jamk

Räätälöity WordPress-teemakehitys

Tuomas Marttila

Opinnäytetyö, AMK
Toukokuu 2021
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Marttila, Tuomas

Räätälöity WordPress-teemakehitys

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2021, 54 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

WordPress on maailman suosituin sisällönjulkaisujärjestelmä, eikä turhaan. WordPressin käyttö on helppoa ja melkein kuka tahansa kykenee sen asentamiseen ja käyttöön, mutta helppokäyttöisyydellä on kuitenkin haittapuoli. Helppokäyttöisyyden vuoksi huonoja, WordPressin päälle rakennettuja, valmisteemalla varustettuja verkkosivuja löytyy paljon. Tämän takia WordPress-sivut usein leimataan ”valmissivuiksi”. Näin ei kuitenkaan tarvitse olla. WordPress-sivut voi toteuttaa oikein tekemällä teeman itse käyttämällä hyviä tekniikoita.

Opinnäytetyön tehtävänä oli toteuttaa asiakastyönä räätälöity WordPress-teema käyttäen Digitoimisto Duden kehittämää Air-light-aloitusteemaa pohjana. Pääpaino opinnäytetyössä oli CSS/Sass-tyylittely. Tehtävän lisäksi tavoitteina oli henkilökohtaisen osaamisen syventäminen sekä WordPress-sivustoihin kohdistuvien ennakkoluulojen murtaminen.

Opinnäytetyössä tutkittiin teemakehitykseen liittyviä hyviä käytänteitä ja tekniikoita. Työssä käydään läpi olennaiset WordPress-teemakehityksen osa-alueet keskittyen tyylittelyyn. Teoriaa ja omaa aiempaa osaamista käytettiin käytännön toteutuksen eli WordPress-teeman tekemiseen. Näiden perusteella opinnäytetyö toteutettiin soveltavana kehitystyönä.

Lopputuloksena on näyttävä teema, joka on tehty modernein tekniikoin. Teemassa on 16 asiakkaan vapaasti muokattavaa lohkoa ja useita uniikkeja näkymiä. Erityistä huomiota toteutuksessa pidettiin modulaarisuuteen, saavutettavuuteen ja responsiivisuuteen.

Tehtävän suorittamisen lisäksi prosessin aikana onnistuttiin syventämään henkilökohtaista osaamista muutamissa teemakehityksen osa-alueissa. Lopuksi pohdittiin kehittäjänä teeman onnistuvuutta eri näkökulmista ja kuinka tärkeää on kehittäjänä pysyä teknologioiden kehityksessä mukana.

Avainsanat (asiasanat)

Teemakehitys, WordPress, aloitusteema, saavutettavuus

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Marttila, Tuomas

Tailored WordPress theme development

Jyväskylä: JAMK University of Applied Sciences, May 2021, 54 pages.

Engineering and technology. Information and Communication Technologies. Bachelor´s thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

WordPress is the world's leading content management system. WordPress is easy to use and so almost anyone can install and use it. On the other hand, ease of use has downsides. Because of ease of use, there are a lot of bad websites built on WordPress. These websites almost always use a theme from the WordPress theme catalog and are bloated with plugins. For these reasons WordPress websites are often considered as low effort and straight bad. This doesn't have to be the case. You can build WordPress website the right way by yourself using modern techniques.

The task of this thesis was to develop a tailored WordPress theme for a customer using a starter theme called Air-light. The focus of this thesis was CSS/Sass styling. Along with the task, there was two goals. First goal was to deepen my own knowledge about the topic and to improve as a developer. Second goal was to change prejudices that people may have of WordPress being low effort.

In this thesis we did research on the good practices of WordPress theme development and modern technologies involved with it. Using the knowledge gathered from this research and previously learned techniques, we developed a tailored theme for WordPress. For this reason, the thesis was carried out as an applied development work.

The result was a good-looking theme made by using the best practices and the most modern technologies. The theme has 16 modular modules and several unique pages. During the development, special focus was kept on the code being modular, accessibility and responsiveness.

Along with developing a theme, we managed to deepen our knowledge about theme development. In the end we a discussion on the success of the project from different perspectives and how the industry moves forward constantly.

Keywords/tags (subjects)

Theme development, WordPress, starter theme, accessibility

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Tausta ja toimeksiantaja

WordPress on erinomainen alusta verkkosivuille sen helppokäyttöisyyden takia ja sen näkee käyttäjämäärästä. Tästä huolimatta web-kehityksen maailmassa WordPressillä tehdyt sivut usein kategorioidaan “valmissivujen” alle. Tyypillistä on ladata valmisteema sekä asentaa muutama lisäosa ja tämän jälkeen pitää sivuja valmiina. Näin ei kuitenkaan tarvitse välttämättä olla. WordPress-sivut voi tehdä oikein tekemällä ne itse alusta loppuun. Tekemällä itse ja käyttämällä oikeita tekniikoita sekä käytänteitä saa aikaan tekniikaltaan modernit, näyttävät ja yksilölliset sivut.

Toimeksiannon opinnäytetyölle antoi Digitoimisto Dude Oy. Dude toteuttaa vahvasti räätälöityjä verkkosivuja kaiken kokoisille yrityksille WordPress-pohjalle. Yritys toteuttaa kaiken verkkosivuihin liittyvän, suunnittelusta koodaamiseen ja sitä kautta julkaisuun ja ylläpitoon. Jokainen sivusto on alusta asti koodattu asiakkaan näköiseksi ja tarpeiden mukaan. Dude on perustettu 2013 ja sijaitsee Jyväskylässä.

WordPress-yhteisö ja avoin lähdekoodi ovat kulmakiviä Dudelle. Tämän vuoksi avoimen lähdekoodin lisäosia (eng. plugin) ja muita WordPress-kehitystä nopeuttavaa ja helpottavaa koodia on julkaistu GitHubiin. Yksi näistä julkaisuista on Duden kehittämä Air-light WordPress-aloitusteema (eng. starter theme). Tässä opinnäytetyössä perehdytään tarkemmin kyseiseen aloitusteemaan, sen hyviin puoliin ja lopuksi Air-lightin jalostamiseen upeaksi sivustoksi. Dude toteuttaa kaikki WordPress-verkkosivut Air-light aloitusteeman päälle.

1.2 Tehtävä ja tavoitteet

Opinnäytetyön tehtävänä oli tuottaa räätälöity WordPress-teema asiakaskäyttöön. Teema rakennettiin Duden tuottaman Air-light aloitusteeman päälle käyttäen moderneimpia tekniikoita ja hyviä teemakehitykseen liittyviä käytänteitä. Keskeisimpiä osa-alueita olivat responsiivisuus, saavutettavuus sekä siisti ja modulaarinen koodirakenne, joka teki teemasta helpommin jatkokehitettävän. Opinnäytetyö on kirjoitettu puhtaasti front-end-kehittäjän näkökulmasta, jonka

pääasiallinen tehtävä tässä asiakasprojektissa oli CSS/SCSS-tyylittely. Tämän vuoksi vain välttämättömät asiat yleisestä WordPress-kehityksestä käydään läpi, jotta lukija ymmärtää luetun.

Kuten jokaisen verkkosivuprojektin, myös tämän, henkilökohtaisena tavoitteena oli oman taidon syventäminen. Web-kehittäjänä kasvamisen kannalta pahinta on jäädä paikalleen ja lopettaa uusien asioiden opettelu. Olisikin hienoa, jos tämän opinnäytetyön lukija oppisi jotain uutta ja innostuisi tekemään mahdollisimman hyviä WordPress-sivuja itse. Tavoitteena oli myös murtaa WordPress-sivustoihin kohdistuvia ennakkoluuloja, jotka usein pitävät WordPress-sivustoja nopeasti hutaistuina valmissivuina.

1.3 Tutkimusmenetelmällisyys

Tämä opinnäytetyö toteutettiin soveltavana kehitystyönä. Soveltava kehitystyö tähtää käytännön ratkaisuun tai sovellutukseen valmiiksi olemassa olevan tiedon pohjalta (Tutkimus- ja kehittämistoiminta n.d). Toisin sanoen tässä opinnäytetyössä tehdään WordPress-teema eli sovellus aikaisempaa tietoa apuna käyttäen.

Opinnäytetyössä käydään läpi WordPress-teeman räätälöinnin työprosessia. Työprosessissa selostetaan läpi teeman kehittämiseen liittyviä hyviä käytänteitä ja tekniikoita. Käytänteiden ja tekniikoiden tietopohja perustuu aiemmin hankittuun osaamiseen sekä verkossa sijaitseviin artikkeleihin. Työssä käytettiin olemassa olevaa tietoa, omaa ja verkosta löytyvää, lopputuloksen, tässä tapauksessa räätälöidyn teeman, saavuttamiseksi.

2 Työssä käytetyt tekniikat ja teknologiat

2.1 WordPress

WordPress on avoimen lähdekoodin sisällönhallintajärjestelmä eli työkalu, jolla käyttäjä kykenee muuttamaan verkkosivujen sisältöä ilman koodaamista. WordPress on selvästi suosituin sisällönhallintajärjestelmä verkkosivujen rakentamiseen ja sen osuus kaikista verkkosivuista mukaan lukien myös ne, jotka eivät käytä sisällönhallintajärjestelmää, on 41 %. (Usage statistics of content management systems n.d.)

WordPressin pääasiallinen ohjelmointikieli on PHP ja muita käytettyjä ohjelmointikieliä ovat HTML ja JavaScript. Tyylittelyyn käytetään CSS:ää.

Keskeisiä asioita verkkosivua rakentaessa WordPressillä ovat teema ja lisäosat (eng. plugin). Teemalla voidaan vaihtaa sivuston ulkoasua ja lisäosilla saadaan toiminnallisuutta. Teeman ei tule sisältää merkittävää toiminnallisuutta, koska silloin vaihtamalla teemaa menettää samalla kyseisen toiminnallisuuden. (What is a Theme? n.d.)

Teemoja on erilaisia. WordPress ja sen lukuisat käyttäjät tarjoavat valtavan määrän valmiita teemoja, joita voi ladata ja vaihtaa vapaasti sivuille vain muutamalla klikkauksella. Näitä kutsutaan valmisteemoiksi. Iso osa valmisteemoista on ilmaisia käyttää, mutta maksullisiakin löytyy. Kaikki valmisteemat ovat samalla myös pääteemoja (eng. parent theme). Pääteemaksi lasketaan kaikki teemat, jotka sisältävät vaaditut WordPress-sivupohjat (eng. template). Lapsiteemaksi (eng. child theme) kutsutaan teemaa, joka pohjautuu johonkin pääteemaan, mutta siihen on tehty omia muutoksia. Lapsiteema ei tee muutoksia suoraan pääteemaan, vaan lisää omat muutokset erillisistä tiedostoista. Näin pääteeman päivittäminen onnistuu, vaikka siihen on tehty muutoksia lapsiteeman kautta. (Child themes n.d.)

Räätälöityä WordPress-sivustoa kehittäessä on tehtävä teema itse. Teeman voi joko rakentaa alusta alkaen itse, tehden kaiken kansiorakenteesta asti tai rakentamalla aloitusteeman (eng. starter theme) päälle. Aloitusteema on aiemman määritelmän mukaan pääteema, mutta sitä ei suositella käytettäväksi sellaisenaan, vaikka aloitusteema siihen käytännössä voi pystyäkin. Aloitusteemat ovat tehty kehitettäväksi ja räätälöitäväksi omaan tarkoitukseen, niissä on valmiina teemaan liittyviä pakollisia asioita, jolloin kehittäjä pääsee nopeammin tekemään ulkoasua. Aloitusteemasta ei tule tehdä lapsiteemaa. (Hughes 2018.)

