LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

NEUVOTTELUHUONEEN VARAUSKALENTERI

SPA-sovelluksen toteuttaminen Vue.js-ohjelmistokehyksellä

LAHDEN AMMATTIKORKEAKOULU
Insinööri (AMK)
Tieto- ja viestintätekniikka
Ohjelmistotekniikka
Kevät 2019
Juha Kallioniemi

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Kallioniemi, Juha	Opinnäytetyö, AMK	Kevät 2019
	Sivumäärä	
	44	

Työn nimi

Neuvotteluhuoneen varauskalenteri
SPA-sovelluksen toteuttaminen Vue.js-ohjelmistokehyksellä

Tutkinto

Tieto- ja viestintätekniikan koulutus, ohjelmistotekniikka, Insinööri (AMK)

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää reaaliaikainen ja käyttäjäystävällinen SPA-sovellus selkeyttämään neuvotteluhuoneiden varaamista ja niiden tilaa. Työn toimeksiantajana toimi ohjelmistotalo nimeltä Avenla Oy.

Työssä tutkittiin ja kehitettiin pääasiassa selainpuolta (frontend) jo olemassa olevan palvelinpuolen (backend) koodin rinnalle. Selainpuolen kehityksessä käytettiin pääasiassa LESS-tyylikieltä, TypeScript-ohjelmointikieltä ja Vue.js-ohjelmistokehystä.

Työn alkuvaiheessa sovelluksen käyttöliittymää varten annettiin valmiiksi suunniteltu pohja, joka kuitenkin muuttui työn aikana pariin kertaan toimeksiantajan pyynnöstä.

Ulkoasuun tehtävien muutoksien ja ennestään tuntemattomien teknologioiden takia työ vei enemmän aikaa kuin oletettiin. Lopputuloksena syntyi kuitenkin tavoitteita vastaava verkkosovellus, jota on hyvä jatkokehittää.

Asiasanat

LESS, JavaScript, TypeScript, Vue.js, npm, SPA-sovellus, WebSocket, Ajax, JSON

Abstract

Author(s)	Type of publication	Published
Kallioniemi, Juha	Bachelor's thesis	Spring 2019
	Number of pages	
	44	

Title of publication

Meeting room reservation calendar
Building a Single-Page Application with Vue.js framework

Name of Degree

Information and Communications Technology, Software Engineering

Abstract

The objective of this thesis was to build a real-time and user-friendly Single-Page Application to make meeting room reservations and their states clearer. The thesis was commissioned by a software house called Avenla Oy.

The main focus of the thesis was to research and develop a frontend for the already existing backend. The main tools used for the frontend development were a style language LESS, a TypeScript programming language and a Vue.js framework.

At the start of the task there was a ready-made layout for the application's user interface, which was, however, altered a couple of times by the client's request.

As a consequence of layout alterations and previously unknown technologies, the task took longer than expected. Nonetheless, the web application created in the thesis meets the goals and is easy to develop further.

Keywords

LESS, JavaScript, TypeScript, Vue.js, npm, Single-Page Application, WebSocket, Ajax, JSON

SISÄLLYS

1 JOHDANTO

Opinnäytetyössä tutkitaan pääasiassa selainpuolen kehittämistä SPA-sovelluksen yhteydessä. Tavoitteena oli kehittää reaaliaikainen ja käyttäjäystävällinen SPA-sovellus selkeyttämään neuvotteluhuoneiden varaamista ja niiden tilaa. Selainpuolen kehittämisessä pyrittiin käyttämään siihen parhaiten soveltuvia teknologioita, jotka tullaan käymään tämän opinnäytetyön teoriaosuudessa läpi. Vaikka opinnäytetyö on rajattu pääasiassa selainpuoleen, niin siinä käydään lyhyesti läpi myös taustalla toimivaa palvelinpuolen perusajatusta ja koodia.

Suuri osa nykyaikaisista verkkosovelluksista on kehitetty, modernin, SPA-arkkitehtuurin mukaisesti. Termi ”SPA” tulee sanoista Single-Page Application, joka tarkoittaa yhden sivun sovellusta. Vaikka SPA-sovelluksia käytetään selaimessa, niissä saadaan aikaan aidon sovelluksen tuntu. Moni ei varmaan tule ajatelleeksi, että tunnettuja ja jatkuvassa käytössä olevia yhden sivun sovelluksia ovat esimerkiksi

- Gmail
- Google Maps
- Facebook
- GitHub.

Kehittyneet selaimet ja modernit ohjelmistokehykset mahdollistavat SPA-sovelluksien kehittämisen ja kilpailukyvyn. Yksi varteenotettavista vaihtoehdoista SPA-sovelluksen käyttöliittymän toteuttamiseen on Vue.js-ohjelmistokehys. Se on kehitetty käyttöliittymän rakentamiseen ja on nopeuden, progressiivisuuden sekä lyhyen oppimiskäyrän ansiosta yksi suosituimmista JavaScript-ohjelmistokehyksistä tänä päivänä (Cuelogic Technologies 2018).

Verkkosovelluksen kehittämistä hankaloittaa kuitenkin se tosiasia, että kaikki ihmiset eivät automaattisesti siirry käyttämään uusimpia selaimia (Binder 2019). Eri selaimien ja niiden eri versioiden välillä on eroja, jotka voivat aiheuttaa ongelmia, jos niitä ei oteta huomioon. Perehtymällä asiaan ne eivät kuitenkaan aiheuta ylitsepääsemättömiä ongelmia.

Opinnäytetyön toimeksiantajana toimi monipuolinen ohjelmistotalo Avenla Oy, joka on jatkuvasti kehittyvä ja kasvava, sähköisen liiketoiminnan ratkaisuihin erikoistunut asiantuntijaorganisaatio. Yritys on perustettu vuonna 2005, ja sen toimitilat sijaitsevat Lahden ydinkeskustassa. Lahden tilojen lisäksi sillä on toimipisteet myös Helsingissä ja Tampereella.

2 SOVELLUKSESSA KÄYTETTYJÄ WEB-TEKNOLOGIOITA

2.1 HTML

HTML (HyperText Markup Language) on kieli, jota käytetään verkkosivun rakenteen ja sisällön määrittelyyn. HTML:n yhteydessä mainitaan usein myös verkkosivun tyyli (CSS) ja sen toiminnallisuus (JavaScript), jotka käydään yksikohtaisemmin läpi myöhemmin.

Termi ”HyperText” viittaa hyperlinkkeihin (hyperlinks), jotka yhdistävät verkkosivut toisiinsa, joko yhden tai useamman verkkosivuston välillä. Hyper tarkoittaa sitä, että linkki ei ole johdonmukainen (linear). Toisin sanoen hyperlinkistä voi periaatteessa mennä minne vain, milloin vain. Linkit ovat olennainen osa koko verkon toimintaa. (MDN Web Docs 2019a.)

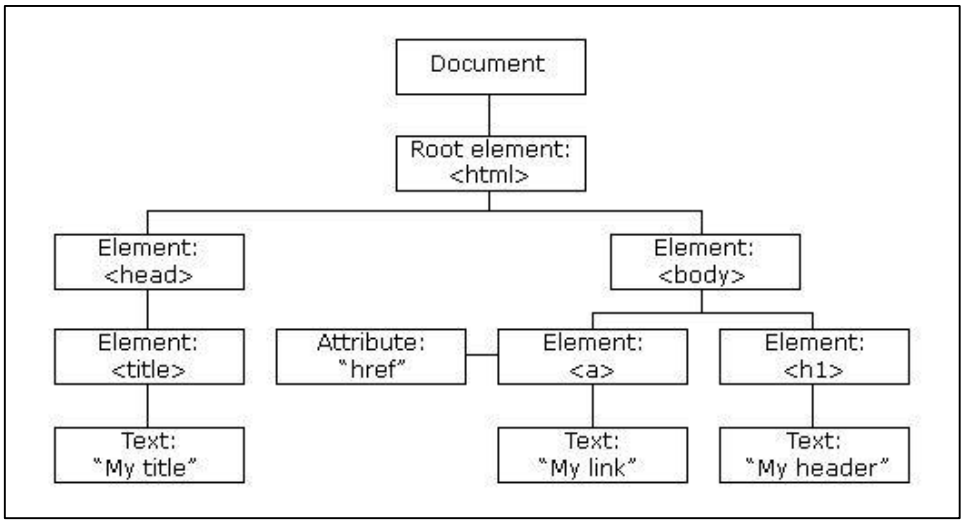
Termi ”Markup” puolestaan viittaa HTML-elementteihin (HTML elements), joilla merkitään tietyn tyyppistä sisältöä verkkosivulla. Elementti erotellaan muusta verkkosivun sisällöstä laittamalla sen ympärille ”tagit” (”< >”), esimerkiksi <title>. HTML-elementti koostuu yleensä avaavasta tag:stä <title> ja sulkevasta tag:stä </title>. Edellä mainittujen tag:ien väliin tulee kyseisen elementin sisäinen sisältö, esimerkiksi tekstiä tai muita lapsielementtejä (child elements). Kaikilla elementeillä, kuten img-elementillä (kuva 1), ei kuitenkaan ole erillistä sisäistä sisältöä. Näitä elementtejä kutsutaan tyhjiksi elementeiksi. Ne voivat sisältää attribuutteja, mutta niillä ei ole sulkevaa tag:ä, koska ne eivät vaikuta kietovasti (wrap) muuhun sisältöön. Lisämääreellä ”src” voidaan esimerkiksi määrittää upotetun (embedded) kuvan polku (path). (MDN Web Docs 2019a.)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <img src="images/firefox-icon.png" alt="My test image">
  </body>
</html>
```

Kuva 1. HTML-dokumentin rakenne (MDN Web Docs 2019a)

2.2 DOM

DOM eli dokumenttioliomalli (Document Object Model) on ohjelmointirajapinta HTML- ja XML-dokumenteille. Ohjelmointirajapinta on määritelmä, jonka mukaan eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään. Verkkosivu on dokumentti, joka voidaan näyttää selaimessa tai HTML-lähteenä. DOM määrittelee verkkosivun siten, että ohjelmat voivat muuttaa dokumentin rakennetta, tyyliä ja sisältöä. Dokumentti esitetään solmuina (nodes) ja olioina (objects). Näiden tekniikoiden avulla ohjelmointikielet, kuten JavaScript, saavat yhteyden verkkosivuun. (MDN Web Docs 2019b.)



Kuvio 1. HTML-dokumenttipuu (w3schools.com 2019a)

DOM on hierarkkinen dokumenttipuu, joka nimensä mukaisesti muistuttaa rakenteeltaan oikeaa puuta (kuvio 1). Dokumenttipuu koostuu listasta solmuja, joilla voi olla yksi tai useampi äitisolmu (parent node), lapsisolmu (child node) tai sisarsolmu (sibling node). HTML-elementti on tietyn tyyppinen solmu. Verkkosivun käsittely tapahtuu pääasiassa näiden elementtien avulla. Tietyn tyyppisiin elementteihin päästään käsiksi metodilla (method) ”getElementsByTagName()” (kuva 2), joka palauttaa listan kaikista dokumenttisolmun alta löytyvistä ankkurielementeistä (anchor elements). Elementeille voidaan myös määritellä id tai luokka (class), joka helpottaa ja selkeyttää niiden käsittelyä. Samannimisiä id:itä saa löytyä dokumenttipuusta vain yksi, mutta samannimisiä luokkia voi olla useita. Edellä mainittujen määrittelyjen jälkeen elementtien hakuun voitaisiin vaihtoehtoisesti käyttää myös metodeja: ”getElementById()” tai ”getElementsByClassName()” (kuva 2).

```
var anchorElements = document.getElementsByTagName('a');
var exampleId = document.getElementById('example-id');
var exampleClasses = document.getElementsByClassName('example-class');
```

Kuva 2. Esimerkkejä HTML-elementtien hakumetodeista

2.3 CSS

CSS (Cascading Style Sheets) on tyylikieli, jota käytetään HTML- ja XML -dokumenttien ulkoasun ja esitystavan luomiseen. CSS määrittelee kuinka elementit esitetään näytöllä. Se on yksi oleellisimmista kielistä verkkosivujen kehityksessä, jonka takia se onkin standardisoitu eri selaimien kesken.

HTML-elementtien tyylit voidaan määritellä suoraan dokumenttipuuhun style-attribuutilla (kuva 3), mutta sitä tulisi välttää mahdollisimman paljon, koska laajemmissa sivustoissa se aiheuttaa epäselvyyttä ja joustamattomuutta. Parempi ja joustavampi tapa on luoda erillinen .css-päätteinen tiedosto, johon määritellään tyylejä CSS:n omalla syntaksilla. CSS-deklaraatiot kirjoitetaan ominaisuus-arvo (property-value) pareina, jotka eritellään toisistaan kaksoispisteellä ja päätetään puolipisteeseen (kuva 4).

```
<!DOCTYPE HTML>
<html>
  <body style="font-weight:bold;">
    <div style="color:red" id="myElement">..</div>
  </body>
</html>
```

Kuva 3. HTML-elementin style-attribuutti (MDN Web Docs 2019c)

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

Kuva 4. CSS-syntaksi (MDN Web Docs 2019c)

Viimeisin versio CSS:stä on CSS3, jonka mukana tuli useita pitkään toivottuja lisäominaisuuksia, kuten

- pyöreät kulmat
- varjot
- animaatiot
- gradientit
- transitiot

- uudet layoutit (multi-column, flexbox, grid).

(MDN Web Docs 2019c.)

CSS3:sta käytettäessä tulee ottaa huomioon, että kaikkein vanhimmat yleisesti käytössä olevat selaimet eivät välttämättä vielä tue kaikkia sen uusia ominaisuuksia. Toistaiseksi vielä monet ihmiset, varsinkin työelämässä, käyttävät vanhempia selaimia. Tämän takia uusimmille CSS-ominaisuuksille olisi hyvä määritellä niin sanottuja varmistusominaisuuksia (fallback properties). CSS-tiedosto luetaan aina ylhäältä alaspäin suoraviivaisessa järjestyksessä. Kuvan 5 esimerkissä luokalle ”example-class” on määritetty sama ominaisuus kahteen kertaan, joten tässä tapauksessa taustaväriksi määritellään ensin vaaleansininen, mutta jälkimmäinen määrittely päällekirjoittaa (overwrite) sen, ja näin jälkimmäinen määrittely jää voimaan. Mikäli käytössä oleva selain ei tukisi linear-gradient-määrittelyä, käytettäisiin varmistusominaisuutta, ja taustaväriksi tulisi vaaleansininen. Jos edellä mainitussa tapauksessa varmistusominaisuutta ei olisi, taustaväriksi tulisi oletusväri, valkoinen.

