavan ohjelman kohdalla. Tässä digitaalisessa messuohjelmassa olisi parantamisen varaa.

LÄHTEET

BuildWith. N.d. JavaScript Library Usage. Luettu 24.3.2018. https://trends.built-with.com/javascript/javascript-library

Ecma International. 2017. Standard ECMA-404. Luettu 15.2.2018. http://www.ecma-international.org/publications/standards/Ecma-404.htm

Envato Tuts+. 19.10.2017. Essential JavaScript Libraries and Frameworks You Should Know About. Luettu 24.3.2018. https://code.tutsplus.com/articles/essential-javascript-libraries-and-frameworks-you-should-know-about--cms-29540

JSON. N.d. Introducing JSON. Luettu 17.1.2018. http://json.org/

JSONLint. 2018. About JSONLint. Luettu 22.2.2018. https://jsonlint.com/

jQuery. N.d. What is jQuery? Luettu 15.1.2018 http://jquery.com/

Korpela, J. K. 2008. CSS verkkosivujen muotoilussa. 1.painos. Porvoo: WS Bookwell.

Kuutti, W. 2003. Käytettävyys, suunnittelu ja arviointi. Saarijärvi: Gummerus Kirjapaino Oy.

Nielsen Norman Group. 2018. Jacob Nielsen. Luettu 18.2.2018. https://www.nngroup.com/people/jakob-nielsen/

Nielsen Norman Group. 1.1.1995. 10 Usability Heuristics for User Interface design. Luettu 18.2.2018. https://www.nngroup.com/articles/ten-usability-heuristics/

Minnick, C. 19.4.2016. The Real Benefits of the Virtual DOM in React.js. Luettu 24.3.2018. https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/

Medium. 2017. Top JavaScript Libraries & Tech to Learn in 2018. Luettu 24.3. https://medium.com/javascript-scene/top-javascript-libraries-tech-to-learn-in-2018-c38028e028e6

Mozilla Developer Network. 15.2.2018. Getting started with HTML. Luettu 14.3.2018. https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

Mozilla Developer Network. 2.1.2018. HTML. Luettu 5.1.2018. https://developer.mozilla.org/en-US/docs/Glossary/HTML

Mozilla Developer Network. 27.1.2018. HTML basics. Luettu 5.1.2018. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

Mozilla Developer Network. 16.1.2018. The Paragraph element. Luettu 15.1.2018. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p

Mozilla Developer Network. 6.12.2017. CSS basics. Luettu 7.1.2018. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics

Mozilla Developer Network. 21.2.2018. A re-introduction to JavaScript. Luettu 13.2.2018 https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

Mozilla Developer Network. N.d. What is JavaScript? Luettu 13.2.2018. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#What_is_JavaScript

Mozilla Developer Network. 28.7.2017. JSON. Luettu 17.2.2018. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Mozilla Developer Network. 10.1.2018. JSON.parse(). Luettu 13.3.2018. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Mozilla Developer Network. 12.1.2018. Using CSS animations. Luettu 17.2.2018. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

Mozilla Developer Network. 12.1.2018. Introduction to the DOM. Luettu 17.3.2018. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Mozilla Developer Network. 26.1.2018. CanvasRenderingContext2D. Luettu 18.3.2018. https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D

Peltomäki, J. & Nykänen, O. 2006. Web-selainohjelmointi. 1.painos. Porvoo: WS Bookwell.

Quackit. N.d. HTML4 !doctype Tag. Luettu 15.4.2018. https://www.quackit.com/html/html_4/tags/doctype_html_public.cfm

React. N.d. Luettu 24.3.2018. https://reactjs.org/

Refactoring Guru. N.d. What is Refactoring. Luettu 19.2.2018. https://refactoring.guru/refactoring/what-is-refactoring

Nielsen, J. 1993. Usability Engineering. United States, CA: Academic Press.

W3C. 13.11.2000. What is the Document Object Model? Luettu 17.3.2018. https://www.w3.org/TR/DOM-Level-2-Core/introduction.html

W3C. N.d. Status Code Definitions. Luettu 20.2.2018. https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

W3C. 29.12.2014. HTML structural elements. Luettu 18.3.2018. https://www.w3.org/wiki/HTML_structural_elements?utm_source=twitter-feed&utm_medium=twitter

W3Schools. N.d. Color Schemes. Luettu 7.1.2018. https://www.w3schools.com/colors/colors_schemes.asp

W3Schools. N.d. HTML5 Introduction. Luettu 15.1.2018. https://www.w3schools.com/html/html5_intro.asp

W3Schools. N.d. jQuery syntax. Luettu 20.2.2018. https://www.w3schools.com/jquery/jquery_syntax.asp

W3Schools. N.d. CSS Animations. Luettu 18.2.2018. https://www.w3schools.com/css/css3_animations.asp

W3Schools. N.d. HTML5 Semantic Elements. Luettu 14.3.2018. https://www.w3schools.com/html/html5_semantic_elements.asp

W3Schools. N.d. HTML5 Introduction. Luettu 14.3.2018. https://www.w3schools.com/html/html5_intro.asp

W3Schools. N.d. HTML canvas tag. Luettu 14.3.2018. https://www.w3schools.com/tags/tag_canvas.asp).

Wikipedia. 9.3.2018. Chromium (web browser). Luettu 14.3.2018. https://en.wikipedia.org/wiki/Chromium_(web_browser)

Wikipedia. 4.11.2017. Proseduraalinen ohjelmointi. Luettu 15.2.2018. https://fi.wikipedia.org/wiki/Proseduraalinen_ohjelmointi

Wikipedia. 9.3.2018. Rasbpian. Luettu 14.3.2018. https://en.wikipedia.org/wiki/Raspbian

XMLHttpRequest Standard. 12.1.2018. 4.4 States. Luettu 20.2.2018. https://xhr.spec.whatwg.org/#handler-xhr-onreadystatechange

LIITTEET

Liite 1. Valokuvia sovelluskehyksen suorittamisesta näyttölaitteella.