Front-End nõuded

versioon 1, jaanuar 2019
https://dok.rik.ee/x/viWcAg

Sisukord

Sissejuhatus

Tehnoloogia areng on tekitanud olukorra, kus pakutavate tehnoloogiate virr-varrist on raske leida asjakohaseid suunised, mille järgi juhinduda. Eriti peegeldub see Front-End tehnoloogiate rakendamisel, kus loodavate lahenduste kvaliteet on seinast-seina. Ühest küljest on sellise olukorra tekkimist võimaldanud asjaolu, et veebilehe arendamisel kasutatavad tehnoloogiad on vigade tolerantsed. See tähendab seda, et HTML-is või CSS-is esinevate vigade kohta ei anna veebilehitsejad vaikimisi arendajatele tagasisidet. Kui programmeerimiskeele interpreteerimisel tekib viga, siis veasituatsiooni põhjustatud kohas tegevus lõpetatakse ja selle katkestamise põhjus kirjutatakse kas ekraanile või mõnda faili. Siis HTML/CSS-i korral see nii ei ole. HTML/CSS-is esinevad vead saab tuvastada vaid kaudselt, kuna veebilehitseja tootjad on loonud enda poolt loodud veebilehitsejasse tööriistad, mis parendavad või ignoreerivad vigast koodi. Seega on võimalik kirjutada vigadega koodi, aga kasutaja neid vigu veebilehel ei näe.
Käesolev dokument käsitleb endas juhiseid ja suuniseid, mida tuleb järgida veebilehtede programmeerimisel. Käsitlevateks tehnoloogiateks on HTML5, CSS3 ja Javascript ehk see osa, mis toimub veebilehitsejas (browseris). Oluline osa sellest dokumendist keskendub ka sellele, kuidas tagada veebilehtede arendamise järjepidevus ja kvaliteet, mis vastaks tänapäeva veebilahenduste arendamise nõuetele ja WCAG suunistele.
Lisaks eelnimetatule tuuakse käesolevas dokumendis välja ka parimad praktikad, mis aitavad meeskonnal paremini luua ja hallata pikaajalisi projekte.

Enne projekti tegelikku algust lepitakse kokku kõik erisused - millised nõuded konkreetses projektis ei rakendu (nt pole võimalik või vajalik).

Mõisted

Grid-system – reeglistik, mis pakub ühtset süsteemi kasutajaliidese komponentide laiuste ja kauguste kirjeldamiseks. Tänapäeval on *grid-system* see, mis kirjeldab ka veebilehe responsiivsuse. Ehk reegleid, mis kirjeldavad, kuidas veebileht erineva suurusega ekraanidel välja näeb.

Javascript – skriptikeel, mille abil luuakse veebilehele erinevaid interaktsioone. Tuvastades kasutaja tekitatud sündmuseid võimaldab keel lisada, muuta või kustutada veebilehe sisu. Üks osa Javascriptist on ka peidetud andmevahetuse võimaldamine.

HTML5 (Hyper Text Markup Language) – struktureeritud markeerimise keel. Viies põhiredaktsioon. Keel, mille abil kirjeldatakse veebilehe andmeid ja struktuuri.

CSS (Cascading Style Sheets) – keel, mille abil kirjeldatakse veebilehe välimust.

WCAG (Web Content Accessibility Guidelines) – dokument, mis pakub suuniseid, kuidas veebilehe sisu juurdepääsetavust parendada.

HTML

Veebilehe valmistamisel on HTML-i roll kirjeldada lehe struktuuri ja seda, mis andmed sellel lehel olemas on. Semantiline HTML tähendab seda, et HTML-li kirjutamisel on eelistatud esmajärjekorras semantilisi (tähendustega) elemente. Semantilised elemendid annavad olulist teavet nii otsingumootoritele kui ka lehe külastaja abitarkvaradele. Teave puudutab just seda, mis andmed on veebilehel ja kuidas need omavahel seotud on.

Semantilise HTML-i teine oluline eelis seisneb selles, et tagab kindluse tuleviku suhtes (future proofing) või kasutuskõlblikuse / väljanägemise browseris ilma CSS'ita. Koodi loetakse arendajate poolt ja tõlgendatakse arvutite poolt.

Nõuded

Nr	Nõude sisu	Nõude seletused
F E H1	**Lehe kirjeldamisel on kasutatud semantilisi elemente.**	Semantiliste elementide kohta saab täpsemat infot leheküljelt: http://html5doctor.com Nimetatud lehekülg koondab informatsiooni *W3C Working Group*i dokumentide refereeringut. Leht annab esmase ülevaate võimalikest elementidest ja nende kasutamiskohtadest. Täpsema info saamiseks viidatakse algmaterjalile. Halb näide `<div>` `  <div>` `  <h1>Pealkiri</h1>` `  <div><a href="#">Tagasi</a></div>` `  </div>` `  <p>Sisu...</p>` `</div>` Hea näide `<article>` `  <header>` `  <h1>Pealkiri</h1>` `  <nav><a href="#">Tagasi</a></nav>` `  </header>` `  <p>Sisu...</p>` `</article>`
F E H2	**Lehe sisupuu peegeldab lehe tegelikku hierarhiat.**	Pealkirjad ja mõned HTML5 elemendid (*Sections and Outlines of an HTML5 Document)* kirjeldavad veebilehe sisupuud. Puudulike pealkirjade või ebakorrektse HTML5 elementide kasutamine moonutab lehe tegelikku sisupuud. Testimiseks sobivad rakendused, mis oskavad välja joonistada HTML5 dokumendi sisupuu. Näiteks *WAVE Toolbar* või *HTML5 Outliner.* *Loe lähemat WCAG suunistest käesoleva dokumendi vastavast peatükist.*