```
.example-class {
    background: lightblue;
    background: linear-gradient(lightblue, lightgreen);
}
```

Kuva 5. CSS:n varmistusominaisuus

2.4 LESS

LESS (Leaner Style Sheets) on yksi monista CSS:n esikääntäjistä (preprocessors). CSS:n osaajalle LESS:n oppiminen on helppoa, koska sen syntaksi muistuttaa hyvin paljon CSS:ää. LESS-tiedosto tulee aina kääntää (compile) .css-päätteiseksi tiedostoksi, jotta selain osaa lukea sen. Kääntämisen voi suorittaa esimerkiksi Visual Studion Web Compiler lisäosan tai ”less-watch-compiler”-kirjaston avulla. Kääntämisen voi suorittaa manuaalisesti, mutta ohjelmoinnin kannalta tehokkain tapa on suorittaa LESS-tiedoston kääntäminen automaattisesti aina kun .less-päätteinen tiedosto tallennetaan tai projekti käännetään (build). Automaattisen kääntämisen voi toteuttaa esimerkiksi tarkkailutilan (watch mode) aktivoimalla suoraan komentoriviltä (command line) komennolla: ”less-watch-compiler”. Tiedoston ”compilerconfig.json” avulla kääntämisen käyttäytymistä, kuten esimerkiksi käännetyn tiedoston sijaintia, voidaan muunnella.

LESS:llä saadaan CSS:ään useita lisäominaisuuksia, jotka tekevät tyylien määrittelystä selkeämpää ja vähentävät työn määrää. LESS:n lisäominaisuuksia ovat

- muuttujat (variables)

- mixinsit
- sisäkkäisyys (nesting)
- laskutoimitukset (operations)
- satunnainen merkkijono (arbitrary string/escaping)
- funktiot
- nimiavaruudet (namespaces and accessors)
- omien ominaisuusnimien valitseminen mixinseille (maps).

LESS:n muuttujia käytetään pääasiassa useasti toistuvien arvojen tallentamiseen. Ne määritellään koodissa laittamalla niiden eteen ”@”-merkki. Muuttujat helpottavat koodin ylläpitämistä, koska useaa arvoa voidaan tällöin muuttaa yhdestä paikasta. LESS mahdollistaa myös ”$prop”-syntaksin käytön (kuva 6), jossa saman lohkon (block) sisällä olevaa ominaisuutta voidaan käyttää muuttujana. Lohko tarkoittaa jonkin tietyn aaltosulkuparin ({…}) sisään jäävää osaa. "$prop”-syntaksin hyödyntäminen tekee koodista hieman kevyemmän (lighter).

```
@myRed: #FF0000;

.example-class {
    border: 1px solid;
    border-color: @myRed;
    color: $border-color;
}
```

Kuva 6. Esimerkki LESS:n muuttujien käytöstä

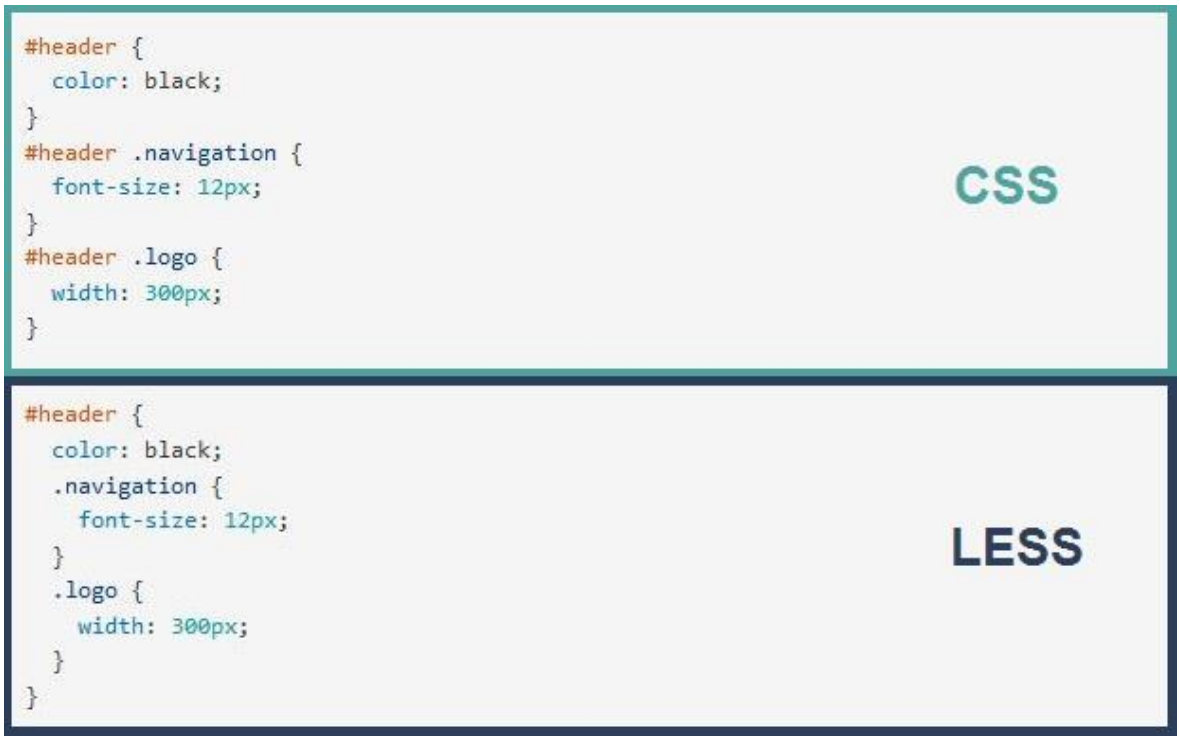
Mixinseillä voidaan lisätä esimerkiksi jonkin tietyn luokan sisältämät määrittelyt usealle eri luokalle, nopeasti ja helposti. Lohkon sisään lisätään haluttu luokka ja sen perään sulkeet, kuvan 7 ”.bordered()” mukaisesti. Edellä mainittu ominaisuus saattaa toimia ilman sulkeitakin, mutta se on vanhentumassa, joten sulkeiden käyttö on suositeltavaa. Mixinsien käytön suhteen kannattaa olla tarkkana, koska huolimaton käyttö voi johtaa tarpeettomaan koodin kopiointiin. Mixinseillä voidaan toteuttaa myös muun muassa ehdollista määrittelyä ja silmukoita (loops) avainsanaa ”when” hyödyntäen.

```
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}

#menu a {
  color: #111;
  .bordered();
}
```

Kuva 7. LESS - mixins (mukailtu lesscss.org 2019)

LESS:n sisäkkäisyys tekee koodista suppeamman ja muistuttaa enemmän HTML:n rakennetta, joka tekee siitä myös intuitiivisempaa. Kuvassa 8 on esimerkit CSS- ja LESS-tiedostoista, joiden koodit vastaavat toiminnaltaan täysin toisiaan. Kuvan 8 CSS-tiedosto on esitetty siinä muodossa, johon se on LESS-tiedostosta käännetty. Sisäkkäisten luokkien lisäksi myös esimerkiksi pseudovalitsimia (pseudo-selectors), kuten ”:hover”, voidaan syöttää sisäkkäin ”&”-operaattorin avulla.



Kuva 8. LESS - sisäkkäisyys (mukailtu lesscss.org 2019)

LESS:n aritmeettiset laskutoimitukset (yhteen-, vähennys-, kerto- ja jakolaskut) pystyvät käsittelemään mitä tahansa numeroa, muuttujaa tai väriä. LESS osaa huomioida myös laskutoimitusten yksiköt oikein, mikäli ne ovat laskuopillisesti yhteensopivia.

Mobiililaitteiden käytön yleistymisen myötä, verkkosivun ulkoasua tehdessä, on hyvä ottaa huomioon se, että sivulla tullaan vierailemaan hyvin monella eri laitteella. Erilaisten laitteiden erikokoiset selainikkunat (browser viewport) vaikuttavat siihen, miltä sivut näyttävät. LESS:n muuttujia ja satunnaista merkkijonoa hyödyntäen voidaan määritellä erilaisia ehtoja käsittelemään erikokoiset laitteet. Kuvassa 9 on esimerkki tyypillisestä pienen ikkunakoon huomioimisesta. Siinä luodaan muuttuja nimeltä ”small”, jonka arvo aloitetaan ensin aaltoviivalla (tilde), ja heti sen perään lisätään satunnainen merkkijono, joka määrittelee ehdon kyseiselle muuttujalle. Kuvan 9 alalaidassa on esimerkki ”small”-muuttujan hyödyntämisestä. Luokan ”example-class” sisään asetetaan mediakysely (media query), johon sijoitetaan muuttuja ”small” ja sen sisään yksi tai useampi LESS-deklaraatio. Tässä

tapauksessa mediakysely saa aikaan sen, että kun selainikkunan leveys on pienempi kuin 768 pikseliä, fontin kokoa kasvatetaan.

```
@small: ~"(max-width: 768px)";

.example-class {
    font-size: 1.5em;

    @media @small {
        font-size: 2.5em;
    }
}
```

Kuva 9. LESS - satunnainen merkkijono ja mediakysely

LESS tarjoaa lukuisia eri funktioita, joiden avulla voidaan muun muassa muuttaa värejä, käsitellä merkkijonoja ja laskea laskuja. Funktioilla, kuten ”if”- ja ”boolean”-funktiolla, voidaan toteuttaa myös ehdollista määrittelyä. ”if”-funktiossa määritellään ehto ja sen perään kaksi arvoa. Ehto voidaan määritellä suoraan ”if”-funktion sisään tai se voidaan esimerkiksi ensin tallentaa ”boolean”-funktion avulla muuttujaan, minkä jälkeen se voidaan sijoittaa muuttujana ehdon paikalle. ”if-funktio” palauttaa ensimmäisen arvon, jos ehto on tosi (true) ja jälkimmäisen arvon jos ehto on epätosi (false).

LESS:n nimiavaruus-ominaisuutta hyödyntäen esimerkiksi muuttujia ja mixineitä voidaan ryhmitellä erilaisiin tarkoituksenmukaisiin ryhmiin. Luotuja ryhmiä voidaan hyödyntää lähes samalla tavalla kuin tavallisia mixineitä. Jos esimerkiksi kuvan 7 ”.bordered()”-mixin ryhmiteltäisiin ”#bundle()”-lohkoon, sitä voitaisiin käyttää lisäämällä ryhmän nimi mixinin nimen eteen, eli tässä tapauksessa ”#bundle.bordered()”. Edellä mainitun ominaisuuden avulla LESS-tiedoston luettavuutta ja käyttöä voidaan selkeyttää sekä nopeuttaa, varsinkin isommissa projekteissa.

LESS:ssä nimiavaruus-ominaisuuden ja ”[]”-syntaksin yhdistämisellä mixinseille voidaan valita myös omia ominaisuusnimiä, kuten kuvan 10 ”primary” ja ”secondary” ominaisuusnimet. Edellä mainitulla ominaisuudella voidaan esimerkiksi välttää turhaa LESS:n sisäistä kommentointia, koska ominaisuuksien nimeäminen tekee niiden käyttötarkoituksesta selkeämpää.

```
#colors() {
  primary: blue;
  secondary: green;
}

.button {
  color: #colors[primary];
  border: 1px solid #colors[secondary];
}
```

Kuva 10. LESS - maps (lesscss.org 2019)

Kokonaisuudessaan LESS:n hyötyjä, verrattuna tavalliseen CSS:ään, ovat muun muassa

- nopeampi ja helpompi koodin tuottaminen
- uudelleenkäytettävyys (reusability)
- selkeämpi koodin rakenne
- selainriippumattomuus (cross-browser compatibility)
- muuttujat, mixinit, funktiot ja muut vastaavat ominaisuudet.

2.5 JavaScript (JS)

JavaScript on järjestelmäriippumaton (cross-platform) ja olioperustainen skriptikieli (scripting language). Skriptikielet eroavat ohjelmointikielistä siten, että niitä ei normaalitilanteessa tarvitse kääntää, vaan ne tulkitaan (interpret), toisin kuin ohjelmointikielet. JavaScript-koodi kirjoitetaan joko suoraan HTML-dokumenttiin tai upotetaan erillisenä tiedostona osaksi sitä, jotta selain pystyy sen tulkitsemaan. Tulkitseminen tarkoittaa sitä, että selain lukee ja suorittaa koodin samanaikaisesti. Tämä hankaloittaa osaltaan koodin debuggaamista eli virheiden etsimistä. Itse koodin kirjoittamisen kannalta tällä ei kuitenkaan ole merkittävää vaikutusta. Termien ero pohjautuukin enemmän kehitysympäristöön kuin itse kieleen.

JavaScriptin oliot perustuvat prototyyppeihin tyypillisten luokkapohjaisten olioiden sijaan. Tämän, monesta muusta ohjelmointikielestä poikkeavan, lähestymistavan takia monelle voi olla epäselvää, miten JavaScript sallii olioiden hierarkian luomisen ja käsittelee periytymisen ominaisuuksia sekä niiden arvoja. Luokkapohjaisissa kielissä luokat kirjoitetaan erillisiin luokka-määritelmiin, jossa luokalle voidaan määritellä sen rakentajametodi (constructor method). Rakentajametodin avulla voidaan luoda ilmentymiä (instance) eli olioita kyseistä luokasta. Ohjelmointikielissä luokka on määritelmä, joka määrittelee tietyn oliojoukon yhteiset piirteet. Rakentajametodissa ilmentymien ominaisuuksille voidaan määritellä oletusarvot ja suorittaa samalla luomisvaiheelle soveliasta muuta käsittelyä.

Luokan sisään voidaan määritellä rakentajametodin lisäksi myös luokan sisäisiä ominaisuuksia ja metodeja (kuva 11).