Teema koostuu sivupohjista, mutta minimissään vain kahdesta tiedostosta: sivupohjasta index.php ja tyylitiedostosta style.css. Sivupohjat sisältävät HTML:ää, PHP:ta sekä sivupohja-tageja. Sivupohjia voi olla, ja usein onkin, useampia kuin, vain index.php.

Kuviossa 1 on koottu etusivu. Sivulle on kutsuttu tageilla header, footer, hero ja käyttäjän asettamat modulaariset lohkot. Käyttämällä tageja, teeman koodin voi rikkoa modulaarisiksi palasiksi ja täten helpommin hallittavaksi. (Template files n.d.)

front-page.php

```php
<?php
namespace Air_Light;

get_header(); ?>

<main class="site-main">
  <?php include get_theme_file_path( 'template-parts/hero-fp.php' ); ?>
  <?php include get_theme_file_path( 'template-parts/content-modular.php' ); ?>
</main>

<?php get_footer();
```

Kuvio 1. front-page.php-template.

2.2 Responsiivisuus

Responsiivisuudella tarkoitetaan sivun elementtien asettumista käyttäjäystävälliseen muotoon päätelaitteen näyttökoon perusteella. Älypuhelimet ja mobiililaitteet ylipäätänsä ovat olleet jo pitkään suosituimpia laitteita internetin selaamiseen. Vuoden 2021 ensimmäisen neljänneksen aikana mobiililaitteiden osuus kaikesta verkkoliikenteestä oli 54,8 % (Clement 2021). On siis ensisijaisen tärkeää, että verkkosivut ovat miellyttäviä käyttää mobiililaitteilla. On olemassa lukemattomia välimallin näyttökokoja 8k resoluution omaavasta televisioista aina pieniin mobiililaitteisiin asti. Kaikkien tähän väliin menevien näyttöjen koot on otettava huomioon, mikäli haluaa täydellisen responsiivisuuden. Tämän takia onkin syytä varmistaa, että sivusto toimii moitteettomasti millä tahansa realistisella resoluutiolla.

”Mobile first” on kehitystapa, joka on noussut kehittäjien keskuudessa suureksi puheenaiheeksi nykyajan suuren mobiilikäytön seurauksena. Mobile first-ajattelu kannustaa kehittämään sovellukset ja verkkosivut ensisijaisesti mobiililaitteille, siirtyen myöhemmin vasta suurempiin näyttökokoihin. Ideana tässä on panostaa mobiilikehitykseen ja tuottaa mahdollisimman hyviä mobiilisovelluksia ja –sivuja. Suurimpia syitä kehittää Mobile first-tavalla on se, että Googlen hakukone indeksoi sivuja nykyään mobiilin kautta, jolloin on tärkeää, että sivut ovat mobiiliystävälliset. Tämä nostaa sivujen hakukonenäkyvyyttä. Lopputulos on kuitenkin se, joka ratkaisee. Samaan lopputulokseen, mobiiliystävälliseen ulkoasuun, pystyy yhtä hyvin pääsemään aloittamalla kehityksen tietokonenäytöistä. Mobile first harvoin säästää koodirivejä, joten suosittelisin käyttämään itselle sopivinta ja helpointa ratkaisua. On tärkeää ottaa huomioon, että mobiilin priorisoiminen on merkittävämmässä osassa, kun puhutaan varsinaisesta sovelluskehityksestä. Tässä opinnäytetyössä tehdään asiakkaan verkkosivuille teemaa, eikä sovellusta.

Responsiivisten sivujen toteuttamiseen on kolme vaihtoehtoa: responsiivinen web design, dynaamisesti vaihtuva HTML-sivupohja tai kokonaan erilliset URL-osoitteet eri laitteille. Näistä tehokkain, ylläpidettävin ja hakukoneoptimointia ajatellen selvästi paras on responsiivinen web design. Responsiivisen web designin etuna muihin on sen pohjautuminen vain yhteen HTML-pohjaan ja CSS-tiedostoon, jotka käyvät kaikille resoluutioille. Muut vaihtoehdot pohjautuvat erillisiin HTML-pohjiin, joita tarjoillaan päätelaitteen selaimen tietojen pohjalta tai kokonaan erillisestä URL-osoitteesta. Esimerkkinä tästä voisi olla www.google.fi sijasta www.mobile.google.fi, kun sivustolla käydään mobiililaitteella. Ongelmana näissä ratkaisuissa on se, että huomioon otettavia näyttökokoja on monia, jolloin pohjia on tarve tehdä useampi kappale. Tämä taas johtaa ylläpidettävyyden heikkenemiseen. (Choose a mobile configuration 2021.)

Teknisesti responsiivinen web design tarkoittaa sitä, että on vain yksi URL, joka muuntautuu näyttökoon perusteella. Tämä toteutetaan asettamalla sivun HTML-koodin sekaan meta viewport tag, joka ohjaa selainta muuttamaan sivuston skaalautumista päätelaitteen näyttökoon mukaan (ks. kuvio 2). (Responsive Web Design 2021.)

Kuvio 2. Meta viewport tag.

Tämän jälkeen sivuston elementtejä ohjataan CSS:n media queryjen avulla muuntautumaan näyt-tökoon perusteella. Näistä myöhemmin lisää.

2.3 Optimointi

2.3.1 Koodin optimointi

Yleisesti koodatessa, kuten myös CSS-tyylittelyssä, on hyvä pitää mielessä yleiset nyrkkisäännöt, jotka ovat DRY (Don't Repeat Yourself) ja KISS (Keep It Simple Stupid). DRY:n periaate, nimensä mukaisesti, on itsensä toistamisen välttäminen. DRY:n noudattaminen vähentää koodin määrää ja tekee koodista modulaarisen. Käytännössä koodia palastellaan pieniksi ja uudelleenkäytettäviksi kokonaisuuksiksi. Esimerkiksi toiminnallisuuden tai fonttikoon muuttuessa, sitä muutetaan vain yhdestä paikasta, jolloin se muuttuu kaikissa paikoissa samanaikaisesti, missä muutettavaa asiaa on käytetty. KISS tähtää koodin luettavuuteen ja ymmärrettävyyteen. Usein ongelmiin on monta ratkaisuvaihtoehtoa, ja näistä tietenkin kannattaa valita selkein ja lyhyin vaihtoehto. Tästä on etenkin hyötyä silloin, kun työskennellään tiimissä, jolloin useampi henkilö saattaa muokata samaa koodia. (Baghel 2018.)

Koodin järjestäminen samalla tavalla jokaisessa tiedostossa edesauttaa koodin lukemista ja pitää sen yhdenmukaisena. Tämä voi olla kuitenkin haastavaa, koska muistettavaa tulee todella paljon. Tähän avuksi on keksitty virheiden lintterit (eng. linter). Lintterit ovat virheenkorjaustyökaluja. Ne ilmoittavat koodin epäkohdista ja saattavat automaattisesti korjata tiettyjä asioita. Esimerkiksi tässä opinnäytetyössä käytetään stylelinttiä ja sen lisäosia, jotka ilmoittavat CSS-tyylien virheistä. Lintereitä löytyy käytännössä joka lähtöön ja ohjelmointikieleen.

2.3.2 Suorituskyvyn optimointi

Suorituskyvyllä tarkoitetaan sivujen latausnopeutta. WordPress-sivujen latausnopeuteen vaikuttavat useat asiat kuten välimuistin käyttö, tiedostojen ja koodin pakkaus, palvelimen suorituskyky ja lähes kaikki palvelimiin liittyvä. Nämä asiat sivuavat tämän opinnäytetyön aihetta eli teemakehitystä, joten perehdytään suorituskykyä nostattaviin tekijöihin front-end-kehityksen näkökulmasta.

Valmisteemoissa tulee usein paljon ylimääräisiä tyylejä, jotka paisuttavat teemapaketin kokoa. Pahimmillaan teemat ovat useiden megatavujen kokoisia, mikä auttamatta hidastaa verkkosivujen latausnopeutta. Tämän vuoksi suorituskykyistä WordPress-teemaa tehdessä on syytä tehdä teema itse. On tärkeää strukturoida koodi oikein soveltaen aiempaa DRY-nyrkkisääntöä. Koodin oikealainen strukturointi ja modularisointi vähentää parhaimmillaan satoja, ellei tuhansia koodirivejä. Näin tyylien jalanjälki minimoidaan jo ennen varsinaista tiedostojen minimoimista ja pakkausta.

Kuvat ovat valtava osa moderneja verkkosivuja. Hyvät kuvat tuovat verkkosivuille eloa ja nostavat yleistä ilmettä. Korkealaatuiset kuvat ovat kuitenkin usein suurikokoisia ja kun niitä on viljelty verkkosivut täyteen, heikkenee sivujen suorituskyky.

Ennen kuvien minimoimista on syytä valita oikeanlainen tiedostomuoto kuvan tarkoituksen perusteella. PNG-kuvat tarjoavat parhainta laatua tiedostokoon kustannuksella ja tämän koon takia PNG-kuvia tulee välttää. PNG-kuvia on välttämätöntä kuitenkin käyttää, mikäli kuvan on tarve olla läpinäkyvä. Tavallisten kuvien kohdalla kannattaa suosia pienempikokoista JPEG-formaattia. SVG-kuvia kannattaa käyttää kaikkialla missä se on vain mahdollista, koska se on formaateista kokotehokkain.

Kuvat ja SVG-ikonit ovat syytä optimoida ennen sivuille lisäämistä. Optimoinnilla lasketaan kuvan laatua, jolloin tiedostokoko pienenee. Optimointityökaluja on useita ja ilmaisia löytää internetistä. Tässä työssä käytetään Optimage-nimistä sovellusta. Optimage on todettu käytössä toimivaksi ja se saattaa pudottaa alkuperäisen tiedostokoon kymmenesosaan, pitäen samalla kuvanlaadun visuaalisesti hyvänä.

Merkittävä sivuston suorituskykyä edistävä tekijä on kuvien lataamisessa lazy loading-tekniikan käyttö. Tällä tekniikalla ladatut kuvat ladataan vasta, kun niitä tarvitaan. Esimerkiksi sivun footerissa sijaitsevat kuvat eivät lataudu heti kun sivu ladataan ensimmäisen kerran, vaan vasta sitten kun niiden kohdalle scrollataan. Lazy loading löytyy tässä opinnäytetyössä käytettävästä Air-light aloitusteemasta ja siihen liittyvästä pluginista, air-helperistä. WordPressin mukana tulee myös siihen sisäänrakennetu lazy loading.

2.4 Saavutettavuus

Saavutettavasta verkkosivustosta voidaan puhua silloin, kun sivustoa pystyy käyttämään laaja kirjo ihmisiä riippumatta siitä, onko heillä toimintarajoitteita tai käyttöä estäviä vammoja. Täten verkkosivua toteuttaessa on tärkeää ottaa huomioon mahdolliset kuulo-, näkö-, toiminta ja liikuntavammat. Verkkosivujen saavutettavuus on kaikille hyödyksi, koska ikääntymisen myötä jokaisen toimintakyky heikkenee.

Verkkosivujen saavutettavuus voidaan jakaa neljään osioon: havaittavuuteen, käytettävyyteen, ymmärrettävyyteen ja kestävyyteen.