FEH3	**Koodis puuduvad elemendid, mida on kasutatud selleks, et näidata ekraanil mõnda kujunduselementi. Näiteks ikoonid või muu graafika.**	Kuna HTML on mõeldud sisu struktureerimiseks, siis ei tohiks kasutada HTML-is elemente, mille eesmärk on vaid mõne kujunduselemendi ekraanile joonistamine. Näiteks ei ole korrektsuse huvides kohane lisada HTML-i elemente, mis on tühjad (ilma väärtuseta) ja on seal vaid selleks, et näiteks kuvada ikooni. Graafika eraldamine HTML-ist pakub võimalust juhtida kujundust ühest kohast. Ühtlasi võimaldab selline meetod muudatusi teha nii, et ei peaks sekkuma rakenduse äriloogikasse. Halb näide `<button type="button"><i class="fas fa-info-circle"></i></button>` Hea näide `<button type="button" class="fas fa-info-circle"><span class="sr-only">Info</span></button>` ***Märkus.*** *Siin pole mõeldud elemente, mille sisu täidetakse Javascripti abil. Näiteks Google Maps jmt. Küll aga peavad sellised elemendid sisaldama informatsiooni juhuks kui Javascript ei rakendu. Või juhtudeks, kui kasutaja veebilehitsejas on Javascripti kasutamine keelatud. Samuti ei kehti see kriteerium nn Backdrop jmt komponentidel, mida kasutatakse kujunduses komponentide nähtavuse parendamiseks. Selliste komponentide jaoks tuleks kasutada vastavaid ARIA parameetreid.*
FEH4	**HTML-is kirjeldatud lehe komponendid on eristatavad.**	Veebileht koosneb komponentidest ja komponendile kuuluvatest elementidest. Komponendipõhine lähenemine veebilehe arendamisel tähendab seda, et lehe loomisel on selgelt defineeritud lehe erinevad väiksemad osad (ehk komponendid/elemendid). Neid väiksemaid osasid kasutatakse erinevates sisulehtedes. Ehk lehe komponendid on korduvkasutatavad ja on sõltumatud neid ümbritsevast HTML-ist. HTML-ist on võimalik selgelt eristada, milline osa koodist on sisu ja milline on see osa, mille abil hoitakse koos lehe paigutust (*grid-system*). Eemaldades või lisades komponente ei muuda see tegevus lehe elementide paigutust. Alljärgnevas näites on kasutatud kahte komponenti: *layout* ja *card*. Esimese abil kirjeldatakse veebilehe sisu paigutust horisontaalsel teljel (teda ennast pole lehe külastajale näha), see hoiab veebilehte koos. Teine on aga sisu komponent, mis on lehe külastajale näha ja mille laius sõltub sellest, palju on talle ruumi antud *layout* komponendi vahendusel. Mõlemad on eraldiseisvad osad, millel on täita oma roll. Seega pole asjakohane märkida *layout*-ile, milline *card* tema sees välja näeb. See, milline üks või teine komponent välja näeb, seda kirjeldatakse vaid selle komponendi juures. Sellest siis alljärgnev halb ja hea näide. Halb näide `<ul class="grid">` `  <li class="col-xs-6 col-sm-4 card card--important">` `    <article class="card-content">` `      <h2 class="card-title">Pealkiri</h2>` `    </article>` `  </li>` `</ul>` Hea näide `<ul class="grid">` `  <li class="col-xs-6 col-sm-4">` `    <article class="card card--important">` `      <h2 class="card-title">Pealkiri</h2>` `    </article>` `  </li>` `</ul>`
FEH5	**Lehel on teade, kui kasutajal puudub Javascripti tugi.**	Javascript on tänapäeval *de facto* veebi osa. Lisades lehele interaktsioone ja kulissidetagust andmevahetust, pakub see lehe külastajale paremat kasutajakogemust kui seda teeks vaid HTML ja CSS-i kasutamine. Kuna Javascripti osakaal lehe vajaliku funktsionaalsuse tekitamiseks on suur, siis on hädavajalik külastajale teada anda, kui tema veebilehitsejal Javascript ei ole toetatud.
FEH6	**Lehel on teade, kui kasutaja külastab veebilehte iganenud veebilehitsejaga.**	Kui külastada lehte iganenud veebilehitsejaga, siis kuvatakse külastajale selle kohta teade. Teade sisaldab viiteid, kuidas uuendada või asendada oma veebilehitsejat. ***Märkus.*** *Iganenud veebilehitseja all peetakse silmas neid veebilehitsejaid, mille toetamine on tootja poolt lõpetatud. Kuna veebilehitsejate uuenduste tsükkel on kaasaegsetes veebilehitsejatel 6-12 nädala tagant, siis need veebilehitsejad tagavad parima kasutajakogemuse ja turvalisuse. Kasutajaid tuleks suunata kasutama just neid veebilehitsejaid. Vt ka nõuet FEU3.*

FEH7	**Kood on W3C valideeruv.**	Validaatorite eesmärk on pakkuda esmast teavet testitava koodi kohta. Valideerimise raportit ei saa käsitleda kui põhjust väita, et loodud kood on kasutuskõlbmatu. Igat raporti rida tuleb käsitleda eraldi ja võrrelda seda kokku lepitud nõuetega. Tarkvaraprojekti planeerimisel tuleb arvestada valitud tehnoloogiatega. Näiteks kui on kasutuses tehnoloogia, mis baseerub mitmetel kolmanda osapoole rakendustel, siis projekti teostajal ei ole võimalik tagada seda, et kõikide osapoolte genereeritud kood oleks täismahus valideeruv. Sellegi poolest tuleks eelistada neid teeke, mille väljund vastaks WCAG suunistele ja võimalikest mööndustest tuleb projekti alustamisel informeerida tellijat. Mööndused peavad olema põhjendatud, näiteks kolmanda osapoole tarkvara kasutamisel ja paremate alternatiivide puudumine. See muidugi ei tähenda seda, et valideerimisel avastatud vead (ehk "*errors*") tuleb aktsepteerida. Koodis esinevad vead tuleb parandada. Muud valideerimisel saadud tulemustele tuleb anda hinnang eraldi.
FEH8	**Graafika, mida on võimalik realiseerida CSS-iga, tuleb ka realiseerida CSS-iga.**	Erinevad kujunduselemendid, mis ei ole lehe sisu osa, tuleks realiseerida kasutades CSS-i võimalusi. Näiteks varjud, kumerad ääred, gradiendid jmt. CSS-is kirjeldatud kujunduselemendid võimaldavad eraldada sisu visuaalist. See pakub väga head paindlikkust kujunduse muutmiseks, täienduste tegemiseks jmt. Ühtlasi CSS-i abil loodud graafikat on kerge kohandada vastavalt ekraanile, millel teda kuvatakse. Veebilehitseja suudab CSS-i abil loodud graafikat vastavalt külastaja ekraani võimekusele kohandada.
FEH9	**Vormi sisestusväljadel on kasutatud asjakohast tüüpi.**	HTML võimaldab ära märkida, millist tüüpi väärtust sisestusväli eeldab (lisaks eeldamisele on võimalik ka lisada valideerimisreeglid, mis ka kontrollivad, kas sisestatud väärtus vastab reeglitele). Eriti oluline on see aina rohkem levima hakkanud puutetundlikele ekraanidele, milledes andmete sisestamisel kuvatavale klaviatuurile saab ette öelda, millist tüüpi sisu on vaja sisestada ja seade automaatselt optimeerib klaviatuuri just selle jaoks. Halb näide `1 <label for="femail">E-posti aadress</label>` `2 <input type="text" name="email" id="femail">` Hea näide `1 <label for="femail">E-posti aadress</label>` `2 <input type="email" name="email" id="femail">`

CSS

Veebilehe valmistamisel on CSS-i roll kirjeldada, milline see veebileht välja näeb. Ehk see, millisena oleme tänapäeval harjunud veebilehti nägema, on kirjeldatud kasutades CSS-i keelt. Kui veebilehel puudub viide CSS failile, siis veebilehe väljanägemise kirjeldab HTML elementide vaikeväärtused, mille on ära kirjeldanud veebilehitseja tootja.