```
class Person {
    private string firstName, lastName, eyeColor;
    private int age;

    public Person(string first, string last, int age, string eyeColor) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
        this.eyeColor = eyeColor;
    }

    public string Nationality { get; set; } = "English";

    public string Name() {
        return this.firstName + " " + this.lastName;
    }
}
```

Kuva 11. Esimerkki luokkapohjaisen ohjelmointikielen (C#) Person-luokasta

JavaScript seuraa samankaltaista mallia, mutta siinä ei ole erillisiä luokkamääritelmiä. Vasta ECMAScript 6 toi mukanaan luokat JavaScriptiin, joka käydään tässä opinnäytetyössä läpi myöhemmin. Tämän takia JavaScriptissä määritellään rakentajafunktioita, joihin suoraan määritellään ominaisuudet ja oletusarvot. JavaScriptissä ei voida luoda luokan sisäisiä ominaisuuksia ja metodeja, joten ne lisätään protyyppi-ominaisuuden avulla suoraan rakentajafunktioon (kuva 12). Molemmissa tapauksissa uusia ilmentymiä voidaan luoda ”new”-operaattorin avulla. Luokkapohjaisissa kielissä ilmentymien luomisessa käytetään luokan nimeä, kun taas JavaScriptissä käytetään rakentajafunktion nimeä. (MDN Web Docs 2019e.)

```
function Person(first, last, age, eyeColor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyeColor;
}

Person.prototype.nationality = "English";

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

Kuva 12. Esimerkki JavaScriptin rakentajafunktiosta ja prototyypistä (mukailtu w3schools 2019c)

Vaikka JavaScript onkin tunnetuin verkkosivujen skriptikielenä, myös monet selaimettomat ympäristöt käyttävät sitä, kuten

- Node.js
- Adobe Acrobat
- MongoDB.

Pääte ”.js” tarkoittaa yleensä sitä, että se juontaa (derive) itsensä JavaScriptistä, kuten esimerkiksi Node.js tai Vue.js. Se voi myös tarkoittaa .js-päätteistä tiedostoa eli tekstitiedostoa, joka sisältää JavaScript-koodia.

JavaScript ajetaan verkon asiakasohjelman puolella (client side), joten sitä voidaan käyttää verkkosivujen suunnitteluun ja siihen, miten ne käyttäytyvät tapahtuman (event) esiintyessä. JavaScriptin perussyntaksi muistuttaa tarkoituksella Javaa sekä C++:aa, jotta uusia käsitteitä, kielen oppimisen kannalta, olisi mahdollisimman vähän. JavaScriptin nimestä huolimatta sillä ei ole mitään tekemistä Javan kanssa, vaikka näin moni saattaa ensin luulla. (MDN Web Docs 2019d.)

2.5.1 ECMAScript (ES)

JavaScriptin standardit perustuvat ECMAScript-standardiin. ECMAScriptistä eli ES:stä on julkaistu vuodesta 1997 lähtien yhteensä 10 eri versiota. Kesäkuusta 2015 lähtien uusia versioita on julkaistu tasan vuoden välein. ES2018 on uusin versio ECMAScriptistä ja ES2019 on jo tekeillä. JavaScriptin uusin versio on 1.8.5, mutta sen versiolla ei ole niin suurta vaikutusta verkkosivujen koodauksen kannalta, kuin ECMAScriptin versiolla. ECMAScriptin versio ja sen selaintuki ovat paljon merkitsevämmässä asemassa.

Vuonna 2009 julkaistu ES5 lisäsi JavaScriptiin ominaisuuden ”strict mode”. Sen tarkoitus on ilmoittaa epätarkasta ja huolimattomasta koodista. Strict mode tekee muun muassa seuraavia muutoksia JavaScriptin semantiikkaan:

- Eliminoi osan JavaScriptin hiljaisista (silent) virheistä, muuttaen ne merkitsevämmiksi throw-virheiksi.
- Optimoi suorituskykyä.
- Saattaa estää sellaisen syntaksin käytön, joka tullaan todennäköisesti määrittelemään tulevissa ECMAScript versioissa.

(MDN Web Docs 2019f.)

Throw-virheet nostavat virheen esiin kyseisessä lohkossa, mikä saa aikaan sen, että koodin ajo joko keskeytyy tai siirtyy catch-lauseen käsiteltäväksi. Hiljaiset virheet tulostuvat

vain selaimen konsoliin, mutta koodin ajo pyrkii jatkumaan. Strict modella ohjelmoitu koodi voidaan joissakin tapauksissa saada toimimaan tehokkaammin kuin vastaava koodi ilman strict modea. Strict mode aktivoidaan lisäämällä koodiin merkkijono: "'use strict';". Se voidaan lisätä joko skriptin alkuun, jolloin se vaikuttaa kaikkeen sen skriptin sisällä, tai sitten esimerkiksi yksittäisen funktion alkuun, jolloin se vaikuttaa pelkästään sen funktion sisäiseen koodiin.

ECMAScript 6, jonka nimi myöhemmin muutettiin nimeksi ECMAScript 2015, toi mukanaan, tähänastisista päivityksistä, ylivoimaisesti suurimmat muutokset. Sen mukana tuli useita uusia ja merkittäviä ominaisuuksia sekä syntaksi-parannuksia, kuten

- luokat ja moduulit
- iteraattorit ja for-of -silmukat
- avainsanat: let ja const
- nuolifunktiot
- promise-toiminto.

Lähes kaikki modernit selaimet tukevat jo ES2015:tä täysin, paitsi Internet Explorer (IE), jonka uusin versio IE11 tukee vain noin 11:tä prosenttia ES2015:n standardeista. Vanhemmat IE-versiot, kuten IE9, tukevat vuonna 2009 julkaistua ES5:takin vain osittain. (w3schools.com 2019b.) Internet Explorer on ollut jo pitkään jäämässä pois käytöstä ja siksi sen ECMAScript tuki on loppunut. Mikäli kehittäjät käyttävät ES2015:tä ja haluavat verkkosivujen toimivan myös vanhemmilla selaimilla, he voivat turvautua polyfilleihin. Termi polyfill tarkoittaa liitännäistä (plugin), joka tarjoaa uudempien selaimien toiminnallisuudet vanhemmille selaimille. Monet verkkosivujen kehittäjät ovat jo lopettaneet Internet Explorer -käyttäjien huomioimisen, mutta tästä huolimatta osa ihmisistä on vieläkin käyttänyt Internet Exploreria selaimenaan. Tästä syystä 8.2.2019 Microsoft pyysi julkisesti ihmisiä lopettamaan sen käytön.

ES2015:n myötä myös JavaScriptissä on voitu käyttää luokkia, mutta ne ovat pääasiassa vain syntaktista parannusta jo olemassa olevaan prototyyppipohjaan. ES2015:n luokka-syntaksi ei siis varsinaisesti tuo JavaScriptiin uutta olio-ohjelmointiin perustuvaa perimämallia, mutta se helpottaa luokkapohjaisista kielistä siirtymistä JavaScriptin pariin, sekä selkeyttää JavaScript-koodin rakennetta. Luokka-syntaksilla on kaksi eri komponenttia: luokka-deklaraatiot (class declarations) (kuva 13) ja luokka-määritelmät (class expressions) (kuva 14). JavaScript-luokka voidaan määritellä kummalla tahansa edellä mainitulla tavalla, koska käytännön eroa niillä ei ole. JavaScript-luokkaan voi määritellä rakentaja-metodin, luokan sisäisiä kenttiä (fields), metodeja, staattisia metodeja sekä get- ja set-metodeja. Luokan sisäisiä ominaisuuksia ei toistaiseksi voi määritellä, mutta lähes samaan

tarkoitukseen voidaan käyttää get- ja set-metodeja (kuva 13). Varteenotettava ero JavaScriptin perinteisemmän funktio-deklaraation ja uudemman luokka-deklaraation välillä on se, että luokka-deklaraation muuttujia ja funktioita ei laiteta muistiin ennen koodin ajoa (execution), toisin kuin funktio-deklaraation. Luokat tulee siis ensin määritellä ja sitten vasta käyttää. Edellä mainittuun, funktio-deklaraatiolle ominaiseen, muuttujien ja funktioiden tallentamiseen käytetään termiä ”hoist”. (MDN Web Docs 2019g.)

```
class Person {
    constructor(first, last, age, eyeColor) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
        this.eyeColor = eyeColor;
    }

    get nationality() {
        return this._nationality || "English";
    }

    set nationality(value) {
        this._nationality = value;
    }

    name() {
        return this.firstName + " " + this.lastName;
    }
}
```

Kuva 13. Esimerkki JavaScriptin luokka-deklaraatiosta sekä get- ja set-metodien määrittelystä

```
let Person = class {
    constructor(first, last, age, eyeColor) {···
```

Kuva 14. Esimerkki JavaScriptin luokka-määritelmän erosta luokka-deklaraatioon

Avainsanalla ”let” voi määritellä lohkon sisäisiä muuttujia, toisin kuin avainsanalla ”var”, joka määrittelee muuttujan joko globaalisti koko tiedostolle tai lokaalisti koko funktiolle. Kuvassa 15 on esimerkki avainsanojen ”var” ja ”let” eroavaisuudesta. Kuvan ylemmässä funktiossa muuttuja ”x” säilyy koko ajan samana, koska käytetään avainsanaa ”var”, siksi sen arvo lopussa on eri kuin alussa. Vaikka alemmassa funktiossa muuttujaa ”x” käsitellään samalla tavalla kuin ylemmässä funktiossa, sen arvo funktion lopussa on eri kuin alussa. Let-avainsanan ansiosta if-lauseen sisäisen lohkon x-muuttuja on eri kuin funktion alussa oleva ”x”, ja siksi funktion lopussa ”x”:n arvo on sama kuin alussa.

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

Kuva 15. Avainsana ”let” (MDN Web Docs 2019h)

Nuolifunktioilla voidaan esimerkiksi lyhentää ja selkeyttää funktioita. Normaalista funktiosta saadaan nuolifunktio poistamalla avainsana ”function” ja lisäämällä ”nuoli” (=>) parametrin tai tyhjien sulkeiden perään, riippuen tilanteesta. Mikäli funktiolla on vain yksi parametri, sen parametrin ympäriltä voidaan poistaa sulkeet kokonaan. Ja jos nuolifunktion ainoa lauseke (statement) on ”return”-lauseke, voidaan avainsana ”return” ja sen ympärillä olevat aaltosulkeet poistaa. Kuvassa 16 verrataan normaalia JavaScriptin funktiota ja supistettua nuolifunktiota (molemmat funktiot palauttavat saman taulukon). Teoriassa nuolifunktion lyhentämät funktiot vaikuttavat hyvältä asialta, mutta käytännössä se saattaa aiheuttaa päänvaivaa. Vaikka funktiot vaikuttavat visuaalisesti paljon selkeämmiltä, niin kokemattomalle JavaScript-ohjelmoijalle ne saattavat aiheuttaa vain lisää päänvaivaa. Ajan kuluessa ja nuolifunktion käytön yleistyessä niiden hyöty kuitenkin selkenee.

```
// 1. Normaali funktio.
elements.map(function(element) {
    return element.name;
});

// 2. Nuolifunktio.
elements.map(element => element.name);
```

Kuva 16. Normaalin funktion ja supistetun nuolifunktion eroavaisuus

Toinen nuolifunktion tärkeistä ominaisuuksista on tapa miten se sitoo (binds) avainsanan ”this”, tai toisin sanoen ei sido. Nuolifunktio sitoo siis avainsanan ”this” siten, että se viittaa näkyvyysalueen (scope) ylimpään kontekstiin. Jos normaalissa funktiossa halutaan viitata näkyvyysalueen ylimpään kontekstiin, funktion sisään pitää lisätä koodi: ”.bind(this)”. Nuolifunktiossa edellä mainittua koodia ei siis tarvitse erikseen lisätä. (MDN Web Docs 2019i.)

Muita merkittäviä uusia ominaisuuksia uusimmista ECMAScript versioista (ES2016 – ES2018) ovat

- potenssioperaattori (**)
- spread-syntaksi (…)
- asynkroniset funktiot.

Potenssioperaattori ajaa saman asian kuin ”Math.pow()”-funktio eli normaalin potenssilaskun. Esimerkiksi 2 ** 4 tarkoittaa samaa kuin 2 * 2 * 2 * 2. Se ei varsinaisesti tee mitään uutta ja mullistavaa, mutta nopeuttaa kuitenkin koodin tuottamista ja selkeyttää koodin rakennetta, varsinkin monimutkaisissa laskukaavoissa. Potenssioperaattoria voidaan käyttää ohjelmoimisessa täysin samalla tavalla kuin mitä tahansa muuta laskuopillista operaattoria.

Spread-syntaksilla voidaan laajentaa iteroitavia muuttujia, kuten taulukkoja tai merkkijonoja (string). Se tekee esimerkiksi taulukkojen tai olioiden keskinäisestä yhdistämisestä helpompaa (kuva 17). Sitä voidaan käyttää myös funktion parametrien määrittelyyn. Spread-syntaksia käyttämällä voidaan välttyä useilta monimutkaisilta silmukoilta ja iteroinneilta. Se käy valitun muuttujan arvot läpi järjestyksessä ja sijoittaa ne sen mukaisesti. Olioita yhdistäessä on huomioitava se, että jos jokin tunniste (identifier), kuten kuvassa 17 tunniste ”name”, on sama, viimeisimmän yhdistetyn olion arvo päällekirjoittaa edelliset arvot.

```
var arr1 = [2, 3, 4];
var arr2 = [1, ...arr1, 5];
// arr2: [1, 2, 3, 4, 5]

var obj1 = { name: 'Foo Bar', age: 45 };
var obj2 = { name: 'Foo Bar', occupation: "Programmer" };
var obj3 = { name: 'Foo Baz Bar', nationality: "English"};
var mergedObj = { ...obj1, ...obj2, ...obj3 };
// mergedObj: {name: "Foo Baz Bar", age: 45, occupation: "Programmer", nationality: "English"}
```

Kuva 17. Esimerkkejä spread-syntaksin käytöstä

Normaalisti koodia luetaan synkronisesti, eli peräkkäisessä järjestyksessä, ylhäältä alaspäin. Kuvan 18 esimerkissä synkroninen koodi tulostaa (output) merkkijonot selaimen konsoliin järjestyksessä: ”first”, ”second” ja ”third”.

```
console.log("first");
console.log("second");
console.log("third");
```

Kuva 18. Esimerkki synkronisesta koodista

Kuvan 19 esimerkissä kuvan 18 koodista on tehty asynkronista koodia kahden tuhannen millisekunnin aikakatkaisulla (timeout). Asynkroninen tarkoittaa sitä, että tietty osa koodia suoritetaan ohjelman normaalin virran (flow) ulkopuolella. Tyypillinen tilanne ilmenee esimerkiksi silloin kun tiedon latausta tarvitsee odotella. Kuvan 19 esimerkissä aikakatkaisu kuvastaa tiedon lataamista mahdollisimman yksinkertaisesti. Tässä tapauksessa asynkroninen koodi saa aikaan sen, että merkkijonot tulostuvat selaimen konsoliin järjestyksessä: ”first”, ”third” ja ”second”.

```
console.log("first");
setTimeout(() => {
    console.log("second");
}, 2000);
console.log("third");
```