Havaittavuudella viitataan sivuston sisältöön ja siihen kuinka esimerkiksi heikkonäköinen pystyy ymmärtämään kuvan sisältöä tai kuuro ymmärtämään videosisältöä. Hyvän havaittavuuden takaamiseksi on tärkeää lisätä vaihtoehtoiset tekstit kuville ja muulle tekstittömälle sisällölle sekä multimediasisällölle tekstitykset ja mahdollisesti kuvailevat selostukset. Kontrasti tekstin ja taustan välillä on oltava riittävä ja elementtejä ei tulisi erotella vain värin perusteella. Esimerkiksi linkkejä ei tule erottaa vain värillä, vaan värin lisäksi vaikkapa alleviivaamalla. Tekstin koko tulisi olla riittävä, minimissään 12px. Rakenteeltaan korrekti HTML on tärkeää, jotta sisältö jakautuu järkeviksi kokonaisuuksiksi. Esimerkkinä tästä on otsikoiden h1, h2, h3 käyttö HTML-markupissa porrastetusti. (Zahra 2019.)

Käytettävyydellä tarkoitetaan kykyä selata verkkosivuja rajoitetulla mahdollisuudella käyttää esimerkiksi hiirtä. Hyvän käytettävyyden saavuttamiseksi on suotavaa testata sivujen selaamista pelkällä näppäimistöllä käyttäen tab-näppäintä. Muita esteitä käytettävyydelle on liiallinen ärsykkeiden, kuten animaatioiden tai voimakkaiden efektien käyttö. Ärsykkeistä tulisi varoittaa etukäteen ja tarjota vaihtoehto ottaa ne pois päältä. (Zahra 2019.)

Hyvällä ymmärrettävyydellä tähdätään siihen, että mahdollisimman laaja käyttäjäkunta pystyy sen ymmärtämään. Sisällön tulee olla riittävän yksinkertaista ja säännöllisenä pysyvä navigaatio sekä pakoreitit ovat olennaisia ymmärrettävyyden edistäjiä. Näiden lisäksi riittävät ohjeet sivuston käyttön ja selkeät virheilmoitukset ovat tärkeitä. (Zahra 2019.)

Kestävyydestä puhutaan kun, verkkosivut ovat yhteensopivia nykyisten selainten ja avustavien välineiden kanssa. Kestävyyden varmistamiseksi on hyvä käyttää koodatessa virheiden korjaajia eli linttereitä tai ajaa tiedostot lopuksi validaattorin läpi. (Zahra 2019.)

Saavutettavuutta mitataan WCAG (Web Content Accessibility Guidelines) mukaan. WCAG:n mukaan on olemassa kolmea eri saavutettavuuden tasoa: A, AA ja AAA. A sisältää vain välttämättömät asiat ja kaikki avustavat välineet eivät välttämättä ymmärrä sivun sisältöä. AA on taso, johon tulee tähdätä teemaa tehdessä ja se kertoo sivun täyttävän aiemmin mainitut saavutettavuuden osiot. AAA kertoo erityisestä saavutettavuudesta tietylle käyttäjäryhmälle. (A11Y Checklist n.d.) Muistettavia asioita AA-tason saavuttamiseksi on kohtalaisen paljon, joten teeman kehittämisen loppupuolella on hyvä tapa käydä läpi tarkistuslista saavutettavuuteen liittyvistä asioista. A11Y on yhteisöprojekti, jonka tavoitteena on tehdä verkon saavutettavuudesta helpompaa. A11Y-projekti on koonnut kattavan listan asioista, jotka tulisi tarkistaa ennen julkaisua. Muita saavutettavuuden parantamiseen liittyviä työkaluja ovat lintterit, jotka huomauttavat jo koodatessa virheistä esimerkiksi liian pienestä fontista sekä värityökalut, jotka kertovat ovatko kontrastit riittäviä.

2.5 CSS

Cascading Style Sheets eli CSS on erityisesti WWW-sivuille tarkoitettu tyyliohjeita määrittelevä kieli. Käytännössä CSS:ssä määritellään, miltä jokin esimerkiksi HTML:llä kirjoitettu palikka tai teksti näyttää verkossa käyttäjälle (ks. kuviot 3 ja 4). Määrittelyä on laidasta laitaan, tekstin kursivoittamisesta elementtien animoimiseen. CSS on front-end-kehittäjän tärkeimpiä työkaluja ja sitä kehittää the World Wide Web Consortium (W3C). Tämä osio ei suinkaan opeta kuinka kirjoitetaan CSS:ää alusta alkaen, vaan käy läpi tehokkaita ja välttämättömiä CSS:n kulmakiviä verkkosivua toteuttaessa.

Kuvio 3. WorkPower etusivu ilman CCS:ää.

Kuvio 4. WorkPower etusivu CCS:n kanssa.

2.5.1 Muuttujat

Tyylitiedostot paisuvat jo pienemmissäkin sivustoissa valtaviksi. Usein tämä johtaa koodin toistamiseen. Esimerkiksi tiettyä sinisen sävyä voidaan käyttää kymmenessä eri kohdassa ja kun väri halutaan vaihtaa hivenen tummemmaksi, joutuu erikseen vaihtamaan nämä kymmenen väriä hieman tummemmaksi. Tätä varten on kehitetty muuttujat. Muuttujat (eng. variables) ovat useille tuttuja eri ohjelmointikielistä. Muuttujalle voidaan antaa arvoja ja se varastoi arvot myöhempää käyttöä varten. Arvoja muuttujan sisällä voidaan manipuloida komennoilla. Tyylejä kirjoittaessa muuttujat eivät ole pakollisia samalla tavalla, kuin useissa ohjelmointikielissä. Ilman muuttujia pärjää, mutta niiden käyttämättä jättäminen tekee koodista vaikealukuisemman ja heikommin modulaarisen. CSS-muuttujat ovat kohtalaisen uusi asia. Alkuperäinen implementaatio tuli vuonna 2012 Firefoxiin ja Chromeen, mutta se sai varsinaisen selaintuen vasta vuoden 2016 aikana (Gash 2017). Nykyään yli 91 % selaimista tukee CSS-muuttujia (Can I use 2021).

Syntaksiltaan CSS-muuttujat ovat yksinkertaisia. Suositeltavaa on määrittää muuttuja juureen, jolloin se on käytettävissä kaikissa dokumentin elementeissä. Kuvion 5 esimerkissä väri on määritetty samassa tiedostossa, jossa on myös elementtejä. Oikean käytänteen mukaisesti värit määriteltäisiin erillisessä tiedostossa.

Kuvio 5. CSS-muuttujan syntaksi.

Kuvassa käytetty väri voisi olla aiemman esimerkin mukaisesti käytössä kymmenessä paikassa, mutta käyttämällä muuttujia, voi väriä vaihtaa samanaikaisesti kymmeneen paikkaan muuttamalla vain yhtä riviä. Muuttujilla toteutetaan usein nykyään suosiossa olevat vaihtoehtoiset tummat teemat. Muuttujia voi käyttää useisiin tarkoituksiin ja ne ovat todella tehokas tapa pitää suurikin tyylien kokonaisuus hallinnassa (ks. kuvio 6).

Kuvio 6. Esimerkkejä CSS-muuttujista.

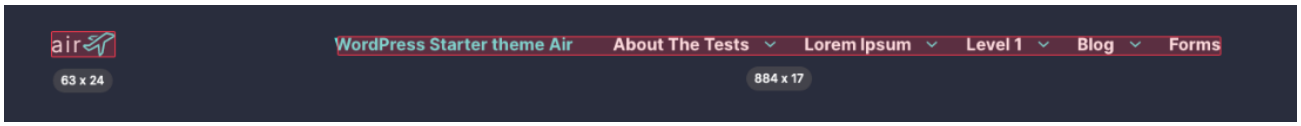
2.5.2 Flexbox & CSS Grid

Ei voi sanoa kirjoittavansa nykyaikaista CSS:ää, jos ei käytä flexboxia (lyh. flex) tai CSS-gridiä (lyh. grid). Yhdessä flexillä ja gridillä tekee tekstien ja elementtien sijoittelemisesta helppoa. Flex ja grid ovat myös erityisen tärkeitä responsiivisten verkkosivujen toteuttamiseen.

Flexbox sai alkunsa vuonna 2009, kun ensimmäinen toimiva versio julkaistiin. Tämä versio oli kuitenkin suorituskyvyltään huono ja oli riippuvainen vanhanaikaisista tekniikoista. Flexbox kirjoitettiin uudelleen 2011 ja se sai vuonna 2012 tuen useimpiin selaimiin. (Ogidan 2018.) Huhtikuussa 2021 noin 98 % selaimista tukee flexboxia (Can I use 2021).

Flexin pääasiallinen tarkoitus on sijoitella elementtejä (flex item) niitä ympäröivän elementin (flex container) sisällä. Flexiä käytetään monissa paikoissa pitkin tyylitiedostoja, mutta yleisimpiä tapauksia ovat navigaatiot, footerit ja muut pienemmät kokonaisuudet, joissa gridin käytölle ei ole tarpeeksi suurta perustetta (ks. kuviot 7, 8 ja 9).

Kuvio 7. Linjassa olevien svg-kuvien keskittäminen ja tasaus onnistuu Flexboxilla.



Kuvio 8. Navigaation asettelu Flexboxilla.

Kuvio 9. Pala navigaation lähdekoodia.

Huhtikuun loppupuolella 2021 tätä opinnäytetyötä kirjoittaessa, flexboxin ominaisuus lisätä rivi- ja palstavälit, sai selaintuen moderneihin selaimiin (Can I use 2021). Tämä lisäys tekee flexistä entistäkin tehokkaamman ja monipuolisemman työkalun. Elementtien välistys oli ennen flexin kompastuskivi, koska sen joutui tekemään marginilla tai paddingilla. Tämä aiheutti ongelmia tilanteissa, joissa elementtien on tarkoitus wrapata seuraavalle riville resoluutiota pienentäessä. Tämä ominaisuus myös vähentää CSS-gridin käyttötarkoituksia.

CSS-gridillä jäsennellään sivuston elementit haluttuun muotoon. Nimensä mukaisesti elementit voidaan asetella kaksiulotteiseen ruudukkoon kuten kuvioista 10 ja 11 näkyy. Grid sai natiivin tuen useimmille selaimille 2017 aikana (House 2021). 2021 huhtikuussa gridiä tukee 95 % selaimista (Can I use 2021).

Kuvio 10. CSS-gridillä muotoiltu lohko.

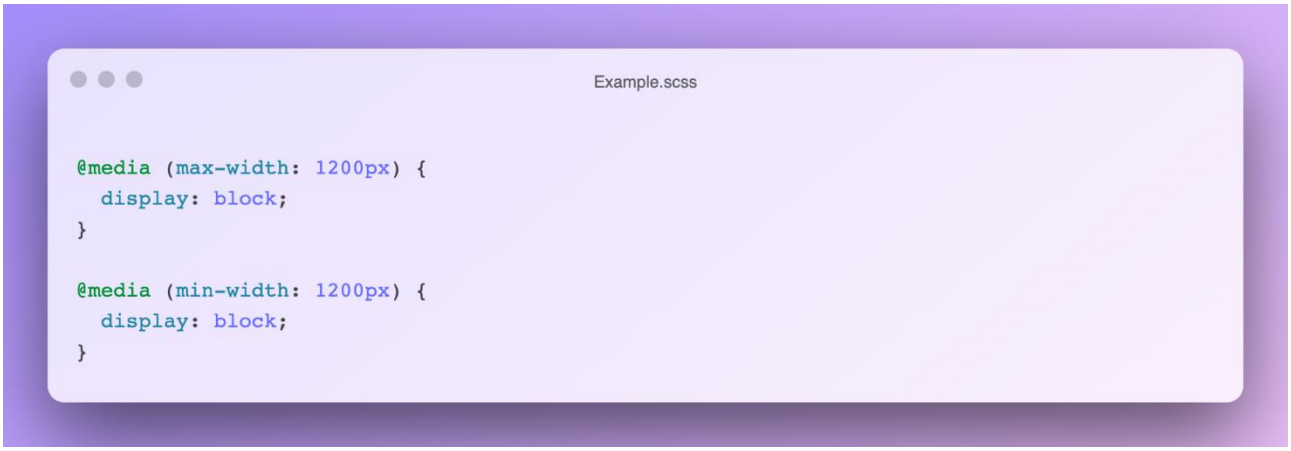
Kuvio 11. CSS-gridillä muotoiltu artikkelit-lohko.