Keskmiste ja suuremate veebilehtede loomisel ja selle CSS-i koodibaasi haldamiseks on soovitav veebilehe komponendid jagada erinevateks failideks. Siinkohal mängivad olulist rolli erinevad nn CSS-i preprotsessorid, mis võimaldavad luua CSS-i läbi struktureeritud lähenemise. Preprotsessorid on tööriistad, mis pakuvad tarkvaraloojale võimaluse jagada oma CSS pisemateks osadeks. Võimaldavad luua CSS-i arhitektuuri, kus suured komponendid toetuvad pisematele.

Tänapäeval ühed levinumad CSS-i preprotsessorid on: LESS ja SASS/SCSS. RIK kasutab SCSS-i.

CSS-i koodistandardi paika panemiseks ja kontrollimiseks oleks soovitav kasutada ka SCSS Linterit. Linter on tööriist, mis aitab valideerida SCSS-i koodi vastavalt kokkulepitud reeglitele. Tänapäeval on olemas abivahendid, mis kontrollivad koodi korrektsust töö käigus automaatselt.

Nõuded

Nr	Nõude sisu	Nõude seletused

<table>
<tr>
<td>F
E
C1</td>
<td>Kasutatud on mõnda levinumat CSS-i raamistikku (eelistatult Twitter Bootstrap).</td>
<td>Arendusprotsessi kiirendamiseks ja jätkusuutlikuse tagamiseks on hea aluseks võtta valmis Front-End komponentide teek, mis pakub ühtset ökosüsteemi ja dokumentatsiooni veebilehe valmistamiseks. RIK kasutab Twitter Bootstrap'i ja seda tuleks uutes projektides ka eelistada, kui ei ole kokku lepitud teisiti.
Kuna valmis teek pakub juba hulganisti komponente ja elemente, mida veebilehe valmistamisel saab kasutada, siis nende kombineerimine võimaldab luua pidevalt juurde uusi vaateid, ilma et protsessi peaks sekkuma kujundaja.
Teegi valimisel peab peale kvaliteetse dokumentatsiooni ja populaarsusele olema oluliseks kriteeriumiks ka asjaolu, et see oleks modulaarne ja võimaldaks muuta sisendväärtuseid (on kasutatud SCSS CSS-i preprotsessorit). Igal projektil on oma nõuded ja vajadused. Teek peab võimaldama sisse-välja lülitada oma funktsionaalsust. Modulaarsus ja sisendväärtuste muutmine võimaldab kokku panna just sellise koodibaasi, mida konkreetne projekt vajab. Projektis raames ~~on~~ peab sisse lülitatud olema ainult vajalik funktsionaalsus (kasutaja browser ei tõmba alla staatilisi ressursse, mida ei kasutata).</td>
</tr>
<tr>
<td>F
E
C2</td>
<td>CSS-i lähtekood sisaldab vaid standardseid atribuute.</td>
<td>Veebilehitsejate tootjad on loonud võimaluse eksperimentaalseid ja/või mitte standardseid atribuute kasutada lisades atribuudi algusesse veebilehitseja tootja eesliidese. Kuna neid eesliiteid on vaja vaid ajal, mil kasutatav tehnoloogia pole veel standardiks saanud või nimetatud funktsionaalsus pole veel tootja poolt täies mahus implementeeritud, siis lähtekoodis neid sisaldada ei tohi. Lähtekood peab sisaldama ainult standardile vastavaid atribuute.

Veebilehitsejate tootjate eesliidesed lisab eraldi rakendus (post-production). Selle jaoks on olemas erinevad nn autoprefixer tööriistad, mis pakuvad võimaluse juhtida, milliste veebilehitsejate ja milliste versioonidele eesliidest tuleb suunata (PostCSS).

<table>
<tr><th>Halb näide</th><th>Hea näide</th></tr>
<tr>
<td>

```
-webkit-box-shadow: 0 0 4px #333;
-moz-box-shadow: 0 0 4px #333;
-ms-box-shadow: 0 0 4px #333;
box-shadow: 0 0 4px #333;
```

</td>
<td>

```
box-shadow: 0 0 4px #333;
```

</td>
</tr>
</table>

Antud kriteerium puudutab ka tänaseks juba iganenud meetodeid, mille abil püütakse sihtida mõne konkreetse veebilehitseja spetsiifilisi vigu, mis võimaldab just selle veebilehitseja jaoks kirjutada eraldi CSS-i kirjeid. Selle asemel tuleb välja selgitada mitte konkreetse veebilehitseja ja selle versioon, vaid selle võimekus. Selle jaoks on olemas nii CSS-is kui ka Javascriptis vastavad tööriistad.

CSS-is on selle jaoks näiteks @support tingimus, mis võimaldab tuvastada CSS-i abil veebilehitseja võimekust. Näiteks:


```
@supports (display: grid) {
    .row {
        display: grid;
    }
}

@supports not (display: grid) {
  .row {
    display: flex;
  }
}

```

</td>
</tr>
</table>

FEC3	**CSS-i selektorina on eelistatud peamiselt CSS klasse.**	Visuaali kirjeldamiseks on kasutatud CSS-i klassi selektorit. See tähendab seda, et visuaali rakendamiseks kasutatakse CSS klasse ja/või HTML elemente. Eelistada tuleb visuaali sidumist CSS klassi ja vältida ID selektorit. Nõuet ei kohaldata tüpograafia kirjeldamiseks. Tüpograafia puudutab lehekülje sisu autoreid, kes sisestavad sisu mõne tekstiredaktori kaudu. Nendeks tekstiredaktoriteks on peamiselt nn WYSIWYG tüüpi tööriistad, mis võimaldavad sisu sisestada nagu seda tehakse näiteks Microsoft Wordis. Need tööriistad kirjutavad HTML-i koodi taustas ise ja nemad oskavad tähistada teksti osasid peamiselt HTML elementidega (p, h1-h6, table jne). ***Märkus.*** *CSS-i klassi eelistamise soovitus tuleneb sellest, kuidas toimub CSS-i selektori reeglite täitmine. Kui CSS-is on sarnaseid reegleid, siis mille põhjal otsustab veebilehitseja, millist reeglit rakendada? Et otsus oleks võimalik teha, antakse igale selektorile kaal. Suurema kaaluga selektorite reeglid rakenduvad. ID-ga ja pikad selektorid omavad märkimisväärset kaalu. Kui erinevate arendajate poolt mingi konkreetse komponendi visuaali muutmisel on igaüks kirjutanud enda selektori, siis on ilmselge, et iga arendaja koodi lisandumisel on konkreetse visuaali kirjeldamist puudutav CSS omandanud märkimisväärset kaalu. Selle tulemusel tekib olukord, kus hilisemate muudatuste tegemine võib osutuda võimatuks. Ühtlasi selline lähenemine läheb vastuollu levinumate CSS-i klassi nimetamise põhimõtetega.* Halb näide `button { background-color: red; font-size: 16px; }` Hea näide `.btn { background-color: red; font-size: 16px; }`
FEC4	**CSS selektorite pikkus on mõistlik.**	Mida pikem on CSS-i selektor, seda kauem võtab veebilehitsejal aega selle täitmine, samuti väheneb koodi jälgitavus ja loetavus. Seepärast on soovitav, et CSS-i selektor ei oleks pikem kui kolm-neli sammu. Kuna CSS-i selektori täitmist alustatakse vasakult paremale, siis need kolm-neli sammu peaks eelistatuna olema lähimad struktuurid. Tuleb ka arvestada, et pikem selektor omab märkimisväärset suuremat kaalu reeglite rakendumisel *(loe eelmise punkti märkust)*. Kui erand on põhjendatud, siis on erandid lubatud. Aga läbivalt tuleb siiski kasutada kuni kolme-nelja sammu pikkusi selektoreid. Halb näide `.mainnav ul li a { display: block; background-color: red; }` Hea näide `.mainnav a { display: block; background-color: red; }` Veebilehitseja tootjad pidevalt optimeerivad oma algoritme, et CSS-is kirjeldatud reeglite täitmine oleks võimalikult kiire. Kiirust mõõdetakse küll millisekundites, kuid mahukamate projektide korral võib tekkida olukord, kus mahukate selektorite täitmine võtab põhjendamatult palju ressurssi.