Kuva 19. Esimerkki asynkronisesta koodista

ES2017-päivitys toi mukanaan asynkroniset funktiot, jotka perustuvat ES2015:n mukana tulleeseen promise-toimintoon. Asynkronisten funktioiden eli async- ja await-funktioiden tarkoitus on yksinkertaistaa promisen syntaksia ja synkronista käyttäytymistä. Asynkronisten funktioiden avulla asynkronisesta koodista voidaan tehdä ikään kuin synkronista. Tiedon lataamiseen käytetty aika ei varsinaisesti häviä minnekään, mutta koodi saadaan suoritettua siinä järjestyksessä missä se kirjoitetaan. Kuvan 20 esimerkissä on myös aikakatkaisu kuvastamassa tiedon lataamista, mutta asynkronisten funktioiden avulla konsoliin tulostuu merkkijonot järjestyksessä: ”first”, ”second” ja ”third”. Merkkijono ”third” tulostuu vasta sitten, kun merkkijono ”second” on ladattu eli kahden sekunnin kuluttua.

```
function loadData() {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve("second");
        }, 2000);
    });
}

async function dataHandler() {
  console.log("first");
  var data = await loadData();
  console.log(data);
  console.log("third");
}
```

Kuva 20. Esimerkki asynkronisesta funktiosta

Await-operoijaa käytetään promisen odottamiseen. Sitä voidaan käyttää pelkästään async-funktion sisällä. Jos asynkronisen pyynnön lataamista ei odoteta await-operoijalla, koodi suoritetaan synkronisesti, ja promise jää odotustilaan (pending). Await-operoijia voidaan luetella saman funktion sisälle useita peräjälkeen, jolloin jokaista promisea odotetaan erikseen, ja ne suoritetaan peräjälkeen. Mikäli promiseiden suorittamisesta halutaan samanaikaista, voidaan käyttää metodia ”Promise.all()”. Kyseisen metodin avulla kaikki halutut promiset voidaan suorittaa samanaikaisesti. Kun näistä viimeinen, eniten aikaa vievä, promise saadaan käsiteltyä, metodi palauttaa kaiken yhtenä promisena, ja funktion suorittaminen jatkuu siitä. (MDN Web Docs 2019j.)

2.5.2 TypeScript (TS)

TypeScript on Microsoftin kehittämä avoimen lähdekoodin (open source) ohjelmointikieli. Se ei ole kokonaan oma kielensä vaan se on rakennettu JavaScriptin päälle. Periaatteessa se on vahvasti tyypitettyä JavaScriptiä. TypeScript on siis jollain tapaa sama JavaScriptille kuin LESS on CSS:lle, koska myös TypeScript tulee aina lopuksi kääntää JavaScript-tiedostoon, jotta selain osaa lukea sen. TypeScript-projektissa kehitysympäristön juurihakemistoon (root directory) voidaan lisätä ”tsconfig.json” niminen tiedosto. Tiedoston voi lisätä manuaalisesti luomalla sen itse tai automaattisesti komennolla ”tsc --init”. Edellä mainittuun tiedostoon voidaan määritellä muun muassa kääntämisasetuksia (compiler options), mutta myös monia muita asetuksia. Mikäli tsconfig-tiedoston kääntämisasetuksia ei erikseen määritellä, käytetään niiden oletusasetuksia.

Tyypittäminen TypeScriptissä on vapaaehtoista, mutta suositeltavaa. Tyyppejä voi lisätä muun muassa muuttujille, funktioille ja ominaisuuksille. Tyyppeinä voi toimia esimerkiksi ”boolean”, ”number” tai ”string”. TypeScriptin kääntäjä (compiler) ilmoittaa virheestä, mikäli

se epäilee vääränlaista tyypin sisältöä. Tyypittäminen on siksi vapaaehtoista, että kääntäjä vain ilmoittaa mahdollisesta virheestä, mutta kääntää koodin loppuun siitä huolimatta. TypeScriptin kääntäminen tyyppivirheen esiintyessä voidaan halutessa estää muuttamalla kääntämisasetuksen ”noEmitOnError” arvo epätodesta todeksi. Ilman tyyppien määrittelyä kääntäjä ei osaa tunnistaa tyyppivirheitä, koska ilman tyypittämistä TypeScript on periaatteessa vain JavaScriptiä. JavaScriptin yksi heikkouksista on tyyppien puuttuminen, koska se lisää mahdollisia huolimattomuusvirheitä. Kuvassa 21 on yksinkertainen esimerkki tyypin määrittelystä sekä mahdollisesta huolimattomuusvirheestä, joka voi tapahtua ilman TypeScriptiä. Kuvan 21 tapauksessa TypeScriptin kääntäjä ilmoittaa virheen: ”Type 'string' is not assignable to type 'number'.”, koska muuttujan ”y” tyyppi on määritelty numeroksi, mutta se sisältää merkkijonon ”19”. Mikäli virheilmoituksesta ei välitetä, niin ”y”:n arvoksi tulee ”2019”, eikä haluttu ”39”. Monimutkaisemmassa koodissa vastaavanlainen virhe voi jäädä helpostikin huomaamatta, mikäli tyyppiä ei ole määritelty. Virheilmoitus voidaan poistaa joko muuttamalla merkkijono ”19” numeroksi tai esimerkiksi muuttamalla tyyppi ”number” tyypiksi ”string” tai ”any”, mikäli niin halutaan. Tyyppiä ”any” voidaan käyttää esimerkiksi silloin, kun tyyppiä ei tiedetä riittävän tarkasti.

```
const tsExample = document.getElementById("ts-example") as HTMLElement;

var x: number = 20 + 18;
var y: number = 20 + "19";

tsExample.innerHTML = y.toString();
```

Kuva 21. Esimerkki TypeScriptin tyyppimäärittelyistä

Kuvassa 21 käytettyä ”as”-syntaksia voidaan käyttää esimerkiksi silloin kun halutaan, että TypeScript kohtelee jotakin tietyn tyyppisesti. Tällainen tilanne voi tulla vastaan esimerkiksi silloin kun kehittäjä itse tietää arvosta enemmän kuin TypeScript. Kuvan 21 esimerkissä ”getElementById”-metodin palautustyyppi voi olla joko HTMLElement tai null (tyhjä), joten avainsanaa ”as” käyttämällä voidaan itse ilmoittaa, että tyyppi on aina HTMLElement. Vaihtoehtoisesti muuttujalle ”tsExample” voitaisiin määritellä tyyppi ”HTMLElement | null”, ja ennen käsittelyä tarkistaa, if-lauseella, onko sen arvo null vai ei. Kyseisessä tapauksessa myös tyyppi ”any” olisi mahdollista, mutta sitä tulisi välttää viimeiseen asti, koska se poistaa tyyppitarkastuksen.

Yksi TypeScriptin ydintoiminnoista (core principles) on sen sisäiset ”interface”:t (kuva 22). Ne eivät TypeScriptin tapauksessa viittaa rajapintoihin, vaan ne tarkoittavat rakenteellista tyyppiä (structural subtyping). Esimerkiksi olioille muodostetaan usein rakenteellisia tyyppejä.

```
interface LabelledValue {
    label: string;
}
```

Kuva 22. Esimerkki TypeScriptin rakenteellisesta tyypistä (Microsoft 2019a)

2.6 Node.js

Node.js on avoimen lähdekoodin ajonaikainen (run-time) kehitysympäristö JavaScript-koodin suorittamiseen selaimen ulkopuolella. Se on rakennettu Google Chromen JavaScript-moottorin (V8 Engine) päälle. Node.js on alustariippumaton, joten sitä voidaan käyttää muun muassa Microsoft Windows-, Linux- ja OS X -käyttöjärjestelmillä. Tärkeitä piirteitä, jotka tekevät Node.js:stä ensimmäisen vaihtoehdon monelle kehittäjälle ovat

- Ohjelmointi on asynkronista ja tapahtumapainotteista (event driven).
- Node.js-kirjasto (library) on tehokas.
- Kyky suoriutua useista samanaikaisista palvelinpyynnöistä tapahtumasilmukka-mekanismin (event looping mechanism) avulla.
- Node.js-sovellukset eivät koskaan puskuroi (buffer) dataa, eli varastoi sitä tilapäisesti, vaan siirtävät sen lohkoina (chunks).
- Node.js on MIT-lisenssin alainen, joka sallii kaikenlaisen käytön, kunhan alkuperäiset tekijänoikeustiedot säilytetään (Opensource.org 2019).

(TutorialsPoint 2019.)

2.6.1 Node.js package manager (npm)

Node.js package manager eli npm on yksi maailman suurimmista ohjelmistorekistereistä (software registry) tai -kirjastoista JavaScript-paketeille eli moduuleille (Medium 2018). Moduuli tarkoittaa ohjelman itsenäistä osaa, joka on muusta ohjelmasta riippumatonta. Avoimen lähdekoodin kehittäjät ympäri maailmaa käyttävät npm:ää jakaakseen ja hyödyntääkseen paketteja. Kehittäjät pääsevät npm:n paketteihin käsiksi asentamalla ensin Node.js kehitysympäristön, jonka yhteydessä asentuu myös npm. Monet organisaatiot hoitavat myös yksityisen kehittämisen npm:n avulla. Npm koostuu kolmesta erillisestä komponentista

- verkkosivusta
- komentorivin käyttöliittymästä (Command Line Interface (CLI))
- rekisteristä.

Verkkosivua voidaan käyttää esimerkiksi erilaisten pakettien etsimiseen, lisäämiseen tai profiilin luomiseen. CLI ajetaan terminaalista, joka on yleisin tapa kehittäjän ja npm:n vuorovaikutukseen. Rekisteri on laaja ja julkinen tietokanta JavaScript-ohjelmistoille. (npm, Inc. 2019.)

Komento ”npm install” on tapa, jolla komentoriviltä voidaan asentaa paketteja joko lokaalista tai globaalisti. Lokaalin ja globaalin asennuksen ero on se, että lokaalissa asennuksessa paketit asennetaan projektin sisällä kansioon nimeltä ”node_modules”, jolloin paketti toimii vain kyseisessä projektissa. Globaalissa asennuksessa komennon ”npm install” perään laitetaan optio ”-g”, joka asentaa paketin käyttöjärjestelmän tiettyyn paikkaan, jolloin paketti on käytettävissä kyseisen käyttöjärjestelmän jokaisessa projektissa. Esimerkiksi tyylikieli LESS voidaan yksinkertaisesti asentaa globaalisti komennolla: ”npm install -g less”. Tyypillisesti paketit halutaan asentaa lokaalisti, koska muuten pakettien versioita päivittäessä voi ilmetä esimerkiksi ylläpitoon tai yhteensopivuuteen liittyviä ongelmia.

2.6.2 Package.json

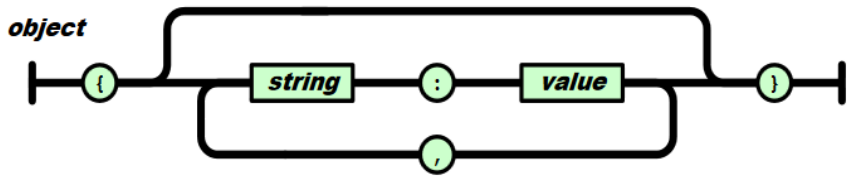
Jokainen paketti sisältää oman ”package.json” nimisen tiedoston. Myös projektin juurihakemistossa on yleensä oma package.json-tiedosto (kuva 23). Jos juurihakemiston package.json-tiedostoa ei vielä ole, sen voi lisätä joko manuaalisesti tai komentoriviltä komennolla: ”npm init -y”. Npm käyttää package.json-tiedostoa pakettien hallintaan ja asentamiseen. Se sisältää muun muassa pakettien ja projektien metatietoa. Juurihakemiston package.json-tiedoston merkittävimmät tiedot ovat sen riippuvuudet: dependencies ja devDependencies. Kun kehittäjä käyttää komentoa: ”npm install”, kaikki paketit, jotka löytyvät joko dependencies- tai devDependencies-listasta, ladataan ja asennetaan. Tämä ominaisuus on hyödyllinen silloin kun käytetään versionhallintaa ja kehittäjiä on useampi kuin yksi. Dependencies-listaan lisätään paketit, joita sovellus käyttää tuotannossa, ja devDependencies-listaan paketit, joita tarvitaan vain paikalliseen kehitykseen ja testaukseen. Npm-versiosta 5.0.0 lähtien asennetutut paketit lisätään automaattisesti dependencies-listaan. Aikaisemmissa versioissa asennuskomennon perään täytyi lisätä komento ”--save”. DevDependencies-metatietoihin paketti lisätään komennolla: ”--save-dev”.

```json
{
  "name": "example_package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "less": "^3.9.0",
    "typescript": "^3.3.3333",
    "vue": "^2.6.8"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Kuva 23. Esimerkki package.json-tiedostosta

3 JAVASCRIPT OBJECT NOTATION (JSON)

JavaScript Object Notation eli JSON on nopea tiedonvaihtoformaatti. Koneiden on helppo jäsentää (parse) ja tuottaa (generate) sitä. Se perustuu JavaScript-ohjelmointikielen osajoukkoon (subset). Se on siis tekstiformaatti, joka on täysin riippumaton ohjelmointikielestä, mutta käyttää merkintätapoja, jotka ovat luontevia suurimmalle osalle ohjelmoijista. (Ecma International 2017). JSON on syntaksi olioiden (objects), taulukoiden (array), numeroiden (numbers), merkkijonojen (strings), totuusarvomuuttujien (booleans) ja tyhjien (null) sarjallistamiselle (serialize). JSON-oliot kirjoitetaan aina avain-arvo-pareina (key-value pairs), jossa avain on aina merkkijono ja arvon tyyppi jokin edellä mainituista muuttujan tyypeistä (kuvio 2).



Kuvio 2. JSON-formaatti (Ecma International 2017)

JSON-formaattia käytetään usein silloin, kun tietoa tarvitsee esimerkiksi siirtää tai tallentaa. Vaikeaselkoiset oliot muutetaan JSON-formaatin mukaisiksi merkkijonoiksi (kuva 24), joista ne sitten tarvittaessa muutetaan takaisin olioiksi (kuva 25). Vaikka JSON-formaatti on ohjelmointikielestä riippumaton, niin sen sarjallistamiseen käytetään yleensä kielestä riippuen eri metodeja. JavaScriptissä sarjallistamiseen käytetään ”JSON.stringify()”- ja ”JSON.parse()” -metodeja, ja esimerkiksi C#:ssa voidaan käyttää Newtonsoftin: ”JsonConvert.SerializeObject()” ja ”JsonConvert.DeserializeObject<>()” -metodeja.