Gridiä käyttäessä tulee ottaa huomioon responsiivisuus, koska gridillä tehdyt ruudukot eivät pysty hajoamaan automaattisesti uudelle riville näyttökokoa pienentäessä, toisin kun flexillä tehdyt pystyvät. Grid on vahva työkalu ja sillä saa jäsenneltyä semanttisesti todella huonoa, esimerkiksi integraation mukana tullutta, HTML-markuppia järkevään muotoon. Air-light aloitusteemassa on valmiina gridin käyttöä nopeuttava Sass-mixin.

2.5.3 Media Query

Media queryjen avulla voidaan säädellä sitä, miltä sivusto näyttää päätelaitteen ominaisuuksien ja olemassa olevien parametrien tai arvojen mukaan (ks. kuvio 12). Media queryt ovat avain responsiivisiin sivuihin. Responsiivisuuden, joka ottaa huomioon näytön koon, lisäksi media queryillä voidaan ottaa huomioon käyttäjän asettamat saavutettavuusasetukset. Media queryillä voidaan esimerkiksi poistaa animaatiot tai lisätä kontrastia käytössä olevien saavutettavuusasetuksien perusteella (Using Media Queries for Accessibility 2021).

Pääasiassa media queryjä käytetään säätelemään elementtien ulkonäköä tietyllä resoluutiovälillä. Esimerkiksi kuvion 11 CSS-gridillä tietokoneen näytölle tyylitelty kolmen sarakkeen ruudukko tulee muuttaa kahden sarakkeen ruudukoksi välimallin resoluutioihin, kuten iPadiin, siirryttäessä. Tämä tehdään media queryllä.



Kuvio 12. Media query-syntaksi

Media queryjen kohdalla tehdään päätös, aloitetaanko kehitys suuri- vai pieniresoluutioisista laitteista. Käyttämällä min-widthiä-parametria aloitetaan kehitys pienemmästä näyttökoosta edeten suurempiin. Tässä kohtaa tulee miettiä, haluaako kirjoittaa tyylit mobile first-ajattelulla. Kuten aiemmin jo todettiin, mobile first harvoin säästää koodirivejä. Mikäli lopputulos on paras mahdollinen, kummin päin tyylit kirjoittaa, ei ole väliä.

2.5.4 CSS framework

CSS-frameworkit (suom. kehykset) ovat valmiita tyylitiedostokokoelmia (Lawrence 2019). Frameworkeista löytyy yleisiin palikoihin, kuten navigointiin, footeriin tai vaikka kolmen sarakkeen sisältöön, valmiit tyylit ja yleensä myös responsiivisuus sisältyy. Frameworkit ovat joko ladattavia tiedostoja tai ulkoisia tiedostoja, jotka voidaan linkittää suoraan HTML:n joukkoon. Tyylejä käytetään rakentamalla HTML-markup niiden mukaisiksi ja laittamalla elementeille oikeat class-määritykset. Yksi suosituimmista CSS-frameworkeista on Bootstrap.

Ongelmana frameworkeissa on se, että niiden mukana tulee usein paljon ylimääräistä koodia, jota ei koskaan tule käytettyä. Myöskin käyttämällä frameworkkeja menettää täyden kontrollin tyyleistä ja modulaarisuudesta. Frameworkkia käyttäessä lähes aina tulee vastaan tilanteita, joissa joutuu ylikirjoittamaan kehyksen tyylejä ja tekemään tyylejä itse. Tehtäessä täysin räätälöityä teemaa, kontrollin menettäminen ja turhan koodin mukana raahaaminen ei ole ideaalista. Ilman frameworkkia tulee myös vapaus pitää HTML erillään tyylimäärittelyistä. Frameworkit pohjautuvat class-määrityksiin ja tällä tavoin ohjaavat HTML-markupin rakennetta. Myöskin jatkuva CSS:n kehitys, etenkin gridin ja flexin, on helpottanut valtavasti tyylien kirjoittamista ilman frameworkkeja. Vaarana valmiiden tyylien käytössä on oppimisen pysähtyminen, kopioimalla toisen tyylejä ei opi. Näiden syiden vuoksi räätälöityä tai vähemmänkin räätälöityä teemaa kirjoittaessa on suotavaa tehdä tyylit täysin itse alusta loppuun tai kuten tässä työssä tullaan tekemään, minimalistisen aloitusteeman päälle.

2.6 Sass

Tässä työssä käytetään apuna Sassia tapana kirjoittaa tyylejä. Sass eli Syntactically Awesome Style Sheets on CSS-esiprosessori (eng. preproceccor). Esiprosessori kääntää tässä tapauksessa Sassilla kirjoitetusta tiedostosta CSS-tiedoston, jota voi käyttää normaalin tyylitiedoston tapaan verkkosivuilla (Sass Basics n.d). Sass lisää tyylien kirjoittamiseen helpottavia ja nopeuttavia ominaisuuksia. Sass muistuttaa vahvasti CSS:ää, joten Sassin käyttöönotto ei ole vaikeaa, jos CSS on entuudestaan tuttu. Tässä kappaleessa käydään läpi Sassin keskeisimpiä ominaisuuksia, jotka ovat hyödyllisiä lähes jokaisessa projektissa.

Sassia voi kirjoittaa kahdella eri tavalla: SCSS tai Sass. Syntaksiltaan nämä muistuttavat toisiaan, erona se, että Sass-syntaksista on karsittu pois sulkumerkit ja puolipilkut. Tässä työssä käytetään SCSS-tapaa, koska se muistuttaa enemmän traditionaalista CSS:ää.