FEC5	**On kasutatud mõnda levinumat CSS klassi nimetamise põhimõtet.**	CSS-i klasside nimetamine peab olema läbi rakenduse ühesugune. Tänapäeval kõige levinum CSS klasside nimetamise põhimõte on BEM (*block-element-modifier).* Nõue tagab selle, et kui koodi kirjutavad erinevad arendajad, siis klasside nimetamise loogika on läbivalt sama. See muudab koodi ühtsemaks ja lihtsamini jälgitavamaks. Ühtlasi pakub BEM kahele kõige suuremale veebilehtede arendamise murekohale lahendust. Esiteks nõuab see seda, et kood oleks semantiline ja modulaarne. Teiseks vähendab võimalust, et erinevate arendajate poolt koostatud kood tekitab konflikte (ehk kirjutatakse teineteise vormingud üle). BEM kohta leiab täpsemat infot siit: https://en.bem.info

Halb näide

```
/* Arendaja 1 kood */
.box01 {
  background-color: red;
}

.box02 {
  background-color: green;
}

.large {
  height: 400px;
}

/* Arendaja 2 kood */

.box2 {
  background-color: yellow;
}
```

Hea näide

```
/* Arendaja 1 kood */
.card {
  background-color: red;
}

.card-success {
  background-color: green;
}

.card--large {
  height: 400px;
}

/* Arendaja 2 kood */

.card-warning {
  background-color: yellow;
}
```

FEC6	**SCSS-i lähtekood on struktureeritud komponendipõhiselt.**	Senised praktikad on näidanud, et veebilehe arendamisel ja ülesehitamisel on kõige paremaid tulemusi andnud komponendipõhine lähenemine: SCSS-i lähtekood on struktureeritud nii, et iga komponent on kirjeldatud oma failis. Sealjuures üldised ehk globaalsed parameetrid on koondatud ühte faili. See tähendab seda, et leht koosneb erinevastest plokkidest ja plokid elementidest. Iga komponendi muutmine/parendamine on võimaldatud nii, et see ei vajaks teiste komponentide samaaegset muutmist (st ei segaks teiste komponentide arendajaid). Selleks tuleb jagada SCSS koodibaas väiksemateks loogilisteks osadeks, milleks on näiteks navigatsioon, vormi komponendid, modaalaknad jmt. 1 // SETTINGS 2 @import "settings/settings.global"; 3 4 // TOOLS 5 @import "node_modules/sass-mq/mq"; 6 7 // GENERIC 8 @import "generic/generic.box-sizing"; 9 @import "generic/generic.normalize"; 10 @import "generic/generic.shared"; 11 12 // ELEMENTS 13 @import "elements/elements.page"; 14 @import "elements/elements.headings"; 15 @import "elements/elements.links"; 16 @import "elements/elements.quotes"; 17 18 // OBJECTS 19 @import "objects/objects.layout"; 20 @import "objects/objects.list-bare"; 21 @import "objects/objects.list-inline"; 22 @import "objects/objects.box"; 23 @import "objects/objects.table"; 24 25 // COMPONENTS 26 @import "components/components.buttons"; 27 @import "components/components.page-head"; 28 @import "components/components.page-foot"; 29 @import "components/components.site-nav"; 30 31 // UTILITIES 32 @import "utilities/utilities.widths"; 33 @import "utilities.spacings"; 34

FEC7	**CSS klasse kasutatakse vaid visuaali kirjeldamiseks.**	Ei ole hea praktika siduda Javascripti funktsionaalsust HTML elemendiga CSS klassi abi. Kasuta selleks data-* atribuute. Nõnda tekitatakse Javascripti ja HTML-i sidumiseks universaalne põhimõtte, mis ei mõjuta lehe semantikat ega sega visuaali ja Javascripti funktsionaalsust. `<button type="button" data-action="whatToDo">Button<button>` `<button type="button" data-apply="whatToApply">Button<button>` `<button type="button" data-toggle="whatToToggle">Button<button>` `$(document).on('click', '[data-action="whatToDo"]', doSomething);`

Responsiivsus

Responsiivsete veebilehtede arendamine sai hoo sisse, kui oli vaja hakata veebilehte muutma kasutatavaks ka väiksematel ekraanidel: nutitelefonidel, tahvlitel. Nüüd aga me ei saa enam lähtuda kolmest ekraanist: mobiil, tahvel ja arvuti. Ekraani mõõte, millega veebilehti tänapäeval külastatakse, on väga palju. Selleks, et igas suuruses oleks veebileht tarbitav, tuleb muuta kogu veebileht sõltuvaks ekraani mõõdust. Ühest küljest aitab meil seda teha juba *grid* süsteemi rakendamine, kuid *grid* määrab ära vaid komponentide laiused. Et oleks võimalik juhtida ka komponentide ja nende elementide propotsioone (teksti suurus, *padding-marging* jne) lähtuvalt ekraani suurusest, siis seda aitab meil saavutada juba mõnda aega CSS-is kasutatav ühik „rem“.