```
let person = {
    fullName: "Foo Bar",
    occupation: "Programmer",
    nationality: "English"
};

let json = JSON.stringify(person);
console.log(json);

// Output: {"fullName":"Foo Bar","occupation":"Programmer","nationality":"English"}
```

Kuva 24. Esimerkki olion muuttamisesta JSON-merkkijonoksi

```
let obj = JSON.parse(json);
console.log(obj.fullName);

// Output: Foo Bar
```

Kuva 25. Esimerkki JSON-merkkijonon muuttamisesta olioksi

4 SPA-ARKKITEHTUURI

4.1 SPA-sovellus

Single-Page Application eli SPA-sovellus (tai yhden sivun sovellus) tarkoittaa verkkosovellusta, joka lataa yhden HTML-sivun ja sen jälkeen dynaamisesti päivittää kyseistä sivua. Päivitykset sivulla tapahtuvat heti kun käyttäjä on vuorovaikutuksessa sovelluksen kanssa, tai esimerkiksi koodissa voidaan määritellä, että jotakin tapahtuu suhteessa kuluneeseen aikaan. SPA-sovelluksen on tarkoitus saada aikaan aidon sovelluksen tuntu. Jatkuvassa käytössä olevia yhden sivun sovelluksia ovat esimerkiksi

- Gmail
- Google Maps
- Facebook
- GitHub.

SPA-sovellukset käyttävät muun muassa Ajaxia ja HTML5:sta responsiivisen verkkosovelluksen luomiseen. SPA-sovelluksen ideana on se, että päivitetään vain sitä kohtaa jota tarvitsee päivittää, jolloin vältytään jatkuvilta sivun uudelleenlatauksilta. Tämä tarkoittaa sitä, että suuri osa kehittäjän työstä tapahtuu asiakasohjelman puolella, JavaScriptissä. Työn helpottamiseksi on luotu useita avoimen lähdekoodin JavaScript-ohjelmistokehyksiä. (Wasson 2013.) Suosittuja JavaScript-ohjelmistokehyksiä ovat esimerkiksi

- Vue
- Angular
- React
- Ember
- Meteor
- Knockout.

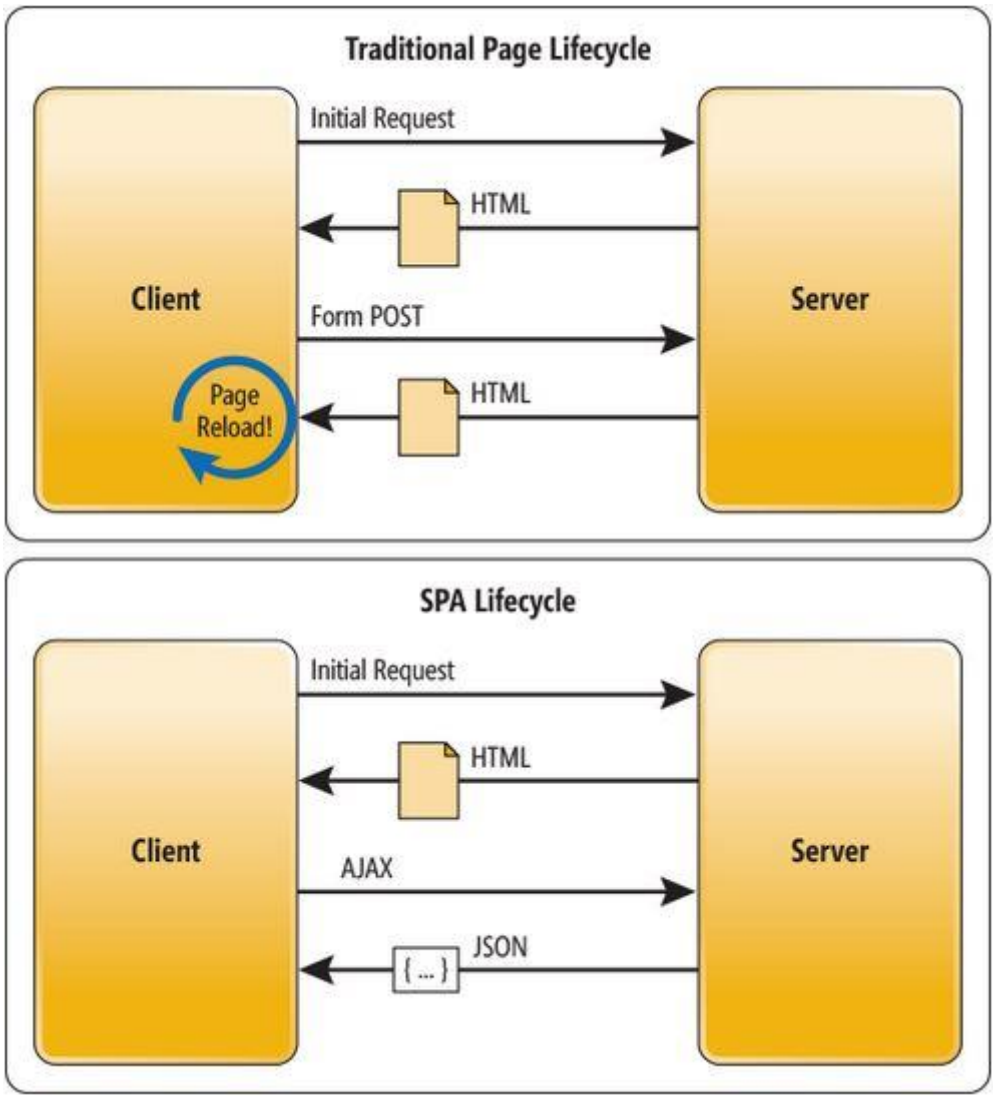
SPA-sovelluksen voi luoda myös jQuery-kirjaston avulla.

4.2 Ajax

Ajax eli ”Asynchronous JavaScript And XML” on tärkeä termi SPA-sovelluksissa. Se ei varsinaisesti ole teknologia itsessään, vaan lähestymistapa usean eri teknologian käyttämiseen yhdessä. Vaikka termin Ajax kirjain ”x” tarkoittaa XML:ää, sen käyttö ei ole pakollista, vaan voidaan käyttää myös esimerkiksi JSON:a. Ajax on tekniikka, jolla asiakasohjelma kommunikoi palvelimen kanssa asynkronisesti ilman sivun uudelleenlataamista. (MDN Web Docs 2019k.)

4.3 SPA-sovelluksen ja MPA-sovelluksen ero

Perinteisempi (traditional) tapa verkkosovelluksen luomiseen on Multi-Page Application eli MPA. Suurin ero SPA-sovelluksen ja MPA-sovelluksen välillä on se, että SPA-sovelluksessa vältytään sivujen jatkuvalta uudelleenlataamiselta. MPA-sovelluksessa, aina kun palvelin palauttaa tietoa takaisin sovellukselle, selain käynnistää sivun uudelleenlataamisen, jotta tiedot näkymässä pystytään päivittämään (kuvio 3). Suurin osa MPA-sovelluksen logiikasta suoritetaan palvelimella, kun taas SPA-sovelluksessa käyttöliittymän logiikka suoritetaan asiakasohjelmassa, joka kommunikoi palvelimen kanssa pääasiallisesti rajapintoja hyödyntäen.



Kuvio 3. SPA-sovelluksen ja perinteisen verkkosovelluksen ero (Wasson 2013)

Vaikka SPA-arkkitehtuuri saattaa ensisilmäykseltä näyttää aina paremmalta vaihtoehdolta, on MPA-arkkitehtuurille myös paikkansa. Esimerkiksi MPA-sovelluksen SEO, eli ha-

kukoneoptimointi, on parempaa kuin SPA-sovelluksen, sekä sovelluksen sisäinen navigointi on yleensä selkeämpää. MPA-sovellus on myös turvallisempi kuin SPA-sovellus, koska SPA-sovellus on haavoittuvaisempi Cross-Site Scripting (XSS) hyökkäyksiä vastaan. Cross-Site Scripting on tietoturva-aukko, joka mahdollistaa koodin syöttämisen asiakasohjelman kautta sovellukseen. Vaikka SPA-sovellus on kokonaisuudessaan nopeampi kuin MPA-sovellus, voi sovelluksen suuruus tehdä sen ensimmäisestä lataamisesta hitaan. Suurempien sovelluksien luomiseen MPA-arkkitehtuuri on vielä todennäköisesti parempi vaihtoehto.

4.4 WebSocket

WebSocket on hyvä vaihtoehto dynaamisen SPA-sovelluksen luomiseen. WebSocket-protokolla (protocol) mahdollistaa kaksisuuntaisen, viestipainotteisen kommunikoinnin asiakasohjelman ja palvelimen välillä. Tämä tarkoittaa sitä, että molemmat kommunikoivat ja vaihtavat tietoa samanaikaisesti, ja kommunikointi on reaaliaikaista (real-time). WebSocket:ien yhteydessä käytetään termiä kättely (handshake), joka kuvastaa neuvottelua, jossa varmistetaan, että palvelin ja asiakasohjelma ovat niin sanotusti yhteisymmärryksessä (sync) toistensa kanssa. Mikäli molemmat osapuolet hyväksyvät kättelyn, yhteys pyrkii pysymään auki koko sen ajan kun sovellus on käynnissä. Palvelimen täytyy seurata sen asiakasohjelmien socket:eja, jotta se ei turhaan kättele uudestaan sellaisen asiakasohjelman kanssa, joka on jo suorittanut kättelyn. Yksi WebSocket:ien hienouksista on se, että kättelyn jälkeen kumpi tahansa osapuoli voi lähettää viestin toiselle osapuolelle milloin tahansa. Yhteyden tarkistamiseen ja ylläpitämiseen voidaan käyttää esimerkiksi ”Ping ja Pong” -menetelmää. Edellä mainittu menetelmä tarkoittaa sitä, että jompikumpi osapuoli lähettää ”Ping”-viestin toiselle osapuolelle, joka vastaa takaisin ”Pong”. Mikäli vastausta ei kuulu, voidaan esimerkiksi sulkea yhteys. (MDN Web Docs 2019l.)

5 VUE.JS-OHJELMISTOKEHYS

5.1 Vue.js – Progressiivinen ohjelmistokehys

Vue.js on yksi monista JavaScript-ohjelmistokehyksistä. Vue.js on nopea ja kevyt ohjelmistokehys, ja siksi se on hyvä valinta SPA-sovelluksen toteuttamiseen. Se on Evan You:n kehittämä progressiivinen ohjelmistokehys käyttöliittymien rakentamiseen. Vue.js on suunniteltu helposti opittavaksi ja adaptoitavaksi. Se on helppo yhdistää muiden kirjastojen kanssa tai ottaa käyttöön jo olemissa olevissa projekteissa. Tästä huolimatta Vue.js kykenee suorittamaan myös monimutkaisia SPA-sovelluksia modernien työkalujen ja avustavien kirjastojen yhdistelmällä. (You 2019a.)

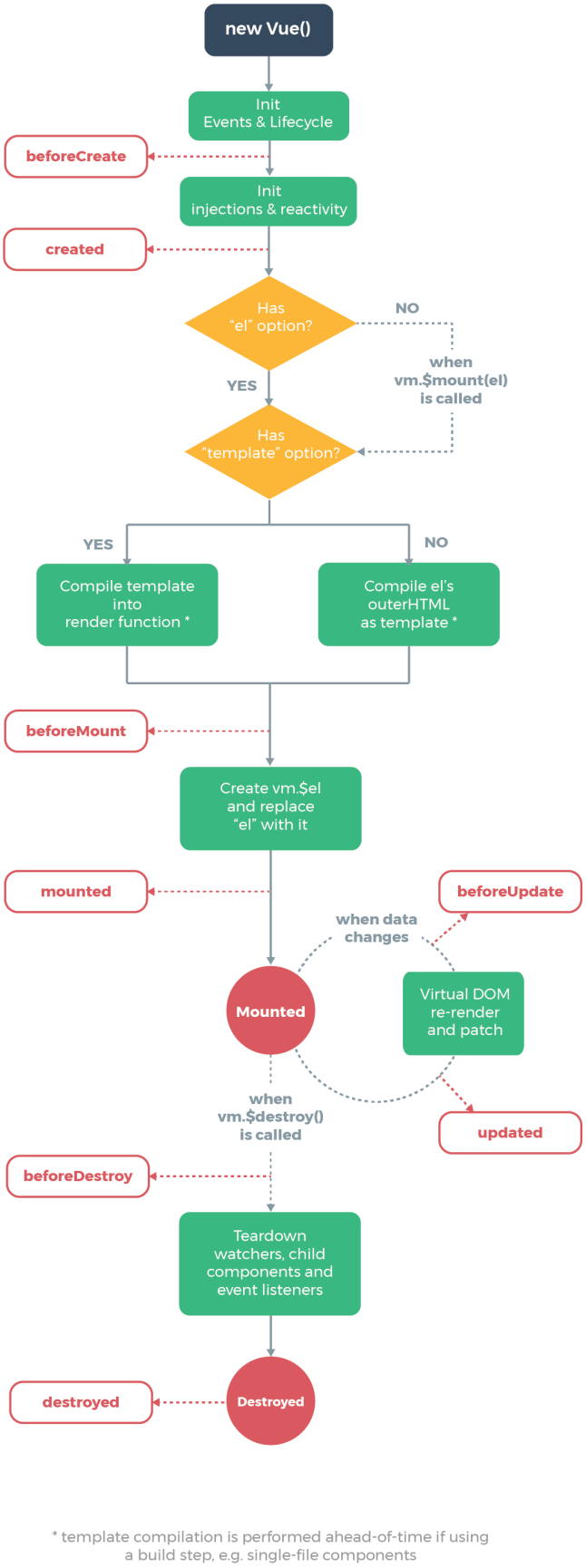
Jokainen Vue-sovellus alkaa Vue-ilmentymän luomisella. Kun Vue-ilmentymä luodaan, käydään läpi Vue.js:n sisäinen valintaolio (options object), joka sisältää listan seuraavan tyyppisistä valintalistoista

- tieto (data)
- DOM
- elinkaarikoukut (lifecycle hooks) (kuvio 4)
- resurssit (assets)
- asetelma (composition)
- sekalaiset (misc).

Valintalistat pitävät sisällään kullekin tyypille ominaisia sisäisiä valintoja. Valinnat voivat olla tyypiltään esimerkiksi olioita, merkkijonoja tai funktioita. Valintaolion sisältämillä valinnoilla saadaan aikaiseksi sovelluksen toivottu käytös. Jokainen Vue-instanssi käy luodessaan läpi sarjan erilaisia alustamisvaiheita. Elinkaarensa varrella se käy läpi myös funktiot nimeltä elinkaarikoukut, joita myös Vue-komponentit voivat käyttää. Kaikkia elinkaarikoukkuja ei ole pakko käyttää, mutta ne antavat kehittäjälle mahdollisuuden lisätä omaa koodia tiettyihin luomisen vaiheisiin. Esimerkiksi created()-funktiota kutsutaan heti kun Vue-instanssi tai -komponentti on luotu. Muita elinkaarikoukkuja ovat esimerkiksi

- mounted
- updated
- destroyed.

(You 2019b.)



Kuvio 4. Kaaviokuva Vue-ilmentymän elinkaaresta (You 2019b)

Vue.js-ohjelmistokehys käyttää HTML-pohjaista template-syntaksia (kuva 26), joka mahdollistaa sen, että kaikki Vue-templatet ovat myös validia HTML:ää. Vaikka Vue.js:stä löytyy myös modernit renderöintifunkiot sekä JSX-tuki, niin se suosii perinteisiä web-teknologioita ja rakentuu niiden päälle. Kehittäjille, joille HTML:n parissa työskentely on jo tuttua, Vue.js:n oppimiskäyrä on siis lyhyt. (You 2019c.) Template-tagien sisään voidaan määritellä HTML:ää samalla tavalla kuin normaalissa HTML-dokumentissa, mutta sinne voidaan sijoittaa myös Vue.js:n tuomia lisäominaisuuksia, kuten esimerkiksi ”viiksi”-syntaksi (”mustache”) eli tupla-aaltosulkeet ”{{ }}” (kuva 26). Viiksi-syntaksi on Vue.js:n kaikkein yksinkertaisin muoto tiedon sitomiselle (binding). Sitä käytetään tekstin sitomiseen. Kuvan 26 esimerkkikomponentissa muuttuja ”greeting” on asetettu viiksi-syntaksien sisälle, joka tarkoittaa sitä, että näkymän renderöityessä sen tilalle tulostuu muuttujan oletusarvo ”Hello”. Mikäli muuttujan arvo muuttuu ohjelman taustalla, niin sen arvo päivitetään myös näkymässä.

Hello.vue