2.6.1 Sass variables

Aiemmin käytiin läpi CSS:n natiivit muuttujat ja niiden ominaisuudet. Sassilla on omat muuttujansa. Syntaksissa ne merkitään $-merkillä kuten kuvioissa 13 ja 14 näkyy. Sassin muuttujat toimivat hyvin samalla tavalla, kuin CSS:n natiivit. Ero CSS:n natiiveihin muuttujiin on se että, Sassin muuttujia ei voida manipuloida tai muuttaa. Tästä herää kysymys, kumpia muuttujia tulisi käyttää. Nyrkkisääntönä yleisesti kehittäessä, kuten myös tässä muuttujien valinnassa onkin se että, natiiveja ominaisuuksia tulisi suosia. Natiiveja muuttujia tulee siis priorisoida, mutta Sassin omille muuttujille on omat paikkansa. Saadakseen kaiken hyödyn irti muista Sassin ominaisuuksista, on välillä välttämätöntä käyttää sen omia muuttujia. (Laukkarinen 2021.)

```
$font-stack:    Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

Kuvio 13. Sass-syntaksi (Sass basics n.d)

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Kuvio 14. SCSS-syntaksi (Sass basics n.d)

2.6.2 Mixins

Mixin on joukko CSS-määrityksiä, joita voidaan kutsua ympäri tyylitiedostoja. Mixin määritellään @mixin ja sitä kutsutaan @include. Mixinien maailma on laaja ja niitä voi soveltaa todella moneen eri asiaan. Elementille, joka toistuu usein ympäri sivua, on perusteltua tehdä mixin.

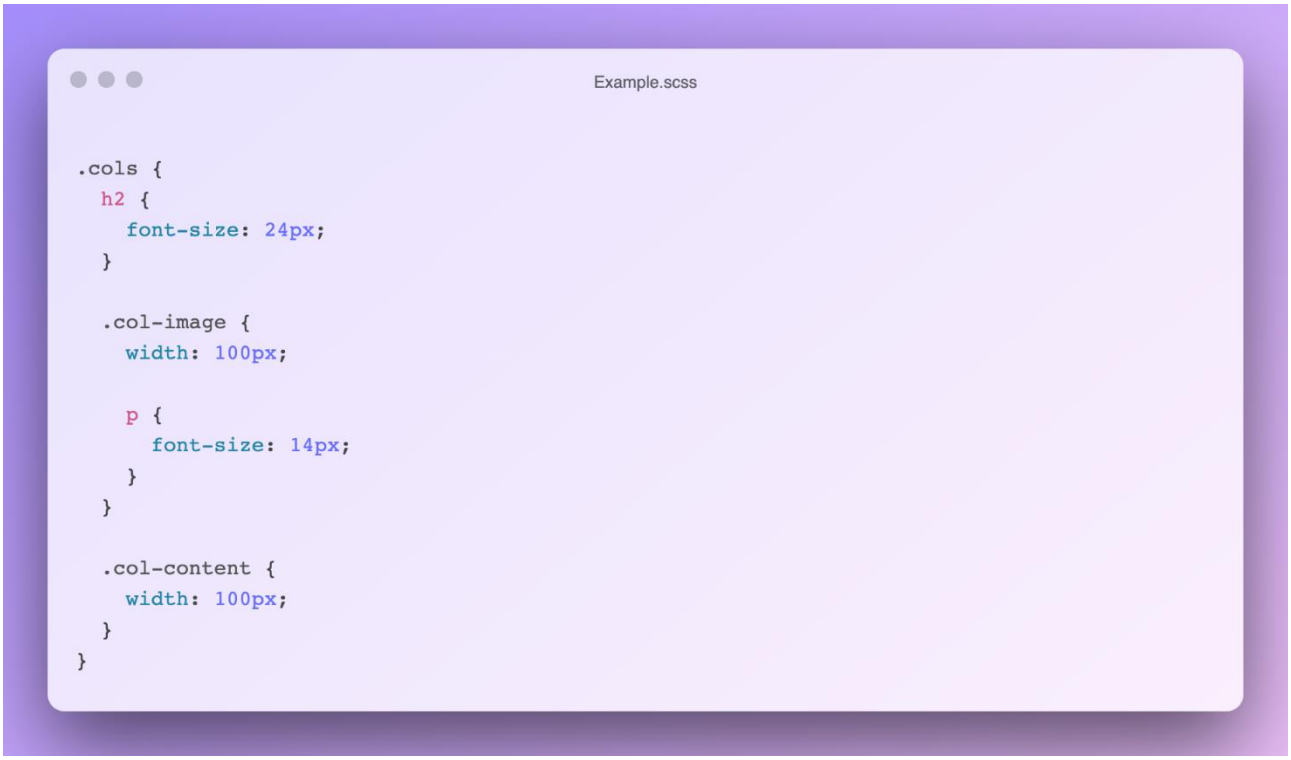
Kuviossa 15 on tehty alkeellinen mixin. Mixinin nimi on example-button. Mixiniä tehtäessä, voi sille asettaa parametreja, joita se ottaa vastaan. Tässä vastaanotettavaksi parametriksi on laitettu $color_bg, joka ottaa vastaan taustavärin. Kutsuessa mixiniä, voi sille antaa parametreja, kuten button-1 kohdalla on tehty ja tässä tapauksessa parametriksi annettu väri saa nappulan vaihtamaan taustavärin valkoiseksi. Mikäli mixiniä kutsuessa ei ole annettu parametreja, käyttää mixin sille alkuperäisesti määritettyjä arvoja. Kuvion 15 tapauksessa button-2 on musta.

Mixin on yksi niistä asioista, joissa on pakko käyttää Sassin omia muuttujia. Natiivia muuttujaa ei voi asettaa parametriksi, mutta parametrille voi antaa arvoksi natiivin muuttujan kuten kuviossa 15 on annettu.

Kuvio 15. Mixinin määritys ja kutsunta

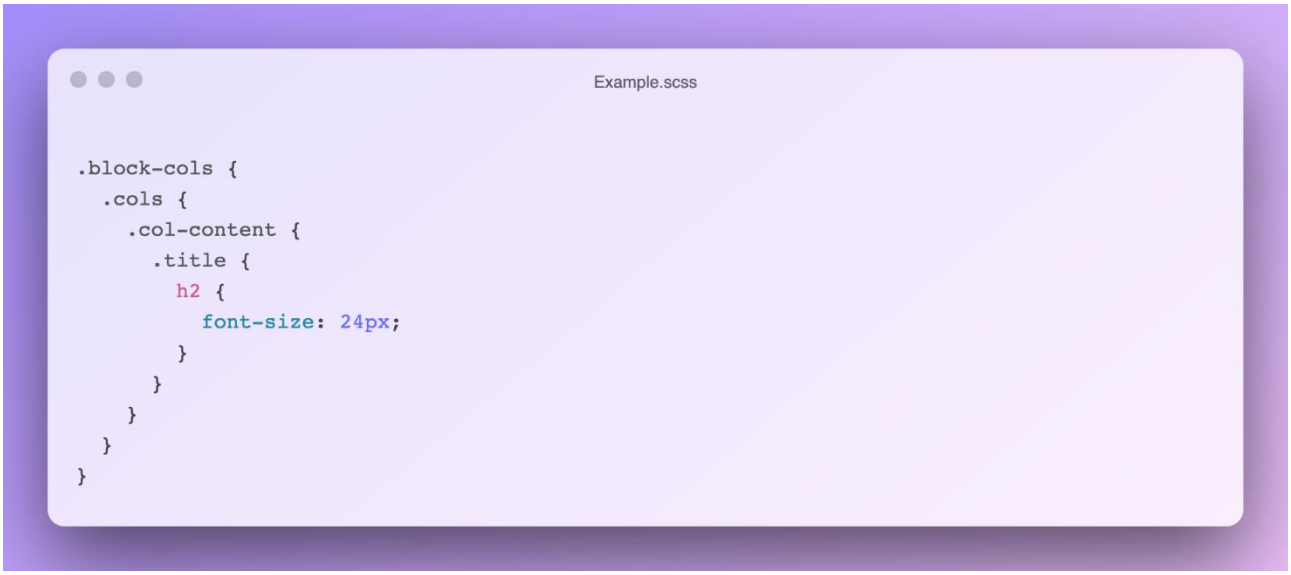
2.6.3 Nesting

Vakiona CSS ei sisällä minkäänlaista hierarkiaa kuten esimerkiksi HTML. Sass lisää mahdollisuuden “nestata” (eng. nesting) eli lisätä hierarkiaa CSS:ään. Nestaamisen avulla pystyy niputtamaan koodia pieniksi kokonaisuuksiksi, mikä edesauttaa koodin lukua ja järjestelemistä (ks. kuvio 16).



Kuvio 16. Esimerkki nestaamisesta

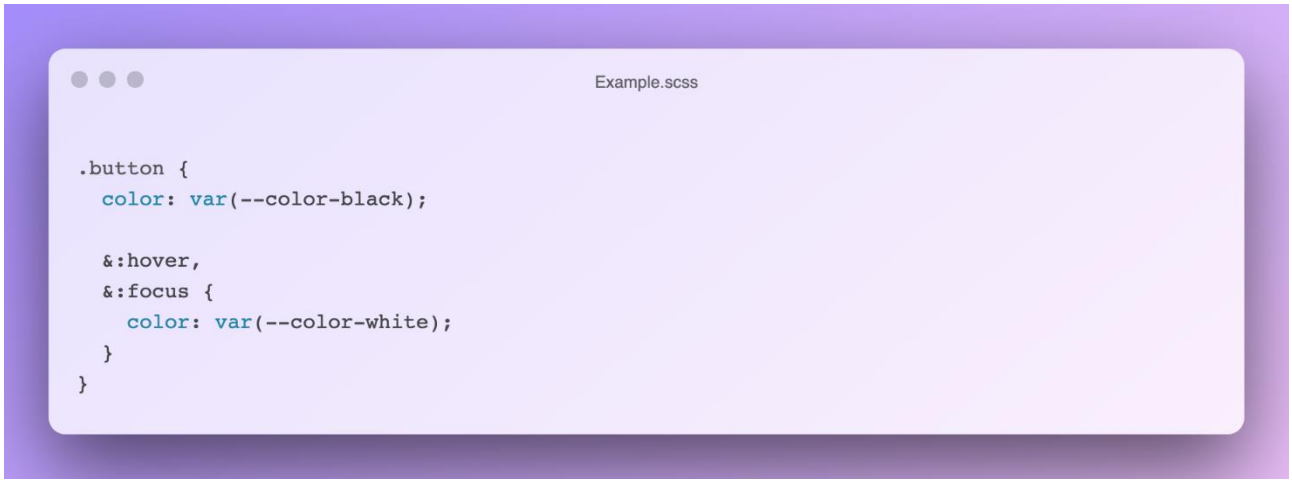
Nestaamisen kanssa on oltava varovainen. Liian syvälle hierarkiassa meneminen vaikeuttaa koodin ylläpitoa ja on huono käytäntö kuten kuviosta 17 näkyy. Koodia kirjoittaessa on hyvä pitää nyrkkisääntönä menemättä kolmea tasoa syvemmälle.



Kuvio 17. Esimerkki huonosta nestauksesta

2.6.4 Parent selector

Sassin mukana tulee parent selector (suom. valitsin), jonka avulla pystyy uudelleenkäyttämään parent selectoria ympäröivää classia (ks. kuvio 18). Parent selectorin merkki on &. Parent selector on tehokas tapa lisätä pseudoelementti sen ympäröivään classiin.



Kuvio 18. Parent-selector esimerkki

2.7 Air-light

Air-light on Duden kehittämä aloitusteema. Se tähtää minimalistisuuteen sekä yksinkertaisuuteen. Air-light on hyväksytty WordPressin teemavalikoimaan. Ideana Air-lightissa sen yhdenmukaisuus alkuperäisen WordPressin kanssa ja se, että kaikki turha on karsittu pois. Käytännössä kaikki, mitä aloitusteema sisältää, tulee käytettyä jossain välissä teemakehitystä. Air-light käyttää WordPressin alkuperäisiä ohjelmointikieliä PHP:ta, JavaScriptia ja CSS:ää. On pidettävä mielessä se, että aloitusteema on muovattu sopimaan Duden tapaan tehdä verkkosivuja, joten se ei välttämättä ole juuri sopiva ratkaisu sinun omaan projektiisi. Air-lightia tukee Duden tekemä plugin air-helper, joka on tarkoitettu kulkemaan yhdessä aloitusteeman kanssa. Air helper sisältää keskeisiä toiminnallisuuksia kuten kuvien lazy loadingin. Air-lightin kansiorakenne muistuttaa klassista WordPress-teeman rakennetta. (ks. Liite 1).

Air-lightin kehityksessä on pidetty mielessä aiemmin mainitut KISS ja DRY. Koodin määrä on jätetty minimiin ja tämän vuoksi aloitusteeman koko on vain 140 kilotavua ja pakattuna hieman alle 30

kilotavua. Tietysti koko kasvaa teemaa räätälöitäessä ja riippuen verkkosivujen laajuudesta, nousee koko usein 200–500 kilotavuun.

Aloitusteeman erityisiin ominaisuuksiin kuuluu sen saavutettavuus. Air-light on saavutettavuutensa ansiosta läpäissyt WordPressin testin aiheesta, ja sen vuoksi omaa saavutettava-avainsanan WordPressin teemavalikoimassa.

3 WorkPower

Tässä opinnäytetyössä toteutetaan räätälöity teema WorkPowerin uusille sivuille. WorkPower on henkilöstön rekrytoinnin, vuokrauksen ja ulkoistuksen asiantuntija. Opinnäytetyössä keskitytään lähinnä tyylien kirjoittamiseen, mutta on kuitenkin syytä selittää yleiseen kehitykseen liittyvät asiat.

3.1 Duden tapa tehdä WordPress-sivustoja

Tyylejä kirjoitetaan suunnittelijoiden tekemien layouttien pohjalta. Suunnittelija suunnittelee ulkoasun prototyyppityökalu Figmaan asiakkaan yrityksen, kohderyhmän ja toiveiden perusteella. Tämä tarkoittaa sitä että, käyttöliittymä ja käyttäjäkokemus ovat mietitty etukäteen, ennen koodauksen aloittamista. Tällöin kehittäjien ei tarvitse miettiä esimerkiksi värien tai fonttien valitsemista. On kuitenkin asioita, joita ei näy välttämättä suunnittelijan layoutista. Tällaisia asioita ovat esimerkiksi elementtien hover- ja focus-tyylit, animaatiot ja transitiot. WorkPower erityisesti toivoi pientä liikkuvuutta sivuihinsa animaatioiden muodossa. Sivuston toiminnallisuudesta ja rajapintaintegraatioista vastaa back-end kehittäjät. Front-end-kehittäjä tekee HTML-markupin templateihin ja tyylittelee sivustosta suunnittelijan layoutin mukaisen.

Dude tekee asiakaskohtaisia modulaarisia lohkoja. Näihin lohkoihin asiakas pystyy itse lisäämään haluamansa sisällön ja rakentamaan sivuja WordPress-hallintapaneelin kautta (ks. kuvio 19). Lohkot tehdää ACF (Advanced Custom Fields) -pluginin avulla. Teknisesti puhuttaessa lohkoista, tarkoitetaan /template-parts/modules-kansion alle (ks. liite 1.) tehtäviä moduuleita. Näihin moduuleihin rakennetaan ACF:än avulla toiminnallisuus, jotta asiakas pystyy muokkaamaan niiden sisältöä.

images-two.php

```php
<?php
 @package workpower

$image_left = get_sub_field( 'image_left' );
$image_right = get_sub_field( 'image_right' );

if ( empty( $image_left ) || empty( $image_right ) ) {
  return;
}
?>

<section class="block block-two-images">
  <div class="container">
    <div class="cols">

      <div class="image-small has-lazyload" aria-hidden="true">
        <?php vanilla_lazyload_div( $image_left ); ?>
      </div>

      <div class="image has-lazyload" aria-hidden="true">
        <?php vanilla_lazyload_div( $image_right ); ?>
      </div>