Mida ühik „rem“ tähendab? Kõige lihtsam on seda seletada nõnda, et selle väärtus sõltub HTML-i juurelemendi (*root element)* teksti suurusest. HTML-i juurelement on element „html“. Vaikimisi on selle elemendi teksti suurus 16px. See ütleb meile, et 1rem = 16px. Mida me saame selle teadmisega peale hakata? Me saame kogu veebilehel kasutatava disaini panna sõltuma juurelemendi teksti suurusest. Vaatame alljärgnevat näidet:

```
:root {
    font-size: 16px;
}

.btn {
    font-size: 0.888rem;
    padding: 0.5rem 1rem;
}
```

Pikslites kirjutades oleks:

```
.btn {
    font-size: 14px;
    padding: 8px 16px;
}
```

Kui aga muudame juurelemendi teksti suuruse sõltuvaks ekraani suurusest (ühtlasi määrame ära minimaalse ja maksimaalse teksti suuruse), siis saame luua olukorra, kus veebilehe komponentide ja elementide propotsioonid muutuvad vastavalt sellele, kui suur on lehe külastaja veebilehitseja aken. Nõnda ei teki enam olukord, kus suurel ekraanil on tekst ja elemendid liiga pisikesed ja väiksel ekraanil aga vastupidi. Meil on olemas nüüd keskne koht, mille kaudu saame muuta kogu veebilahenduse kujunduse propotsioone.

```css
:root {
    font-size: 14px;
}

@media screen and (min-width: 320px) {
    :root {
        font-size: calc(14px + 8 * ((100vw - 320px) / 2240));
    }
}

@media screen and (min-width: 2560px) {
    :root {
        font-size: 22px;
    }
}
```

```css
.btn {
    font-size: 0.888rem;
    padding: 0.5rem 1rem;
}
```

Loe täpsemalt *Fluid Typegraphy.*

Lisaks

CSS-i rakendamisel on mõistlik kaaluda varianti luua täiendavaid tööriistu (*Utilities*), mis hõlbustavad teha kasutajaliidese kujunduses parendusi nõnda, et selle jaoks ei pea muutma komponenti või selle elemente. Näiteks *Bootstrap* pakub selliseid tööriistu https://getbootstrap.com/docs/4.1/utilities/borders/ Responsiivsete lahenduste maailmas osutuvad need tööriistad hädavajalikuks, kuna arenduse käigus on vajadus veebilahenduse propotsioone juhtida vastavalt tekkinud olukorrale (näiteks lähtuvalt ekraani suurusest).

Javascript

Tänapäeval kuulub *Javascript* iga veebilahenduse juurde. *Javascript* on see osa veebilehest, mis annab sellele dünaamilisuse ja võime reageerida kasutaja toimingutele. Ühtlasi aitab luua ka nn *polyfill*-e, mille abil saab arendaja kasutada veebilehitsejas API-sid, mis on vaikimisi arendaja poolt toetatud veebilehitsejas puudu. See võimaldab kasutada tehnoloogiaid, mis on muidu kasutatavad vaid uuemates veebilehitsejates.

Dokumendiobjektide mudel *(DOM, Document Object Model)*

DOM on platvormist ja keelest sõltumatu W3C poolt standardiseeritud HTML dokumentidega suhtlemise liides. See liides võimaldab näiteks *Javascripti* lugeda ja muuta veebilehte. Selle liidesega suhtlemine on ka põhjus, miks osad veebilahendused tunduvad kasutaja jaoks aeglased või lihtsalt ei toimi ootuspäraselt. Kuna DOM sisaldab kogu informatsiooni kuvatava veebilehe kohta, siis on tegemist väga suure objektiga. Selles mingite muudatuste tegemine nõuab veebilehitsejal erinevaid tegevusi. Ja need tegevused, mida veebilehitseja mingi toimingu tulemusel peab täiendavalt tegema, on need, mida me soovime arendamise käigus hoida minimaalsena. Oluline on siinkohal ka meeles pidada, et on andmeid, mida DOM-is ei sisaldu. Näiteks kui me küsime Javascripti vahendusel elemendi kõrgust-laiust, siis veebilehitseja peab selle väärtuse küsimise hetkel välja uurima. Kui seda küsimist aga tehakse korduvalt ja erinevates skriptiosades erinevatel põhjustel, siis see on üks nendest põhjustest, mis võib muuta rakenduse aegalseks.

Selles, et neid probleeme vältida, on veebilehitseja tootjad loonud erinevaid täiendavaid API-sid, mille abil oleks võimalik DOM liidesega efektiivsemalt suhelda. Näiteks väga tihti on veebilahendustes vaja teada saada, kas üks või teine element on kasutaja jaoks nähtav. Ajalooliselt on seda probleemi erinevalt lahendatud, kuid kõik need nõuavad veebilehitsejalt liigset tööd. Selle jaoks on nüüd aga olemas eraldi *Intersection Observer API*, mis võimaldab seda funktsionaalsust paremini oma arendustest rakendada. Arendajal võib tekkida ka vajadus teada saada, kas DOM-is on midagi muutunud. Kas mingi osa skriptist on teinud HTML-is mingeid muudatusi. Varasemalt tuli selleks kirjutada eraldi skript, mis käib DOM-ist perioodiliselt muudatusi otsimas /kontrollimas. Skripti kirjutamise asemel on nüüd olemas efektiivsem muudatuste jälgimise võimalus - kasutada *MutationObserver API*.

DOM-iga suhtlemise meetodites on toimunud viimastel aastatel oluline nihe. Kui eelnevalt tuli arendajal kirjutada kood, mis mingi reegli alusel küsib DOM-ilt infot, siis nüüd on pigem vastupidi. DOM ütleb ise, kui jälgitav tunnus on muutunud.

Kui eelnimetatud DOM-i puudutavad API-d on midagi, mida pakuvad veebilehitsejad, siis nendele lisaks on erinevad arendajad loonud veel täiendavaid liideseid, mis on veebilehitsejast sõltumatud. Üks populaarsem liides on *Virtual DOM*. Internetist võib selle kohta leida väga palju materjali, aga lihtsamalt öeldes on tegemist DOM-i abstraktsiooniga ehk lihtsustatud ja eraldiseisva versiooniga kuvatavast veebilehest. Kuna veebilehitseja enda DOM on väga mahukas ja sellega suhtlemine võib osutuda ressursinõudlikuks, siis *Virtual DOM* pakub võimalust teha muudatusi veebilehes nõnda, et *Virtual DOM* ise vastutab selle eest, millised muudatused tuleb saata veebilehitseja DOM-i, et kasutaja saaks neid ka näha. Sellisel viisil muudetakse veebilehitseja DOM-i vaid siis, kui seda tõesti on vaja teha ja seda püütakse teha võimalikult ressursisäästlikult.

Virtual DOM-i kasutavad näiteks sellised tehnoloogiad nagu React, Vue, Angular jne. Eelnimetatud tehnoloogiad on pigem raamistikud, mille üks osa on *Virtual DOM*-i rakendaine. Internetist võib leida ka vaid *Virtual DOM*-i liidese jaoks mõeldud *Javascripti* teeke.