```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Kuva 26. Esimerkki Vue-komponentista ja template-syntaksista (You 2019d)

Vue.js:ssä myös esimerkiksi luokkia ja tyylejä voidaan vaihdella HTML-elementeille dynaamisesti. Luokkien ja tyylien sitomisessa syntaksi eroaa hieman tekstien viiksi-syntaksista. Luokkien ja tyylien sitomiseen käytetään ”v-bind”-direktiiviä (kuva 27) eli toimintaohjetta. Direktiivi ”v-bind” voidaan halutessa supistaa pelkkään kaksoispisteeseen. Kuvan 27 esimerkissä div-elementille määritellään data-ominaisuuden avulla tekstin oletusväri ja -koko. Mikäli data-ominaisuutta käytetään komponentissa, niin se pitää esittää funktiomuodossa, kuten kuvassa 26. Tässäkin tapauksessa, jos esimerkiksi muuttujan ”activeColor” arvo taustalla muuttuu, myös div-elementin tekstin väri muuttuu dynaamisesti.

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

Kuva 27. Esimerkki tyylin sitomisesta Vue.js:ssä (You 2019e)

Vue-komponentit ovat uudelleenkäytettäviä Vue-ilmentymiä. Komponentin nimeksi on hyvä valita jokin niiden toimintaa vastaava nimi. Ne sisältävät yleensä templaten, tiedon ja niiden välisen logiikan. Komponentit voivat sisältää myös tyylikieltä, samalla tavalla kuin HTML-dokumentit. Uudelleenkäytettävyyden ansiosta komponentteja voidaan käyttää niin monta kertaa kuin tarvitaan. Vue-komponentteja voidaan käyttää joko niiden äitikomponenteissa tai Vue-ilmentymien sisällä. Myös Vue-ilmentymiä voi olla useita, mutta projektissa on vain yksi juuri-ilmentymä. Tietoa äitikomponentilta lapsikomponentille voidaan välittää props-ominaisuuden avulla. Ne ovat mukautettuja (custom) elementtejä ja ne esitetään koodissa samalla tavalla kuin HTML-elementit (kuva 28). Kuvan 28 esimerkissä ”components-demo” on Vue.js:n juuri-ilmentymä, jonka sisään on asetettu komponentti ”button-counter”. Vue-komponenttien riippuvuuksia seurataan automaattisesti sen renderöinnin aikana, joten järjestelmä tietää täsmälleen, mitä komponentteja tulee renderöidä uudelleen (re-render), kun tila (state) muuttuu. Tämän vähentää suorituskyvyn optimoimisen tarvetta. (You 2019c.)

HTML

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

Kuva 28. Esimerkki komponentin käytöstä Vue.js:n juuri-ilmentymässä (You 2019f)

Vue.js:n ehdollisen renderöinnin -syntaksi tekee lohkojen ehdollisesta renderöinnistä yksinkertaista ja selkeää. Ehdolliseen renderöintiin voidaan käyttää direktiivejä ”v-if”, ”v-else-if” ja ”v-else” (kuva 29). Edellä mainittujen direktiivien omaksuminen on helppoa, koska ”if”- ja ”else”-lauseiden käyttö on samantapaista kaikissa ohjelmointikielissä. Direktiiviä ”v-if” voidaan käyttää myös yksinään, jolloin HTML-elementti renderöidään vain jos ehto toteutuu. Direktiivi ”v-else” liittyy aina automaattisesti edelliseen ”v-if”- tai ”v-else-if” -lohkoon. Ehdollisen renderöinnin mukaisesti Vue.js:ssä voidaan käyttää myös listan renderöintiä. Listan renderöintiin käytetään vastaavalla tavalla ”v-for”-direktiiviä.

```HTML
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

Kuva 29. Vue.js:n ehdollinen renderöinti (You 2019g)

5.2 Vue.js dekoraattorit (decorators)

Vue-komponentin sisäisen logiikan syntaksia voidaan parantaa käyttämällä ulkoisia kirjastoja, kuten Vue Class Compoment- ja Vue Property Decorator -kirjastoa. Kirjastot voidaan asentaa projektiin esimerkiksi npm:n avulla. Vue Property Decorator -kirjasto on riippuvainen Vue Class Compoment -kirjastosta. Vue Class Compoment -kirjasto on yhteensopiva EcmaScriptin ja TypeScriptin kanssa. Kuvan 30 esimerkissä on Vue-komponentin oletussyntaksi JavaScript-koodilla. TypeScript ei tässä tapauksessa eroa JavaScriptistä sen enempää kuin normaalistikaan. Kuvan 30 esimerkissä Vue-komponenttiin on lisätty kolme valintaolion tieto-tyyppistä ominaisuutta: ”data”, ”computed” ja ”methods”. Kuvien 30 ja 31 esimerkit ovat logiikaltaan täysin samat, mutta kuvassa 31 on otettu Vue Class Component -dekoraattori käyttöön. Dekoraattorin ansioista Vue-komponentin syntaksi muistuttaa JavaScriptin luokka-syntaksia, joka vähentää tarvittavan koodin määrää ja selkeyttää sitä. Muuttujia ei tarvitse enää lisätä data-funktion sisälle, vaan ne voidaan määritellä suoraan luokan jäsenmuuttujina (member variables); computed-ominaisuudet voidaan määritellä suoraan get- ja set-metodeiksi, joka erottaa ne paremmin normaaleista funktioista eli metodeista; metodit voidaan kirjoittaa suoraan luokan sisään, eikä tarvitse käyttää methods-valintaominaisuutta.

```
export default {
    data: function() {
        return {
            message: 'Hello!'
        }
    },
    computed: {
        reversedMessage: function() {
            return this.message.split('').reverse().join('');
        }
    },
    methods: {
        changeMessage: function() {
            this.message = "Good bye!";
        }
    }
}
```

Kuva 30. Esimerkki Vue-komponentin logiikasta oletuksena

```
@Component
export default class TestComponent extends Vue {
    message = 'Hello!';

    get reversedMessage() {
        return this.message.split('').reverse().join('');
    }

    changeMessage() {
        this.message = "Good bye!"
    }
}
```

Kuva 31. Esimerkki Vue-komponentin logiikasta Vue Class Component -dekoraattorilla

Kaikille valintaolion ominaisuuksille ei löydy dekoraattoria Vue Class Compoment -kirjastosta, mutta Vue Property Decorator -kirjastosta löytyy muutama lisää, kuten ”props” (kuva 32). Mikäli props-ominaisuudet listataan oliona, niille voidaan määritellä tyypit. Vue.js:n sisältämät props-ominaisuuden tyyppimäärittelyt eroavat TypeScriptin tyyppimäärittelyistä siten, että ne alkavat isolla alkukirjaimella. Vue Property Decorator ei ole kehittäjälle yhtä intuitiivinen kuin Vue Class Component, eikä sillä ole niin merkittävää vaikutusta syntaksinkaan kannalta, joten sen tuoma hyöty on subjektiivista.

```
props: {
    id: Number,
},
```

Oletus (default)

```
@Prop({ type: Number }) id;
```

Vue Property Decorator

Kuva 32. Esimerkki Vue.js:n props-ominaisuuden eroavaisuuksista

6 NEUVOTTELUHUONEEN VARAUSKALENTERI

6.1 Todennukset ja oikeudet

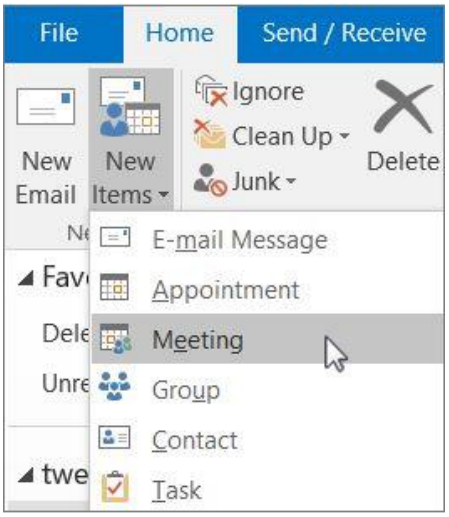
Varauskalenteri käyttää taustalla Outlook Calendar REST -rajapintaa, joka on Outlook REST -rajapinnan osajoukko (subset). Outlook REST -rajapintoihin on saatavilla kaksi eri päätepistettä (endpoint): Outlook API- ja Microsoft Graph -päätepisteet. Toiminnallisuus edellä mainituissa päätepisteissä on pääpiirteittäin sama. Microsoft suosittelee käyttämään Microsoft Graph -päätepistettä aina kun mahdollista, ja koska sen sisältämä toiminnallisuus on tähän sovellukseen riittävä, päädyttiin käyttämään juuri sen päätepistettä ”https://graph.microsoft.com/”. (Microsoft 2019b.)

Varauskalenteri-sovelluksen käyttämä Microsoft Graph turvautuu todennuksissa (authentication) ja oikeuksissa (authorization) Microsoft Azure AD -portaalin (Active Directory) valtuuksiin (tokens). SPA-sovellus täytyy rekisteröidä Azure AD:ssa, Microsoft Graph -käyttöoikeuksien saamiseksi. Rekisteröinnin jälkeen saadaan tietoon tenant (haltija) ID, client ID ja client secret, joita tarvitaan todennuksessa. Todennukseen käytetään OAuth 2.0 -protokollaa. (Kokkinen 2017.)

Neuvotteluhuoneet lisätään Azure AD:ssa rekisteröidyn haltijan alle käyttäjinä. Microsoft Graph:in oikeuksien avulla sovellus saa pääsyn haltijan alle lisättyjen käyttäjien, eli neuvotteluhuoneiden, Outlook-kalentereihin.

6.2 Neuvotteluhuoneen varauksien teko ja hallinta

Neuvotteluhuoneiden varaukseen suunnitellussa SPA-sovelluksessa huoneiden varaukset tehdään toistaiseksi vielä Microsoft Outlook -sähköpostisovelluksen kautta. Tulevaisuudessa varaukset olisi tarkoitus pystyä hoitamaan myös suoraan SPA-sovelluksen käyttöliittymästä, esimerkiksi neuvotteluhuoneiden oven lähettyvillä olevista tableteista. Microsoft Outlookissa on oma sisäinen järjestelmä neuvotteluiden varaamiselle. Neuvotteluhuoneen varaaminen Outlookissa tapahtuu samalla periaatteella kuin palaverin (meeting) varaaminen, mutta varauksen vastaanottajiin lisätään neuvotteluun osallistuvien henkilöiden lisäksi myös neuvotteluhuoneen postilaatikon (mailbox) osoite. Varaaminen aloitetaan ensin kirjautumalla sisään Outlook-sähköpostisovellukseen. Outlookin sisäisestä kalenterivälilehdestä voidaan luoda uusi tehtävä (item) ”Meeting” (kuva 33).

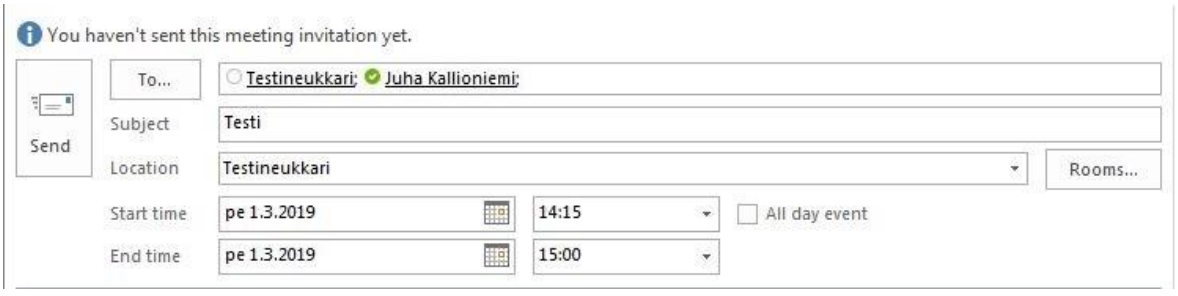


Kuva 33. Uuden neuvottelun luominen Outlook-sähköpostipalvelussa (Microsoft 2019c)

Neuvotteluhuoneen varauksen seuraavassa vaiheessa varaukseen lisätään alla olevat tiedot (kuva 34)

- neuvotteluhuoneen osoite
- kutsuttujen henkilöiden sähköpostiosoitteet
- aihe
- sijainti
- aloitus- ja lopetusaika.

Neuvotteluhuoneen osoite lisätään vastaanottajiin automaattisesti kun se valitaan ”Rooms…”-painikkeesta aukeavasta listasta. Varaus vahvistetaan painamalla lähetä-painiketta (send button). Tehty varaus voidaan myös perua lähettämällä peruutusviesti samoihin osoitteisiin kuin alkuperäinen varaus. Varaukseen lisätyt osoitteet muodostuvat tässä tapauksessa vastaanottajien kenttään automaattisesti, joten varauksen peruminen on helppoa ja nopeaa.



Kuva 34. Esimerkki neuvotteluhuoneen varaamiseen liittyvien tietojen lisäämisestä

Varauskalenterin palvelinpuoli on kirjoitettu pääasiassa C#-ohjelmointikielellä. Varauskalenteri-sovelluksessa kalenteritapahtumien (calendar events) pyyntöihin (request) käytetään asynkronista GetCalendarEvents()-funktiota (kuva 35). Pyynnöt tehdään Microsoft Graph -päätepisteeseen, jonka polkuun lisätään päätepisteen versio, käyttäjä eli neuvotteluhuone, calendarView-prefiksi ja tarvittavat parametrit, kuten esimerkiksi aloitus- ja lopetuspäivä. Edellä mainittu funktio palauttaa vastauksen (response) listana CalendarEvent-tyyppisiä olioita kaikista neuvotteluhuoneen varauksista, jotka täyttävät parametrien ehdot. CalendarEvent-olio sisältää muun muassa

- kalenterin ID:n
- aiheen
- osallistujien nimet ja määrän
- aloitus- ja lopetusajan
- järjestäjän nimen.