    </div>
  </div>
</section>
```

Kuvio 19. WorkPower images-two.php-lohkon lähdekoodi

Tiedosto content-modular.php (ks. Liite 1.) kutsutaan kuvion 1 tavalla template-tiedostossa, jolloin se näyttää valittujen moduulien sisällön. Tiedosto content-modular.php ajaa funktion, joka käy läpi template-parts/modules/ kansion ja tarkistaa onko ACF:ään luodulla asettelulla olemassa moduuli vai ei ja näyttää sitten automaattisesti moduulia vastaavan sisällön sivulla. Moduulien määrä vaihtelee projekteittain, WorkPowerin kohdalla niitä on suunniteltu kuusitoista.

Artikkeleita muokataan WordPressin Gutenberg-editorin avulla. Gutenberg editorin lohkoihin tehdään myös asiakaskohtaiset tyylit.

3.2 Yleistä teemakehityksestä

Tyylittelyn prosessia on helpompi seurata tutustumalla ensin Air-light aloitusteeman rakenteeseen ja seuraamalla kuvioiden tiedostonimiä.

Kaikki /sass-kansion alla sijaitsevat tiedostot kutsutaan _global.scss-tiedostoon (ks. kuvio 20). Tämä mahdollistaa koodin pilkkomisen pieniksi kokonaisuuksiksi ja linkittää tiedostot toisiinsa mahdollistaen muuttujien ja mixinien käytön kaikissa kutsutuissa tiedostoissa.

Kaikki kuvat, jotka lisätään projektin aikana, ajetaan Optimagen läpi ennen kuin ne lisätään sivulle. Svg-kuvat laitetaan /svg-kansion alle. SVG-kuvien koodin seassa on usein paljon turhia kooditageja, jotka SVG-kuvan tekemiseen käytetty ohjelma on jättänyt. Nämä kooditagit ovat helppo poistaa manuaalisesti, mutta tehokkaampaa on käyttää SVG-kuvien siistimistä ja minimoimista varten kehitettyä tyokalua. Suositeltava ja tässä projektissa käytetty työkalu on SVGO.

_global.scss

```
// Variables
@import 'variables/colors';
@import 'variables/breakpoints';
@import 'variables/spacings';
@import 'variables/font-face';
@import 'variables/font-family';
@import 'variables/font-sizes';
```

Kuvio 20. Palanen WorkPower _global.scss-tiedostoa.

3.3 Muuttujat

Uutta projektia aloitettaessa rakentamaan aloitusteeman päälle, aloitetaan laittamalla muuttujat paikoilleen. Muuttujia käytetään käytännössä jokaisessa teeman palasessa, joten ne on hyvä laittaa kuntoon heti alussa.

Fontit hostataan (eng. host) itse laittamalla fonttitiedostot teeman /fonts-hakemistoon. Tällä vältetään riippuvuudet ulkoisiin fonttihosteihin. Hakemistoon laitetaan fonttitiedostot kolmessa eri muodossa ttf, woff ja woff2. Syynä tähän on laajemman selaintuen saaminen ja samalla pitäen nopean latausajan uudempien formaattien ansiosta moderneilla selaimilla. WorkPowerissa käytetään kahta eri fonttia otsikoihin ja leipätekstiin, sekä niiden italic-variaatiota ja useita eri lihavuusvariaatioita. Fonttivalinnat ovat ”Inter” leipätekstiin ja ”Barlow” otsikoihin (ks. kuvio 21).

_font-face.scss

```
@include fontFace('Inter', '../../fonts/Inter-Regular', 400);
@include fontFace('Inter', '../../fonts/Inter-Italic', 400, italic);
@include fontFace('Inter', '../../fonts/Inter-SemiBold', 600);
@include fontFace('Inter', '../../fonts/Inter-Bold', 700);
@include fontFace('Barlow', '../../fonts/barlow-v5-latin-regular', 400);
@include fontFace('Barlow', '../../fonts/barlow-v5-latin-600', 600);
```

Kuvio 21. Fonttien kutsunta.

Jo tässä vaiheessa tulee esiin mixinien hyödyllisyys. fontFace-mixinillä saadaan nopeammin fontit paikoilleen säästäen koodia (ks. kuvio 22). Kyseinen mixin löytyy aloitusteemasta, koska sitä käytetään jokaisessa projektissa.

_typography.scss

```
@mixin fontFace($family, $src, $weight: normal, $style: normal) {
  @font-face {
    font-display: auto;
    font-family: $family;
    font-style: $style;
    font-weight: $weight;

    src: url('#{$src}.woff') format('woff'), url('#{$src}.woff2') format('woff2'), url('#
{$src}.ttf') format('truetype');
  }
}
```

Kuvio 22. fontFace-mixin lähdekoodi.

Kutsutut fontit asetetaan muuttujiin tiedostossa _font-family.scss.

Värit lisätään CSS-muuttujina /variables/_colors.scss-tiedostoon. Väripaletti tulee suunnittelijoiden layouteista. Värien kohdalla on tärkeää tarkistaa taustan ja tekstin välinen kontrasti tilanteissa, joissa kontrastieroa ei näytä olevan riittävästi. Tällaisia tilanteita sattuu harvoin ja usein puutteen näkee paljaalla silmällä. Hyvä sovellus kontrastin tarkistamiseen macOS-käyttöjärjestelmällä on Pika. Pika kertoo kontrastisuhteen ja sen, mihin WCAG tasoon kontrasti riittää.

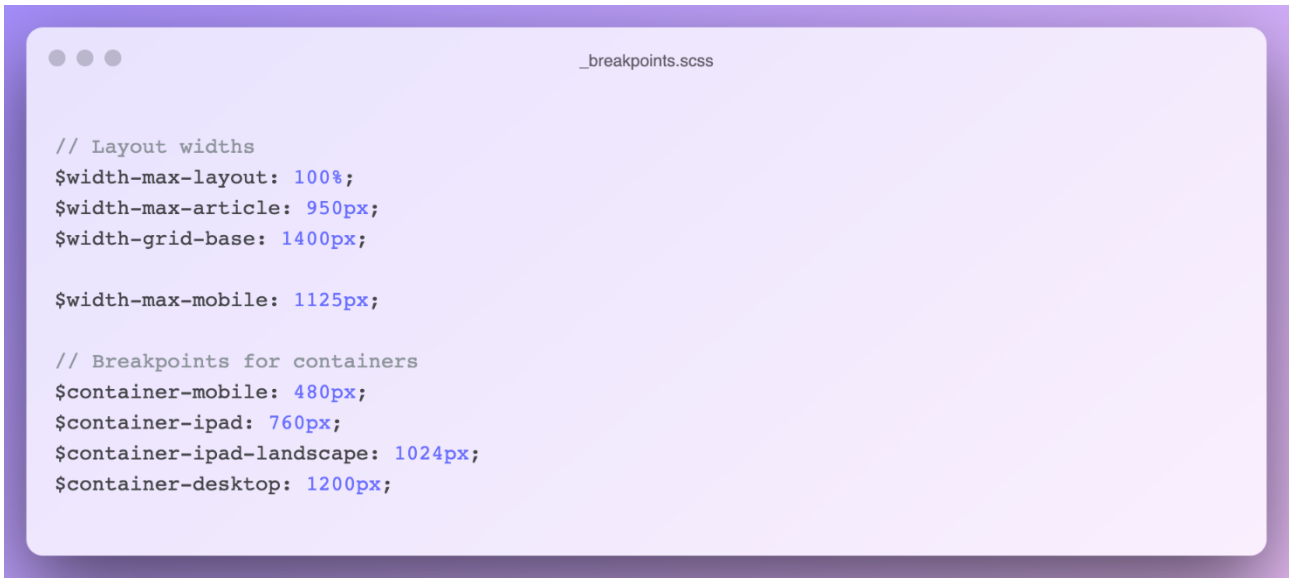
Air-lightissa on valmiiksi asetettuja muuttujia kuten kuviosta 23 näkyy. Color-main väri on asetettu valmiiksi esimerkiksi navigaatioon, jolloin asettamalla brändin värin ko. muuttujaan, saa vaihdettua sivustolle oleellisen elementin värin layoutin mukaiseksi. Vastaavia muuttujia löytyy myös muista /variables-kansion alla olevista tiedostoista.

_colors.scss

```
:root {
  // Brand color pool
  --color-black: #222;
  --color-dark: #222;
  --color-dark-grey: #343434;
  --color-dark-blue: #262c47;
  --color-black-rock: #2b303c;
  --color-blue-chalk: #ece6f2;
  --color-east-bay: #545773;
  --color-gray: #595959;
  --color-midnight-blue: #2b3151;
  --color-white: #fff;
  --color-brand-red: #e63029;
  --color-alto: #d6d6d6;
  --color-ebony: #0d1127;

  // Main element colors
  --color-heading: var(--color-dark);
  --color-main: var(--color-brand-red);
  --color-paragraph: var(--color-dark);
}
```

Kuvio 23. Värimuuttujien lisäys

Muita tärkeitä muuttujia fonttien ja värien lisäksi, mitkä tulisivat määrittää ennen varsinaista tyylittelyä ovat, sisällön ja artikkelien maksimileveys. Nämä määritellään _breakpoints.scss-tiedostossa kuten kuviosta 24 näkyy. Nämä muuttujat muuttavat keskeisten rakennuspalikoiden, blockien ja containereiden (ks. kuvio 26), leveyksiä. _breakpoints.scss-tiedosto sisältää myös muuttujia, jotka kuvastavat tiettyjä näyttökokoja. Näitä muuttujia on hyvä käyttää media queryjen yhteydessä, responsiivisuutta työstäessä.



Kuvio 24. Breakpointit

Kun muuttujat ovat asetettu, voi siirtyä työstämään varsinaisia tyylejä. Muuttujia on paljon, jonka vuoksi usein jotain unohtuu ja tämän takia muuttujia tulee lisättyä usein pitkin projektia.

3.4 Lohkot

Lohkojen tyylittelyyn siirtyessä on hyvä tehdä tarkka katsaus suunniteltuun layouttiin ja pistää ylös elementtejä, joiden tyylit toistuvat useasti. Koodin toistamisen välttämiseksi, näille toistuville elementeille, on hyvä tehdä mixin. Air-lightissa on valmiina mixineitä yleisesti toistuville elementeille, kuten napeille ja gridille. WorkPowerin kohdalla selvästi toistuvin elementti, jolle on kannattavaa tehdä mixin, on kolmesta kulmasta pyöristetty laatikko. Tämä mixin (ks. liite 2) on tehty mahdollisimman monikäyttöiseksi ja se ottaa huomioon laatikon mahdollisen taustakuvan tehden siitä responsiivisen. Lisäksi kyseinen mahdollistaa kaksi erilaista sumennusefektiä. Muita elementtejä,

joille oli perusteltua tehdä mixin (ks. kuvio 25), olivat erilaiset otsikot ja otsikkoon johdatteleva teksti kuten kuviossa 33.

_prefix.scss

```
@mixin prefix() {
  color: var(--color-main);
  font-family: var(--font-barlow);
  font-size: var(--font-size-15);
  font-weight: var(--font-weight-semibold);
  margin-bottom: 1rem;
  text-transform: uppercase;
}
```

Kuvio 25. WorkPowerin prefix-mixin.

Tarvetta mixinille ei välttämättä huomaa vain silmäilemällä layouttia. Kun huomaa kirjoittaneensa samat tyylit kolmatta kertaa, tajuaa vasta, että tämähän kannattaa tehdä mixinillä.

Lohkot ovat yksinkertaisimmillaan kuvion 19 tapaisia. Lohkot alkavat aina sectionilla, jolle annetaan classiksi ”block” ja sen lisäksi lohkolle yksilöllinen ”block-lohkon nimi tähän”. Sectionille voi tulla useampikin class. Tämmöisiä tilanteita on esimerkiksi, jos on suunniteltu, että koko lohkon taustavärin voi muuttaa tummaksi. Tällöin sectionille tulee lisäksi class ”has-dark-bg”.

_general.scss