Nõuded

Nr	Nõude sisu	Nõude seletused
FEJ1	**JS-i lähtekood on komponendipõhine**	JS kirjutamisele tuleb läheneda modulaarselt. Koodibaas tuleb jagada väiksemateks loogilisteks komponentideks. Eelistada tuleb ES6 mooduleid (import/export). See parandab: • funktsionaalsuse selget eraldatust • koodi uuesti kasutamist • koodi testimist isolatsioonis • koodi hoomatavust arendaja poolt `// checkbox.js` `export default Checkbox;` `// app.js` `import Checkbox from 'checkbox/checkbox';` `new Checkbox(document.querySelector('[type="checkbox"]'));`
FEJ2	**Vorming, linter ja build protsess**	Projekt sisaldab käske koodi lintimiseks, testimiskes ja vajadusel kompileerimiskes. Projekti raames peab kood jälgima ühtset stiili ja standardit. Ühtne stiil ja standard tagatakse läbi konfigureeritud style guide (mõne laialt levinud) ja linteri. Linter aitab: • välja tuua potentsiaalseid vigu koodis • parandada koodi loetavust • hoida koodi kvaliteeti • keskenduda probleemi lahendamisele - https://eslint.org/ - https://standardjs.com/ - https://github.com/airbnb/javascript/

		- https://gulpjs.com/ - https://webpack.js.org/ - https://docs.npmjs.com/misc/scripts
FEJ3	**Kommentaarid**	`Koodi kommenteerimiseks/dokumenteerimiseks eelista JSDoc formaati - http://usejsdoc.org/index.html.` kasuta `//` järgneva koodi rea kommenteerimiseks kasuta `// FIXME:` et välja tuua probleeme kasuta `// TODO:` et välja tuua lahendusi probleemidele See annab: • võimaluse genereerida dokumentatsiooni funktsioonide, meetodite, konstruktorite kohta • ühtne kommenteerimise stiil • koodi editorid võivad kuvada lisa informatsiooni kasutatava funktsionaaluse osas

Veebi sisu juurdepääsetavussuunised (WCAG) ja üleüldised kasutajakogemust puudutavad nõuded

Veebi sisu juurdepääsetavussuunised hõlmavad mitmeid soovitusi veebi sisu juurdepääsetavuse parandamiseks. Eesti keeles on võimalik tutvuda suuniste täisdokumendiga siin: http://www.w3.org/Translations/WCAG20-et/.
Arendusprojektis tuleb kasutada raamistikke, mille väljund HTML peab vastama WCAG suunistele.

Kuid peamised põhimõtted on:

- Veebileht on struktureeritud korrektselt.
  Siin on mõeldud just seda, millest oli juttu ka juba HTML-i peatükis. Tuleb eelistada tähendustega elemente. Kui veebilehel on loetelu, siis tuleb kasutada ka vastavat HTML elementi. Täiendavat semantikat võimaldab lisada WAI-ARIA rollide mudel (https://developer.mozilla.org/en-US/docs/Learn/Accessibility/WAI-ARIA_basics) Rolle saab kasutada ka sellistes olukordades, kus semantiliste elementide kasutamine pole mingil põhjusel võimalik. Veebilehel esinevatele ajas muutuvatele meediumitele saab WAI-ARIA *States and Properties* mooduli abil lisada täiendavat informatsiooni.
- Kasuta tekstilist vastet kui visuaalselt on näha vaid ikoon, video, audio jmt.
  Kui lehel on kasutatud nupuna ikooni, siis lisa nupule ka info, mida see ikoon tähendab. Seda on kõige lihtsam teha kasutades HTML-is *title* või *alt* atribuuti. Vormi elementidel tuleks kasutada *label* elementi.
- Ära keskendu ainult visuaalile.
  Lehe sisu peab olema arusaadav ka siis, kui lülitame välja näiteks CSS või kui lehte loeb ette kasutajale tarkvara. Oluline on silmas pidada, et ikooniga või värviga tähistatud kujunduselemendid oleks ka vaegnägijale tarbitavad (https://www.w3.org/TR/WCAG20-TECHS/G95.html, https://www.w3.org/TR/WCAG20-TECHS/G18.html).
- Veebileht on kasutatav ka klaviatuuriga.
  Komponentide loomisel tuleks arvestada kasutaja harjumustega. Kui kasutajal on harjumus *checkbox* teha tühik klahviga aktiivseks, siis see peaks toimima ka siis, kui see *checkbox* on loodud kasutades Javascripti abi.

Täiendavad kasutajaliidese komponentide nõuded leiad RIKi UIG lehelt

Nõuded

Nr	Nõude sisu	Nõude seletused
FEW1	**Meediumitel, mis võimaldavad külastajale interaktsiooni, on kirjeldatud nn *hover* olek.**	Igal elemendil, millega kasutaja saab midagi teha (lingid, vormi väljad, nupud jmt), on kirjeldatud *hover* olek. *Hover* olek on see, kui kasutaja liigub hiirega selle elemendi peale. Erandiks on puutetundlikud ekraanid, kus *hover* olek võrdub *klikiga*. See tähendab, et funktsionaalsused, mis sõltuvad *hover-ist,* tuleb puutetundlikel ekraanidel ümber teha *klikipõhiseks*. Siit ka soovitus otsutada, milline on primaarne funktsionaalsus. Üldiselt liigub maailm selles suunas, et funktsionaalsus, mis toob kasutajale midagi nähtavale *hover-i* abil, ehitatakse siiski *kliki* peale, kuna tõusev trend on see, et veebilehti külastatakse pigem puutetundlike ekraanidega.
FEW2	**Vormi elementidel on kirjeldatud *focus* olek.**	*Focus* olek aitab kasutajal eristada aktiivset vormi elementi teistest. ***Märkus:*** *selleks, et hinnata, kas focus on normaalselt nähtav, tuleb lähtuda WCAG poolsest soovitusest teksti värvi ja teksti taustavärvi vaheline kontrasti määrast. Tööriist testimiseks: https://webaim.org/resources/contrastchecker/*
FEW3	**Elementidel on kirjeldatud *active* olek.**	*Active* oleks aitab kasutajal eristada aktiivset elementi. Näiteks klaviatuuri abil lehel ringi liikudes saab aktiivne interaktsiooni võimaldav element teda eristava vormingu.
FEW4	**Elementidel on kirjeldatud *visited* olek.**	Mahukate veebilehtede korral on hea anda lehe külastajale tagasisidet viidete kohta, mida ta on juba külastanud. Sedasi lehe korduval külastamisel on tema persionaalne ajalugu näha.
FEW5	**Veasituatsioonid on sõnastatud asjakohaselt ja omavad vastavalt situatsioonile korrektset visuaali.**	Veasituatsioonide tekkimisel antakse kasutajale asjakohast tagasisidet. Teade on tähistatud nii värvi kui ka vastava ikooniga. Kehtib reegel, et süsteemi tõrked, mis ei tulene kasutaja tegevusest, tähistatakse punase värviga. Muud tõrked, mis on põhjustatud kasutaja käitumisest, pigem oranziga (ehk hoiatava, informatiivse tooniga).
FEW6	**Situatsioonid, kus informatsioon on mõnes komponendis puudu või on alles saabumas, on selgelt tähistatud.**	Veebilehel on arvestatud olukordadega, kus näiteks tabel on tühi. Sellistel olukordades tuleb kasutajale anda infot, miks see on tühi ja vajadusel lisada juurde, mida ta peaks tegema, et see infoga saaks täidetud. Sama kehtib ka ajas muutuvate meediumite kohta. Kui mingi informatsiooni laadimine nõuab rohkem aega, siis selle laadimise kohta tuleb anda kasutajale tagasidet.
FEW7	**Veebilehel kasutatud animatsioonid parendavad kasutajakogemust.**	Nii nagu päris maailmas, nõnda ka veebis, peavad animatsioonid peegeldama objektide kergust, raskust, suurust jmt. Animatsioonid peavad olema kasutajat toetavad ja kirjeldama animeeritava elemendi suhet ülejäänud lehega. Et animatsioonid ei hakkaks liialt riistvara koormama ega kasutajat ärritama, tuleb animatsioonide kirjeldamisel lähtuda animatsiooni optimeerimise põhimõtetest. Nendeks on: - animatsioonid ei tohi nõuda veebilehitsejalt põhjendamatuid kordusi; - animeerida tuleb CSS-i atribuute, mis on veebilehitsejate poolt optimeeritud; - animatsioonid tuleb käivitada komplektina ja ajastatult veebilehitsejale kõige paremal hetkel (*requestAnimationFrame).* Mida ühe või teise CSS-i atribuudi muutmine veebilehitsejalt nõuab, saab tutvuda lehel *CSS Triggers* ***Märkus****: Animatsioonid peavad olema põhjendatud, kui animatsioon on põhjendamata ja/või näiteks epileptikutele isegi ohtlik, siis see kriteerium ei ole täidetud .*