```
public async Task<List<CalendarEvent>> GetCalendarEvents(string user, DateTime start, DateTime end)
{
    var client = new HttpClient()
    {
        BaseAddress = new Uri("https://graph.microsoft.com/v1.0/"),
        DefaultRequestHeaders = { Authorization = GetMicrosoftGraphAuthorization() },
    };

    string qs = $"startDateTime={start:s}&endDateTime={end:s}&$select=subject,sensitivity,isAllDay,start,end,location&$top=1000";

    var response = client.GetAsync($"users/{user}/calendarView?{qs}").Result;
```

Kuva 35. Kalenteritapahtumien haku Microsoft Graph -päätepisteestä

Tenant ID:n ja neuvotteluhuoneen ID:n perusteella tietokantaan voidaan lisätä neuvotteluhuoneelle lisätietoa, kuten esimerkiksi väri ja otsikko. Lisätietoa voidaan hyödyntää esimerkiksi käyttöliittymän ulkoasussa. Tiedot varauksista lisäarvoineen lähetetään asiakasohjelmalle WebSocketin avulla.

6.3 Client-komponentti

SPA-sovelluksen WebSocketin asiakasohjelman puoleisesta toiminnasta vastaa TypeScript-tyyppinen Vue-komponentti nimeltä ”Client” (kuva 36). Client-komponentilla ei ole ollenkaan templatea vaan sen tehtävänä on hoitaa palvelimen kanssa viestittely WebSocketin välityksellä, ja viedä (export) palvelimelta saapuneet viestit oikeassa muodossa Vue.js:n juurikomponentille. Kun juurikomponentille tuodaan (import) client-komponentin sisältö, se aloittaa, yhdessä lapsikomponenttien kanssa, muodostamaan käyttöliittymän ulkoasua client-komponentin tuoman sisällön mukaisesti.

```
@Component
class Client extends Vue {
    connected: boolean = false;
    deviceName: string = null;
    deviceKey: string = null;
    calendars: CalendarInfo[] = null;

    onError: (err: string) => any = null;

    private sock: any;
    private pingInterval = 60 * 1000;
    private pingIntervalHandle: number;

    created() {
        const url = (window.location.protocol == "https:" ? "wss:" : "ws:")
            + "//" + window.location.host + "/api/websocket";
```

Kuva 36. TypeScript-tyyppinen Client-komponentti

Client-komponentti välittää neljän tyyppisiä viestejä (kuva 37)

- AuthenticateMessage
- CalendarEventsMessage
- DeviceInfoMessage
- PingMessage.

AuhenticateMessage sisältää varmistuksen onnistuneesta tai epäonnistuneesta yhteyden muodostumisesta. Yhteys epäonnistuu esimerkiksi silloin kun haettua laiteavainta (deviceKey) ei löydy. Laiteavain syötetään URL-parametriin, josta sitä verrataan tietokannassa oleviin laiteavaimiin. CalendarEventsMessage sisältää tietylle kalenterin ID:lle tehdyt varaukset. Jos varauksia ei ole tehty, events-lista on tyhjä, jolloin käyttöliittymä renderöidään sen mukaisesti. DeviceInfoMessage sisältää laitteen nimen, ping-aikavälin (pingInterval) ja listan tietokantaan lisätyistä lisätiedoista (CalendarInfo[]). Ping-aikaväli tarkoittaa aikaväliä, kuinka usein laitteen tiedot päivitetään automaattisesti. Laitteen lisätiedot sisältävät tällä hetkellä kalenterin ID:n, värin ja otsikon. Kalenterin ID tarkoittaa tietyn neuvotteluhuoneen alle tehtyjä varauksia, kun taas laitteen ID tarkoittaa laitetta, jossa kalenterit näytetään. DeviceInfoMessagen ping-aikaväli eroaa PingMessage:sta siten, että PingMessage:a käytetään tiheämmin ja sen tarkoitus on varmistaa yhteyden toimivuus. PingMessage perustuu WebSocketille tyypilliseen ”Ping ja Pong” -menetelmään. WebSocket-viestien liikennettä voi tarkastella esimerkiksi Google Chrome -selaimen kehittäjätyökalujen (developer tools) verkko-välilehdeltä (network tab), ja valitsemalla sieltä WS-suodattimen (filter) (kuva 38).

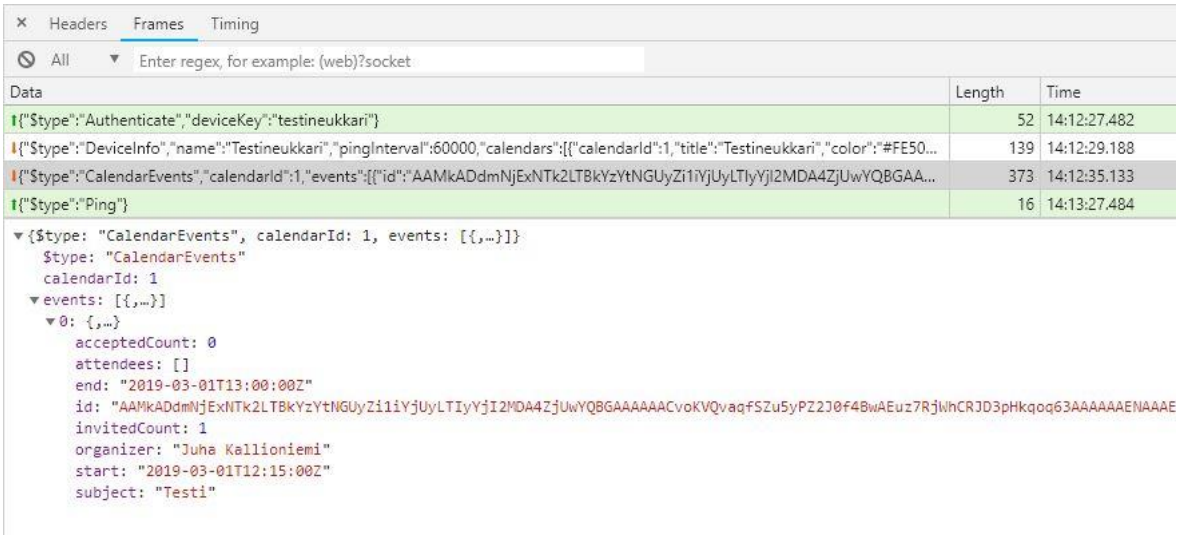
```
export default new Client();

interface CalendarEventsMessage {
    $type: "CalendarEvents";
    calendarId: number;
    events: CalendarEvent[];
}

interface DeviceInfoMessage {
    $type: "DeviceInfo";
    name: string;
    pingInterval: number;
    calendars: CalendarInfo[]
}

interface PingMessage {
    $type: "Ping";
}
```

Kuva 37. Client-komponentin vastaanottamien rakenteellisten viestityyppien määrittelyjä



Kuva 38. Esimerkki varauskalenterin WebSocket-viesteistä Google Chromen kehittäjätyökaluissa

6.4 Vue-komponenttien rakenne varauskalenteri-sovelluksessa

Varauskalenterissa Vue-komponenttien rakenne lähtee liikkeelle ”OtRoot.vue”-nimisestä juurikomponentista. Juurikomponentissa käyttöliittymän ulkoasu määritellään sen mukaisesti, kuinka monta kalenteria kyseiseen laitteeseen on linkitetty (kuva 39). Juurikomponentti haarautuu useampaan lapsikomponenttiin. Se käy läpi löydetyt kalenterit silmukassa, josta se välittää laitteen kalenterit props-ominaisuudella lapsikomponentteihin. Juurikomponentti on suhteellisen yksinkertainen ja suppea komponentti, koska sen tehtävä

on pääasiassa vain delegoida tehtäviä lapsikomponenteille. Juurikomponentti sisältää myös jonkin verran logiikkaa, mutta verrattain vähän suhteessa lapsikomponentteihin.

```
<template v-if="client.calendars">
    <div :class="'calendar-wrapper-' + client.calendars.length">
        <ot-calendar v-for="(calendar, index) in client.calendars" :key="index"
                     :control-lights="index == 0"
                     :calendar="calendar" />
    </div>
</template>
```

Kuva 39. Varauskalenterin Vue-juurikomponentin template

Calendar-props:in avulla lapsikomponentit pääsevät käsiksi kaikkiin tarvittaviin kalenterin tietoihin, kuten esimerkiksi ”calendar.events”-listaan. Events-listasta saadaan varauksen sisältävät tiedot, joiden avulla esimerkiksi lapsikomponentti ”OtEventList.vue” luo varauksiin perustuvan varauslistanäkymän (kuva 40). OtEventList-komponentti käy Vue.js:n ”for”-silmukalla läpi kaikki laitteen sen päiväiset varaukset järjestäjä- ja tuntikohtaisesti. Järjestäjän tekemien varauksien perusteella taulukon kenttiin lisätään ”div”-elementti. Luotujen elementtien sijainti ja leveys määritellään ”getOrganizerReservations()”-funktiossa, OtEventList-komponentin logiikkaosiossa.