```scss
// Default structural element
.block {
  background-color: var(--color-light-blue);
  background-position: center;
  background-repeat: no-repeat;
  background-size: cover;
  margin: 0 auto;
  max-width: $width-max-layout;
  position: relative;
}

// Containers
.container {
  margin: 0 auto;
  margin-left: auto;
  margin-right: auto;
  max-width: $width-grid-base;
  padding-bottom: var(--padding-container-vertical);
  padding-left: var(--padding-container-horizontal);
  padding-right: var(--padding-container-horizontal);
  padding-top: var(--padding-container-vertical);

  &.container-no-padding-y {
    padding-bottom: 0;
    padding-top: 0;
  }
}
```

Kuvio 26. block ja container lähdekoodit.

Lohkoa seuraa aina div, jolla on class ”container” ja mahdollisesti joku sen variantti. Container divin jälkeen HTML-markup vaihtelee vahvasti lohkon sisällön perusteella. Lohkon markupin luonnin jälkeen voi sitä alkaa tyylittelemään. Jokaiselle lohkolle luodaan oma tiedostonsa /sass/modules/ kansion alle, mikä kutsutaan _global.scss-tiedostossa. Tyylit kohdennetaan halutulle lohkolle nestaamalla ne sectionin yksilöllisen classin alle kuten kuviossa 27 on tehty.

_images-two.scss

```scss
.block-two-images {
  .cols {
    @include grid(2, $gutter_x: 13rem);

    @media (max-width: $container-desktop) {
      gap: 6rem;
    }

    @media (max-width: $container-ipad) {
      display: block;
    }
  }

  .image {
    @include rectangle-box($width: 100%, $height: 100%);

    @media (max-width: $container-ipad) {
      @include rectangle-box($width: 500px, $height: 450px);
    }
  }

  .image-small {
    @include rectangle-box($width: 100%, $height: 450px);

    @media (max-width: $container-ipad) {
      @include rectangle-box($width: 500px, $height: 350px);
      margin-bottom: 4rem;
    }
  }
}
```

Kuvio 27. images-two.php (kuvio 19) tyylit

images-two.php on hyvin yksinkertainen lohko, joten tyylit ovat myös yksinkertaisia. Lohko sisältää kaksi eri kokoista kuvaa vierekkäin. Air-light pohjateemaan kuuluva ”grid”-mixin tekee CSS-gridillä sille parametrina annetun määrän palstoja. Kuviin on käytetty aiemmin tehtyä ”rectangle-box”-mixiniä. Hyvän tavan mukaista on myös sijoittaa mixinit ensimmäisinä tyylinä classin alle.

Responsiivisuus on tehty myös välimallin näyttöjä ajatellen käyttämällä _breakpoints.scss-tiedoston (ks. kuvio 24.) muuttujia. Media queryjen sijoittaminen nestattuna suoraan elementin alle tekee koodista helpommin seurattavan verrattuna siihen, jos media queryt sijoittaisi tiedoston loppuun. Henkilökohtaisesti itselle on helpompi tehdä media queryt käyttämällä max-widthiä. On

helpompaa ajatella suuremmasta pienempään, koska näin tulee minimiarvo eli nolla vastaan jossain vaiheessa. Tekemällä pienestä suurempaan maksimiarvoa ei tule koskaan vastaan. Tämä on vain oma preferenssi.

Lohkot ovat eniten aikaa vievä osuus teemakehityksessä. Tässä oli vain yksi esimerkki ja projektikohtaisesti lohkoja voi olla jopa yli 20. Tämän opinnäytetyön liitteistä löytyy monimutkaisemman lohkon template (ks. liite 3) ja tyylit (ks. liite 4). WorkPowerin lohkoissa ripoteltiin pientä liikkuvuutta animaatioiden ja hovereiden muodossa asiakkaan toiveen mukaisesti.

3.5 Header & footer

Air-light- aloitusteeman mukana tulevalla headerilla pääsee pitkälle. Header sisältää helposti muokattavan navigaation. Navigaation tyylit ovat sijoitettu kahteen eri tiedostoon: _nav-desktop.scss ja _nav-mobile.scss. Molempien tiedostojen alussa sijaitsee muuttujia, jotka vaikuttavat navigaation ulkonäköön. Näiden muuttujien ansiosta itse navigaation tyyleihin tarvitsee harvoin koskea, riittää kun muuttaa värit, välistykset ja fontit näihin muuttujiin (ks. kuvio 28).

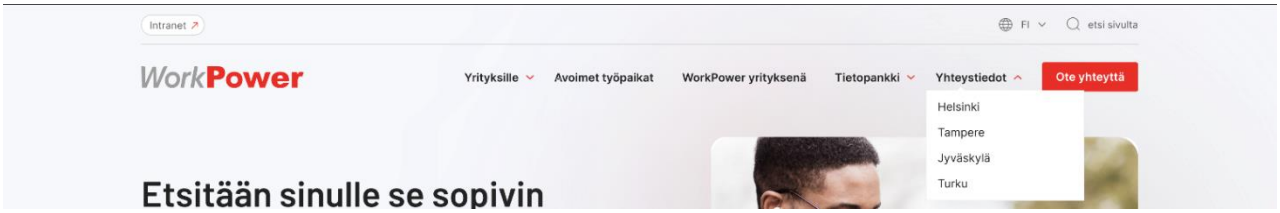
_nav_desktop.scss

```scss
// Site navigation desktop layout
// The main layout for the website navigation, for a desktop devices only.

// Settings
$absolute-navigation: false;
$use-dropdown-bubble: true;
$use-dropdown-toggle-animation: true;

// CSS variables
// These can be adjusted with media queries for in-between breakpoints
:root {
  // Dimensions, gaps and spacings
  --border-radius-sub-menu: 0;
  --box-shadow-navigation-static: 0;
  --dropdown-toggle-size: 12px;
  --gap-dropdown: 1rem;
  --gap-between-dropdown-toggle: 1rem;
  --padding-main-level-vertical: 0;
  --padding-main-level-horizontal: 2rem;
  --padding-sub-menu-vertical: .6rem;
  --padding-sub-menu-horizontal: 1.5rem;
  --width-sub-menu: 22rem;

  // Colors
  --color-background-nav-desktop: transparent;
  --color-background-sub-menu: var(--color-white);
  --color-border-sub-menu: rgba(41, 44, 61, .05);
  --color-dropdown-toggle: var(--color-main);
  --color-sub-menu: var(--color-dark);
  --color-hover-main-level: var(--color-main);
  --color-hover-sub-menu: var(--color-dark);
  --color-main-level: var(--color-dark);
  --color-current: var(--color-main);