Meediumite kasutamine veebilehel (veebilehe laadimiskiirus)

Mitmekesise meediumi kasutamine veebilehel avaldab otsest mõju lehe laadimiskiirusele ja sellele, kui kiiresti veebilehitseja suudab lehe külastajale valmis meisterdada. Mida suurem on veebilehe toimimiseks vaja erinevate ressursside osakaal, seda kauem läheb aega, mil leht on kasutamiseks valmis.
Jättes korraks kõrvale CSS ja Javascript failid, siis kõige suurematekss veebilehe kuvamise blokeerijateks on pildid ja videod.
Piltide kasutamisel tuleb pöörata tähelepanu nende mahule. Ei ole sobilik lisada veebilehele pilte, mille maht on põhjendamatult suur. Kui on alust arvata, et kasutavate piltide mahud on liiga suured, siis alljärgnevad tööriistad võimaldavad nende mahtusid vähendada:

- http://www.jpegmini.com
- https://pngquant.org
- https://imageoptim.com
- https://github.com/google/guetzli

Videode kasutamisel tuleb arvestada sellega, et videode vaatamiseks on vaja selle mängijat. Igasuguse mängija visualiseerimine ja käivitamine võtab aega, kuna mängija jaoks on vaja alla laadida selle käivitamiseks vajalikke ressursse. Kui lehel on mitu sellist mängijat, siis iga mängija jaoks tuleb veebilehitsejal tööd teha. Iga täiendav liigutus aga nõuab aega ja see avaldab märkimisväärset mõju lehe kiirusele.

Mis puudutab aga CSS-i ja Javacripti faile, siis nende kasutamisel tuleb tekitada olukord, kus:

- kõik lehel kasutusel olevad CSS-i failid oleksid liidetud üheks failiks ja faili sisu oleks minimeeritud;
- kõik lehel kasutusel olevad Javascripti failid oleksid liidetud üheks ja faili sisu oleks minimeeritud.

Veebilehe kiiruse optimeerimise ja selle mõjude kohta saab täiendavalt lugeda lehelt: https://kinsta.com/learn/page-speed/ ja https://developers.google.com/web/fundamentals/performance/critical-rendering-path/

Pildigraafika optimeerimine

Tänapäeval on võimalik kasutada erinevaid tööriistu, mis võimaldavad juba töö käigus pildigraafikat optimeerida. Selle jaoks tuleb vaid seadistada taustateenus, mis jälgib kataloogides toimunuid muudatusi ja vastavalt etteantud mustritele käivitavad optimeerimist teostavad skriptid.
Kõige levinumad taustateenuste skriptide jooksutajad on:

- Gulp
- Grunt
- Webpack

Kuna kõik kolm nimetatud kasutavad mooduleid, mis on kirjutatud kasutades Javascripti, siis on võimalik ilma suurema vaevata luua lähtuvalt vajadusest projektile omane skript pildigraafika optimeerimiseks. Seda siis eeldusel, et mõni juba olemasolevatest ei sobi. Küll aga juba vabalt kättesaadavad skriptid sisaldavad enimlevinud algoritme erinevate pildigraafika optimeerimiseks. Tasulistest tarkvaradest tooks välja JPEGmin-i. Tegemist on suurepärase tööriistaga just JPG piltide optimeerimiseks. Tihti saavutab see pea 3x mahu vähenemise ilma et pilt kaotaks silmnähtavalt kvaliteedist.
Tutvu pildigraafika erinevate optimeerimise meetoditega siin.

Responsiivsed pildid

Tehnoloogi aitab lehe autoril lisada veebilehele piltmaterjali sellisel kujul, mis võimaldab neid tarbida erinevate suurutel ekraanidel. Siinkohal tuleks silmas pidada peale pildi kõrguse ja laiusele ka selle mahtu ja fookuspunkti. Fookuspunktist tuleb kinni pidada, et välitda olukordi, kus pildil kujutatud oluline element oleks ebanormaalselt lõigatud või muul viisil rikutud.

HTML-i elementile *img* on võimalik lisada juurde täiendav parameeter, mis sisaldab informatsiooni võimalike alternatiivsete pildi variatsioonide kohta: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images

Koodibaasi kvaliteedi tagamine

CSS-i kirjutamisel on suureks abiks erinevad nn CSS-i preprotsessorid. Need on tööriistad, mille abil on võimalik läheneda CSS-i kirjutamisele struktueeritult. Preprotsessorid muudavad CSS-i skriptikeeleks, mis kompileeritakse tavaliseks CSS-iks, mida veebilehitsejad oskavad lugeda. Skriptikeelelaadne kirjutamine võimaldab korduvaid tegevusi hoida ühes kohas.

CSS-i preprotsessorid

Tänapäeval levinumad CSS-i preprotsessorid on LESS ja SASS/SCSS. Kuna viimasel ajal on populaarsust kogumas pigem SASS/SCSS, siis võiks täna alustavates projektides pigem eelistada SASS/SCSS preprotsessorit. Seda põhjusel, et nimetatud preprotsessori kompetentsi on tööturul lihtsalt rohkem.