```
<tbody>
    <tr v-for="organizer in organizers" :key="organizer">
        <td class="organizer">
            <i class="icon-man"></i>
            <span class="organizer-name">{{ getFirstName(organizer) }}</span>
        </td>
        <td v-for="hour in hours" :key="hour">
            <div v-for="(res, index) in getOrganizerReservations(organizer, hour)" :key="index"
                 class="reservation"
                 :style="{
                     backgroundColor: calendar.color,
                     borderColor: calendar.color,
                     left: res.startPercent + '%',
                     width: res.lengthPercent + '%'
                 }" />
        </td>
    </tr>
</tbody>
```

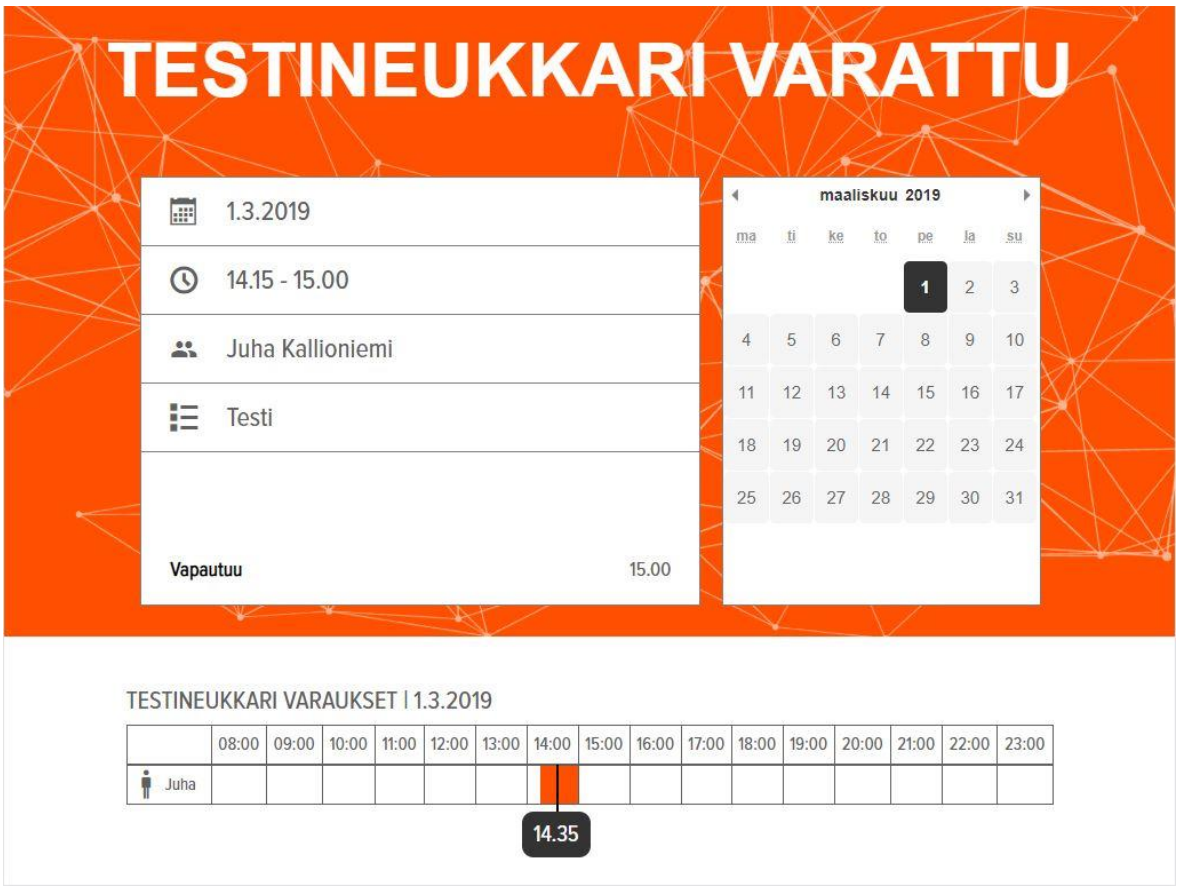
Kuva 40. OtEventList.vue-komponentin varauslistanäkymän template

6.5 Käyttöliittymä (UI)

Käyttöliittymä eli user interface (UI) on se osa sovelluksesta, joka näkyy itse sovelluksen käyttäjälle. Käyttöliittymän avulla käyttäjä pystyy olemaan vuorovaikutuksessa sovelluksen kanssa, esimerkiksi visuaalisesti tai toiminnallisesti. Varauskalenterin käyttöliittymä muodostuu kolmesta lapsikomponentista (kuva 41)

- OtEventInfo.vue

- OtPikaday.vue
- OtEventList.vue.

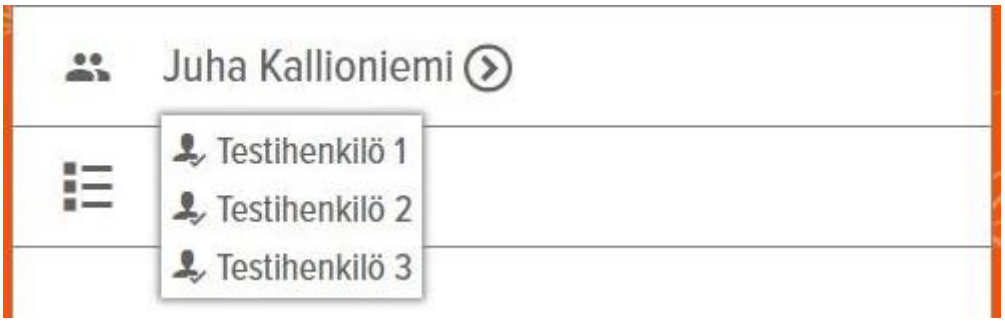


Kuva 41. Neuvotteluhuoneen varauskalenterin käyttöliittymä

OtEventInfo-komponentti muodostuu käynnissä olevan palaverin tiedoista. Se renderöidään kalenterinäkymän vasempaan ylälaitaan (kuva 41). Siinä listataan järjestyksessä ylhäältä alaspäin

- päivämäärä
- palaverin ajankohta
- järjestäjä (+ osallistujat)
- aihe
- neuvotteluhuoneen vapautumisen ajankohta.

Palaveriin osallistuvien nimet saadaan näkyviin järjestäjän oikealla puolella olevasta painikkeesta (kuva 42). Kuvan 41 esimerkki-varauksessa ei ole muita osallistujia järjestäjän lisäksi, jonka takia osallistujien listaa ei ole näkyvillä.



Kuva 42. Esimerkki palaveriin osallistuvien listasta

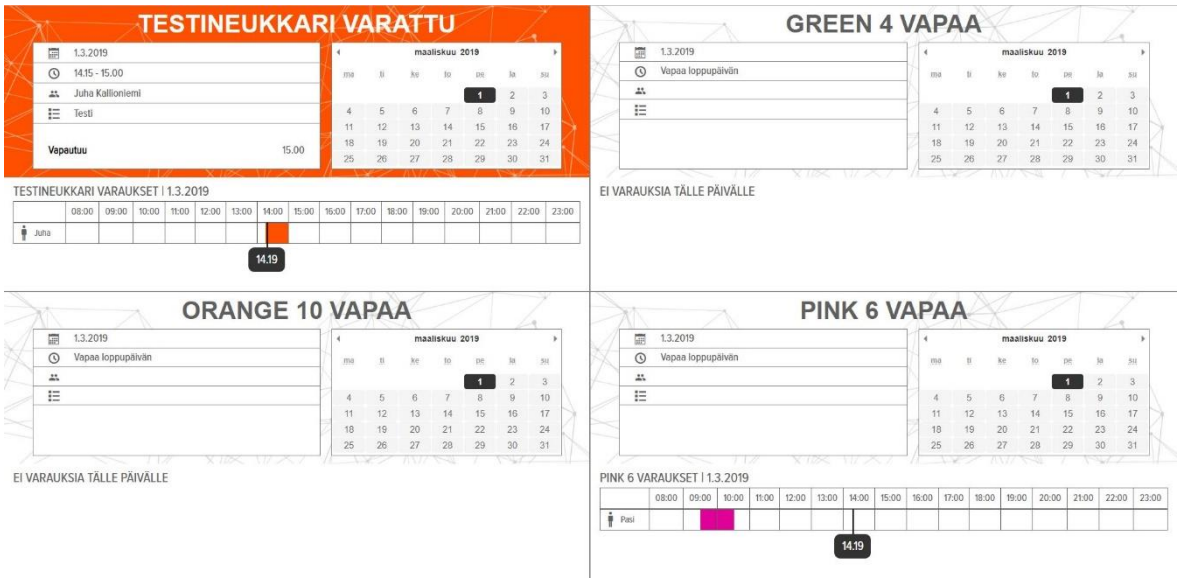
Mikäli neuvotteluhuoneessa ei ole käynnissä palaveria, ajankohta-kentässä näkyy seuraavan palaverin alkamisaika, järjestäjä- ja aihekenttä jäävät tyhjäksi ja kalenterinäkymän tausta vaihtuu, kalenterille määritellystä väristä, valkoiseksi. Mikäli neuvotteluhuone on loppupäivän vapaa, se ilmoitetaan ajankohta-kentässä tekstillä: ”Vapaa loppupäivän”.

Pikaday-komponentti on valmis JavaScript-kirjasto päivän valitsemiseen. Se renderöidään käyttöliittymässä OtEventInfo-komponentin oikealle puolelle (kuva 41). Toistaiseksi sillä on vain visuaalinen merkitys, mutta tulevaisuudessa sen kautta olisi tarkoitus pystyä selaamaan eri päiviä ja tekemään niihin varauksia suoraan käyttöliittymästä.

OtEventList-komponentti renderöi listan neuvotteluhuoneeseen tehdyistä varauksista kalenterinäkymän alalaitaan (kuva 41). Mikäli palaverin järjestäjällä (organizer) eli neuvotteluhuoneen varaajalla on samalle päivälle, samaan neuvotteluhuoneeseen enemmän kuin yksi varaus, niin se listataan samalle riville kuin muutkin sen järjestän varaukset. Eri järjestäjien varaukset listataan omille riveilleen.

6.6 Usean kalenterin näkymä

Samalle laitteelle voidaan asettaa useampikin kalenteri, jotta samalla näytöllä voidaan tarvittaessa näyttää usean neuvotteluhuoneen tila samanaikaisesti. Neljän kalenterin samanaikainen näkymä on tällä hetkellä maksimimäärä yhdelle näytölle (kuva 43). Varauskalenteriin on tehty erilaiset ulkoasut yhden, kahden, kolmen ja neljän kalenterin yhdenaikaiselle näkymälle. Tabletin kokoisella näytöllä on suositeltavaa näyttää maksimissaan vain yksi kalenteri kerralla, jotta ulkoasusta ei tule liian kompakti.



Kuva 43. Varauskalenterin neljän kalenterin näkymä

7 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää reaaliaikainen ja käyttäjäystävällinen SPA-sovellus selkeyttämään neuvotteluhuoneiden varaamista ja niiden tilaa. Käyttöliittymän ulkoasuun tehtävien muutoksien ja ennestään tuntemattomien teknologioiden takia työ vei enemmän aikaa kuin alun perin oli ajateltu. Lopputuloksena syntyi kuitenkin tavoitteita vastaava verkkosovellus, jota on hyvä jatkokehittää. Sovellus on yhteensopiva kaikkien yleisimpien selaimien kanssa. Työn aikaiset ohjelmointitaidot ja itselle tuntemattomat teknologiat huomioon ottaen, työ oli mielestäni onnistunut. Käytetyt teknologiat ja arkkitehtuurit ovat kaikesta huolimatta tuntuneet onnistuneilta valinnoilta.

Sovellus on ollut jo jonkin aikaa käytössä Avenlan toimitiloissa Lahdessa ja se on toiminut hyvin. Toimistolle sisään tultaessa on iso näyttö, jossa sovellus pyörii. Vierailevilta asiakkailta on tullut positiivista palautetta sovelluksen ensivaikutelmasta ja sen ideasta.

Tulevaisuudessa neuvotteluhuoneiden varaukset olisi tarkoitus pystyä hoitamaan myös suoraan SPA-sovelluksen käyttöliittymästä, joka aiheuttaa varmasti jatkotutkimushaasteita niin selainpuolelta kuin palvelinpuoleltakin. Vaikka sovellus onkin jo käytössä, niin kokonaisuutena se tuntuu mielestäni hieman keskeneräiseltä, ennen kuin tämä ominaisuus saadaan toteutettua. Mikäli sovellus aiotaan tuotteistaa tulevaisuudessa, niin se voi aiheuttaa lisää tutkimushaasteita. Voiko asiakas lisätä neuvotteluhuoneita varauskalenteriin vai tarvitaanko ylläpitoa?

LÄHTEET

Binder, M. 2019. Microsoft cybersecurity expert: Please, stop using Internet Explorer as a web browser [viitattu 11.4.2019]. Saatavissa: https://mashable.com/article/microsoft-stop-using-internet-explorer-browser/?europe=true#jEYdIRdp8mqs

Cuelogic Technologies 2018. Top 3 Best JavaScript Frameworks for 2019 [viitattu 11.4.2019]. Saatavissa: https://medium.com/cuelogic-technologies/top-3-best-javascript-frameworks-for-2019-3e6d21eff3d0

Ecma International 2017. The JSON Data Interchange Syntax [viitattu 13.3.2019]. Saatavissa: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Kokkinen, L. 2017. How to set up an Azure AD application registration for calling Microsoft Graph [viitattu 22.3.2019]. Saatavissa: https://laurakokkarinen.com/how-to-set-up-an-azure-ad-application-registration-for-calling-microsoft-graph/#why-azure-ad-v1.0-endpoint

lesscss.org 2019. It's CSS, with just a little more [viitattu 24.2.2019]. Saatavissa: http://lesscss.org/

MDN Web Docs 2019a. HTML: HyperText Markup Language [viitattu 17.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTML

MDN Web Docs 2019b. Introduction to the DOM [viitattu 12.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

MDN Web Docs 2019c. CSS: Cascading Style Sheets [viitattu 17.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/CSS

MDN Web Docs 2019d. JavaScript [viitattu 26.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript

MDN Web Docs 2019e. Details of the object model [viitattu 27.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model#Class-Based_vs._Prototype-Based_Languages

MDN Web Docs 2019f. Strict mode [viitattu 4.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

MDN Web Docs 2019g. Classes [viitattu 28.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

MDN Web Docs 2019h. let [viitattu 4.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

MDN Web Docs 2019i. Arrow functions [viitattu 28.2.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

MDN Web Docs 2019j. Promise [viitattu 6.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

MDN Web Docs 2019k. Ajax [viitattu 13.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

MDN Web Docs 2019l. Writing WebSocket servers [viitattu 14.3.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

Medium 2018. This year in JavaScript: 2018 in review and npm's predictions for 2019 [viitattu 14.4.2019]. Saatavissa: https://medium.com/npm-inc/this-year-in-javascript-2018-in-review-and-npms-predictions-for-2019-3a3d7e5298ef

Microsoft 2019a. Interfaces [viitattu 23.3.2019]. Saatavissa: https://www.typescriptlang.org/docs/handbook/interfaces.html

Microsoft 2019b. Compare Microsoft Graph and Outlook REST API endpoints [viitattu 22.3.2019]. Saatavissa: https://docs.microsoft.com/en-us/outlook/rest/compare-graph

Microsoft 2019c. Schedule a meeting with other people [viitattu 21.3.2019]. Saatavissa: https://support.office.com/en-us/article/schedule-a-meeting-with-other-people-5c9877bc-ab91-4a7c-99fb-b0b68d7ea94f

npm, Inc. 2019. About npm [viitattu 10.3.2019]. Saatavissa: https://docs.npmjs.com/about-npm/

Opensource.org 2019. The MIT License [viitattu 10.3.2019]. Saatavissa: https://opensource.org/licenses/mit-license.html

TutorialsPoint 2019. Node.js – Quick Guide [viitattu 10.3.2019]. Saatavissa: https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm

w3schools.com 2019a. JavaScript HTML DOM [viitattu 12.2.2019]. Saatavissa: https://www.w3schools.com/js/js_htmldom.asp

w3schools.com 2019b. JavaScript Versions [viitattu 27.2.2019]. Saatavissa: https://www.w3schools.com/js/js_versions.asp

w3schools.com 2019c. JavaScript Object Prototypes [viitattu 27.2.2019]. Saatavissa: https://www.w3schools.com/js/js_object_prototypes.asp

Wasson, M. 2013. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. Microsoft Developer Network [viitattu 13.3.2019]. Saatavissa: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

You, E. 2019a. Introduction. Vue.js [viitattu 17.3.2019]. Saatavissa: https://vuejs.org/v2/guide/

You, E. 2019b. The Vue Instance. Vue.js [viitattu 19.3.2019]. Saatavissa: https://vuejs.org/v2/guide/instance.html

You, E. 2019c. Comparison with Other Frameworks. Vue.js [viitattu 17.3.2019]. Saatavissa: https://vuejs.org/v2/guide/comparison.html

You, E. 2019d. Single File Components. Vue.js [viitattu 19.3.2019]. Saatavissa: https://vuejs.org/v2/guide/single-file-components.html

You, E. 2019e. Class and Style Bindings. Vue.js [viitattu 19.3.2019]. Saatavissa: https://vuejs.org/v2/guide/class-and-style.html

You, E. 2019f. Component Basics. Vue.js [viitattu 17.3.2019]. Saatavissa: https://vuejs.org/v2/guide/components.html

You, E. 2019g. Conditional Rendering. Vue.js [viitattu 19.3.2019]. Saatavissa: https://vuejs.org/v2/guide/conditional.html