CSS-i postprotsessorid (PostCSS)

Tänapäeval täiendava võimalusena on loodud tehnoloogia, mille abil on võimalik olemasolevat CSS-i järeltöödelda. Need tööriistad pakuvad võimaluse CSS-i lähtuvalt sisendparameetritele või teatavatele mustritele muuta selliseks nagu vaja. Näitena võib tuua olukorra, kus ühe rakenduse jaoks tehtud CSS oleks vaja muuta teise rakenduse jaoks sobilikuks nõnda, et see hakkaks vastama uue rakenduse stiilinõuetele (näiteks vaja muuta kasutatavat fonti, värve jmt).

Üks levinum põhjusi kasutada *PostCSS*-i on aga vajadus (*Autoprefixer*) tagada uuemate tehnoloogiate tagasiühilduvat tuge ammutades andmeid *Can I Use* andmebaasist.

Tagasiühilduvat tuge vajavad:

- *eksperimentaalses staatuses olev funktsionaalsus*

  See on funktsionaalsus, mis on küll veebilehitsejal olemas, aga tootja poolt on märgitud eksperimentaalseks. Eksperimentaalsete funktsionaalsuste eristamiseks kasutavad veebilehitsejad eesliiteid (*prefix).* Näiteks
  -webkit-box-shadow (spetsifikatsioon näeb ette aga *box-shadow*). Igal tootjal on oma.

- *varasema (kehtetuks tunnistatud) funktsionaalsust kirjeldava spetsifikatsiooni toetamine*

  Uute veebitehnoloogiate lisamine veebilehitsejatesse on modulaarne. Iga tootja otsustab ise, millal millist moodulit oma tootesse lisab. Tulemuseks on see, et osa veebilehitsejaid toetavad funktsionaalsuse varasemat spetsifikatsiooni, teised aga uuemat. Sellest tulenevalt kasutavad veebilehitsejad erinevat süntaksit funktsionaalsuse rakendamiseks.

Autoprefixed on PostCSS-i moodul, mis on loodud selleks, et lahendada eelnimetatud tagasiühilduva toega seotud küsimusi, lisab vajadusel eesliitega juurde puuduvad atribuudid ja teeb süntaksi parendusi seal kus vaja.

PostCSS pakub lahendusi erinevatele probleemidele. Vabalt kättesaadavatest moodulitest saab ülevaate siit: https://www.postcss.parts.
Kuna moodulid on kirjutatud kasutades *Javascripti*, siis on lihtne neid mooduleid ise lähtuvalt projekti spetsiifikast juurde luua.

Linter *(lint, linter, linting)*

Linter on tööriist, mis otsib koodist võimalikke vigu. Front-End arenduses saab sellist tööriista kasutada automaatse validaatorina, mis testib iga arendaja koodi vastavalt ette antud standardile. Selline tööriist pakub võimalust defineerida koodistandard iga projekti jaoks eraldi (ehk saab panna sõltuma projekti spetsiifikast). Selliselt saab tagada, et kõikidel projektis osalevatel arendajatel on kood kirjutatud järgides samu põhimõtteid.

Muud nõuded

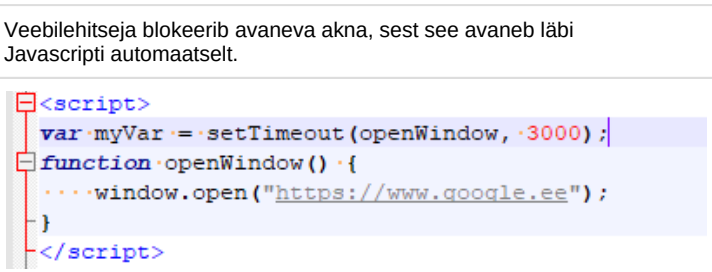
Nõuded

<table>
<tr><th>Nr</th><th>Nõude sisu</th><th>Nõude seletused</th></tr>
<tr><td>FEU1</td><td>Rakendus peab olema graafiliselt eskaleeruv.</td><td>Nii avalikuks kui ka sisemiseks kasutamiseks tehtav rakendus peab olema graafiliselt eskaleeruv (responsive web design) ja mugavalt kasutatav kõigi enamlevinud seadmete resolutsioonidega vahemikus 320x568px kuni 3840x2160px. Ühegi nimetatud resolutsiooni korral ei tohi tekkida horisontaalset kerimisriba. Implementatsioon peab jälgima “Mobile first” kontseptsiooni.</td></tr>
<tr><td>FEU2</td><td>Vaikimisi veebilehitseja poolt blokeeritavaid hüpikaknaid ei tohi kasutada.</td><td>Kõik aknad, mis avanevad peavad olema sellised, mida üksi brauser oma vaikimisi konfiguratsioonis ei blokeeri. St ei tohi tekkida olukorda, kus veebilehitseja blokeerib vaikimisi avatava akna ning kasutaja peab selle avamise eraldi veebilehitsejas lubama.

Veebilehitseja poolt võidakse blokeerida avatav aken olukorras, kus rakendus üritab seda avada automaatselt, st mitte kasutaja vahetu tegevuse peale. Näited:

Veebilehitseja blokeerib avaneva akna, sest see avaneb läbi Javascripti automaatselt.

Veebilehitseja blokeerib avaneva akna, sest see avaneb läbi Javascripti automaatselt.

Veebilehitseja ei blokeeri avanevat akent, sest see avaneb kasutaja vahetu tegevuse peale.

 
</td></tr>
<tr><td>FEU3</td><td>Veebipõhine kasutajaliides peab olema kasutatav enamlevinud veebibrauseritega. Minimaalselt Mozilla Firefox, Safari, Chrome ja Edge arenduse testimise hetkel 2 viimast (major) versiooni ning Internet Explorer 11.</td><td>
<table>
<tr><th>Operating system</th><th>Browser</th></tr>
<tr><td>Windows</td><td>Internet Explorer 11</td></tr>
<tr><td>Windows</td><td>Edge (2 viimast versiooni)</td></tr>
<tr><td>Windows</td><td>Google Chrome (2 viimast versiooni)</td></tr>
<tr><td>Windows</td><td>Mozilla Firefox (2 viimast versiooni)</td></tr>
<tr><td>macOS</td><td>Safari (2 viimast versiooni)</td></tr>
<tr><td>macOS</td><td>Google Chrome (2 viimast versiooni)</td></tr>
<tr><td>macOS</td><td>Mozilla Firefox (2 viimast versiooni)</td></tr>
<tr><td>iOS</td><td>Safari (2 viimast versiooni)</td></tr>
<tr><td>iOS</td><td>Google Chrome (2 viimast versiooni)</td></tr>
<tr><td>Android</td><td>Google Chrome (2 viimast versiooni)</td></tr>
</table>
</td></tr>
</table>