  // Font styles
  --font-size-main-level: var(--font-size-16);
  --font-size-sub-menu: var(--font-size-16);
  --font-weight-main-level: var(--font-weight-semibold);
  --font-weight-sub-menu: var(--font-weight-regular);
}
```

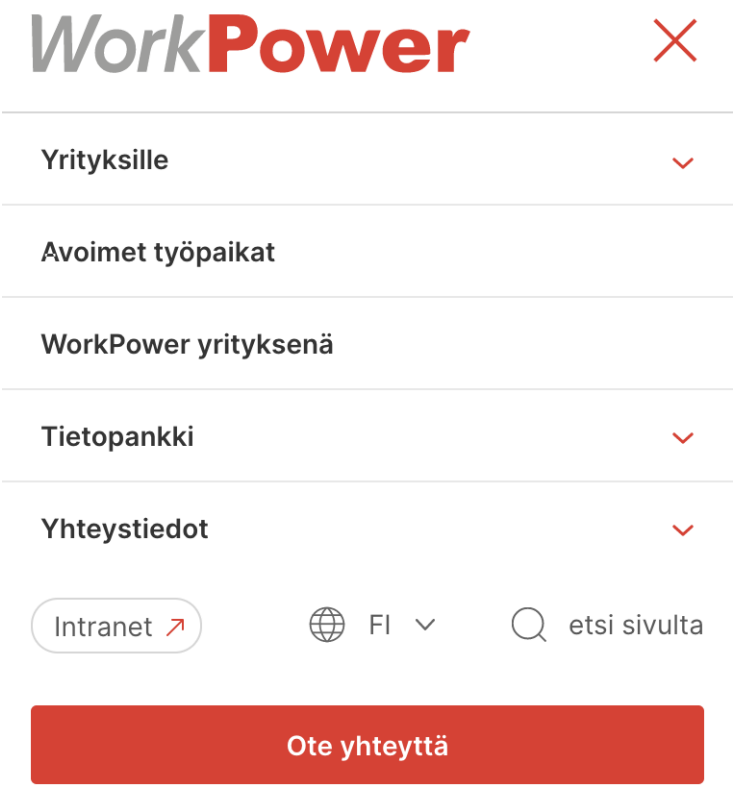
Kuvio 28. Desktop-navigaation muuttujat.

WorkPowerin uusissa sivuissa headeriin tulee myös ylimääräinen top-nav, josta löytyy kielivalinnat, haku ja linkkejä. Top-naville tehdään oma template-part ja se sisällytettiin header.php-templateen sekä navigaation template-partin sisään, jotta top-nav saadaan esille myös mobiilivalikossa.

Top-navin tyylit kuuluvat /layout kansion alle. Header.php templateen sijoitettu top-nav piilotetaan, kun navigaatio muuttuu mobiilissa käytettävään hampurilaisvalikkoon, jotta se ei näkyisi kahteen kertaan. Valmiit navigaatiot kuvioissa 29 ja 30.



Kuvio 29. Valmis desktop-navigaatio



Kuvio 30. Valmis mobiilinavigaatio avattuna

Footerille ei ole Air-lightissa valmista rakennetta sen template-tiedostossa footer.php:ssa tai tyylejä, koska footerit eroavat suuresti projektien välillä ja tämän vuoksi footerin rakentaminen tehdään aina alusta asti. Prosessi muistuttaa pitkälti lohkojen tekoa. Footerin tyylit kirjoitetaan _site-footer.scss-tiedostoon. Footerin alaosa on tässä projektissa erinomainen paikka käyttää flexin gap-ominaisuutta kuten kuviosta 31 näkyy.

Kuvio 31. Flexboxin uusi ”gap” ominaisuus käytössä footerissa

3.6 Näkymät ja artikkelit

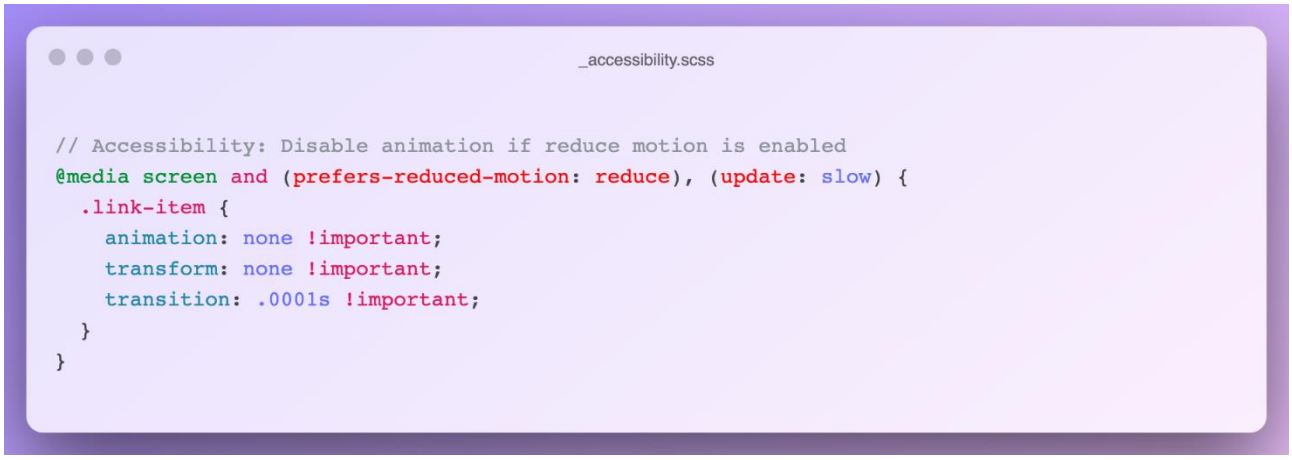
WorkPowerissa tarvittavat näkymät, jotka löytyvät pohjateemasta ovat: 404, haku ja yksittäinen artikkeli. Lisäksi Sivustoon tarvitaan useampi uusi näkymä, joita ovat avoimien työpaikkojen arkistonäkymät, yksittäinen työpaikka, yhteystiedot ja työpaikan hakuprosessi. Näkymien tyylit sijoitetaan /views-kansion alle. Näkymien tyylit menevät myös hyvin samalla kaavalla, kun lohkojen.

Yksittäistä artikkelia muokataan Gutenberg-editorilla. Gutenberg editorin tyylejä muokataan gutenberg-tyylitiedostossa. WorkPowerin artikkelityyleistä muokataan siteerausta, normaalin kuvan leveyttä ja artikkelin tagien ulkonäköä.

3.7 Testaus ja saavutettavuustarkistuslista

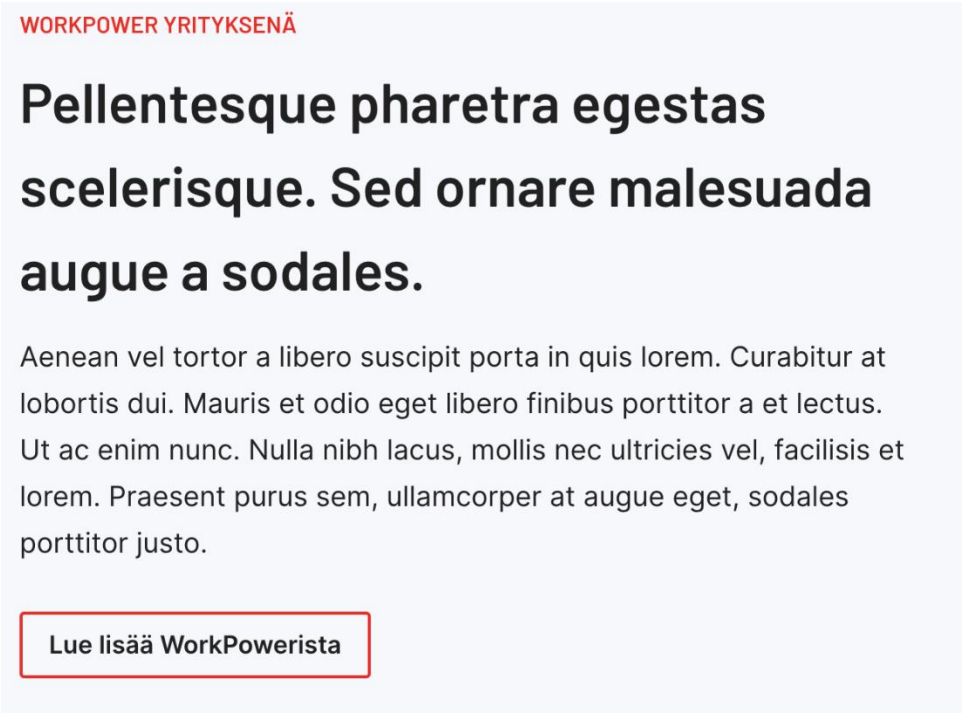
Tyylittelyn jälkeen seuraa laajamittainen testaus. Sivustoa pyöritellään useilla resoluutioilla ja sivuston käytettävyyttä testataan pelkällä näppäimistöllä. Testaamista responsiivisuuden ja saavutettavuuden suhteen tehdään paljon jo kehitysvaiheessa, mutta sivujen laajuuden vuoksi bugeja jää auttamatta kehitysvaiheessa huomaamatta. Tämän takia on hyvä lopuksi käyttää aikaa pelkästään testaamiseen. Testaus tehdään sen jälkeen, kun asiakas on syöttänyt sivustolle oikeat materiaalit.

Dudella käydään läpi aina tyylien jälkeen saavutettavuuden tarkistuslista. Tyylejä kirjoittaessa harvoin tulee saavutettavuuteen liittyviä virheitä linttereiden ansiosta ja Air-lightin valmiuden saavutettavuuden suhteen. Air-lightissa saavutettavuudelle oma tiedostonsa _accessibility.scss. Tiedossa on valmiit tyylit näytönlukijalle, ulkoisille linkeille, päämateriaalin hyppylinkille ja visuaalisesti erotettavat focus-tyylit näppäimistöä käyttäville. Kyseiseen tiedostoon lisätään sivustolla käytetyt animaatiot media queryyn, joka ottaa animaatiot pois käytöstä, mikäli käyttäjä on käyttöjärjestelmässään tai selaimessaan näin määrittänyt (ks. kuvio 32).



Kuvio 32. Animaatioiden poisto

HTML-markup on syytä tarkistaa, koska on inhimillistä unohtaa joitakin aria-atribuutteja. Tarkistuksessa tehdään seuraavat asiat. Ulkopuolisille sivustoille linkitetyille kuville ja logoille asetetaan aria-labelit, jotka ovat muotoa ”Siirry xyz”. Koristeellisille elementeille, koko elementin kattaville globaaleille linkille sekä muille tyhjille elementeille aria-hidden=”true”. Johdatteleville tekstiosuuksille, jotka viittaavat johonkin elementtiin ”aria-describedby”-atribuutit. Tekstin luettavuuden tarkistus, kun suurennus on 200 %. Hyppylinkin asettaminen esimerkiksi sivun h1-pääotsikkoon, joka vie suoraan sivun pääsisältöön. Lopuksi lisätään saavutettavuusseloste. Lopuksi sivusto ajetaan läpi axe-lisäosalla ja korjataan jäljelle jääneet ongelmat.

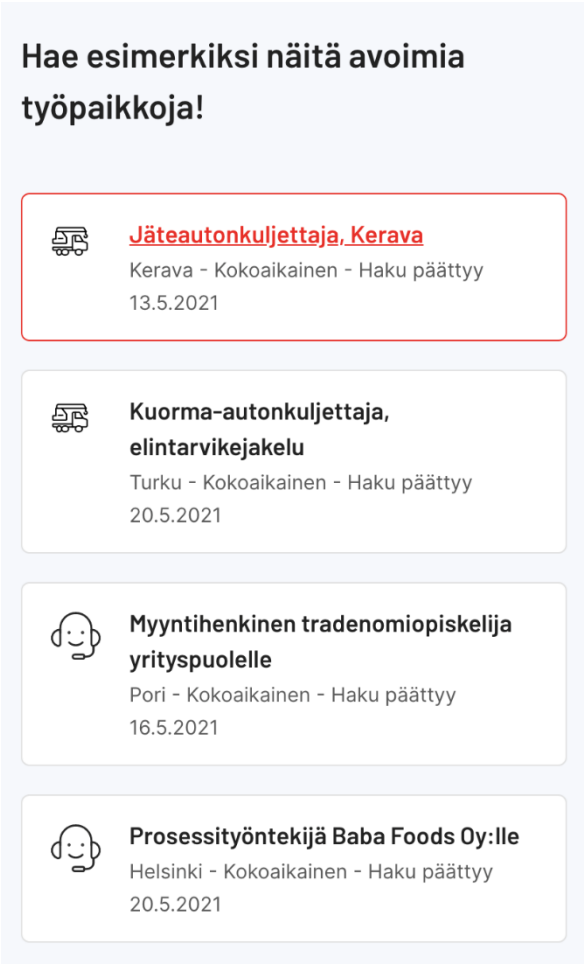


Kuvio 33. Johdatteleva tekstiosuus tässä on ”WorkPower yrityksenä”

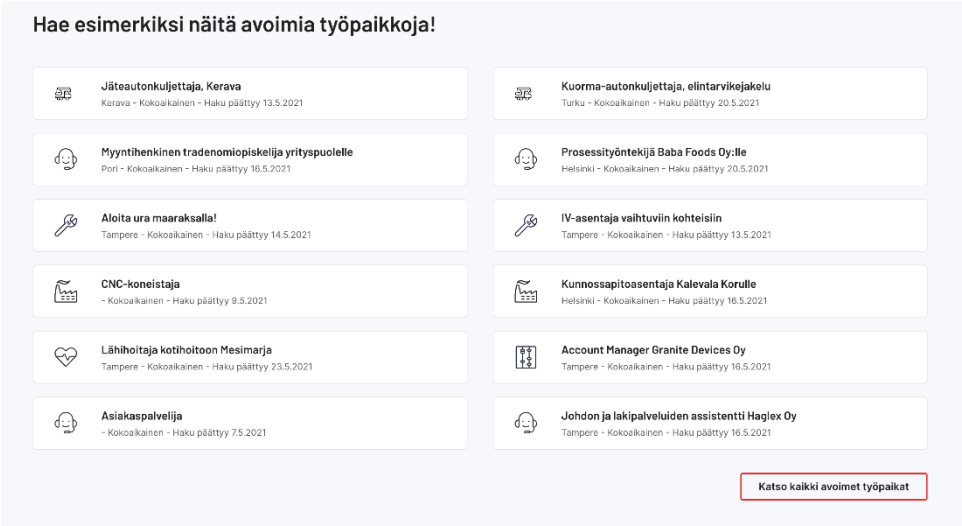
4 Tulokset ja pohdinta

Tehtävänä oli tuottaa WorkPowerille räätälöity WordPress-teema. Teeman valmistamisessa käytettiin tietoperustassa esiteltyjä tekniikoita monipuolisesti. Lopputuloksena on verkkosivu, joka sisältää 16 uniikkia lohkoa (ks. kuviot 34, 35 ja 36) ja useita näkymiä, jotka kertovat projektin laajuudesta. Näiden lisäksi tyylittelyyn kuului sivuston ylä- ja ala-alue, joihin sisältyi navigaatio.

Tavoitteena tällä työllä oli oman osaamisen syventäminen. Suuria harppauksia eteenpäin osaamisessa ei tässä projektissa tapahtunut. Uutena kehittäjänä tieto ja taito kasvaa kuitenkin jokaisessa projektissa, koska uusia asioita tulee jatkuvasti vastaan. Tässä projektissa tuli tutustuttua tarkemmin SVG-kuvien rakenteisiin ja kesken projektin käyttöön otettuun flexboxin ominaisuuteen ”gap”. Monesti aiemmin asetellessa asioita flexboxilla, olin toivonut kyseistä ominaisuutta. Margineilla tai paddingeilla välistäminen oli vaivalloista ja aiheutti ongelmia. Gap-ominaisuus on erinomainen mutta kaipaisin siihen vielä mahdollisuuden määritellä palsta- ja rivivälejä eri kokoisiksi. Myös osaaminen sass-mixinien suhteen kasvoi, etenkin se, milloin mixin on kannattavaa tehdä.



Kuvio 34. Avoimet työpaikat-lohkon mobiili



Kuvio 35. Avoimet työpaikat-lohko

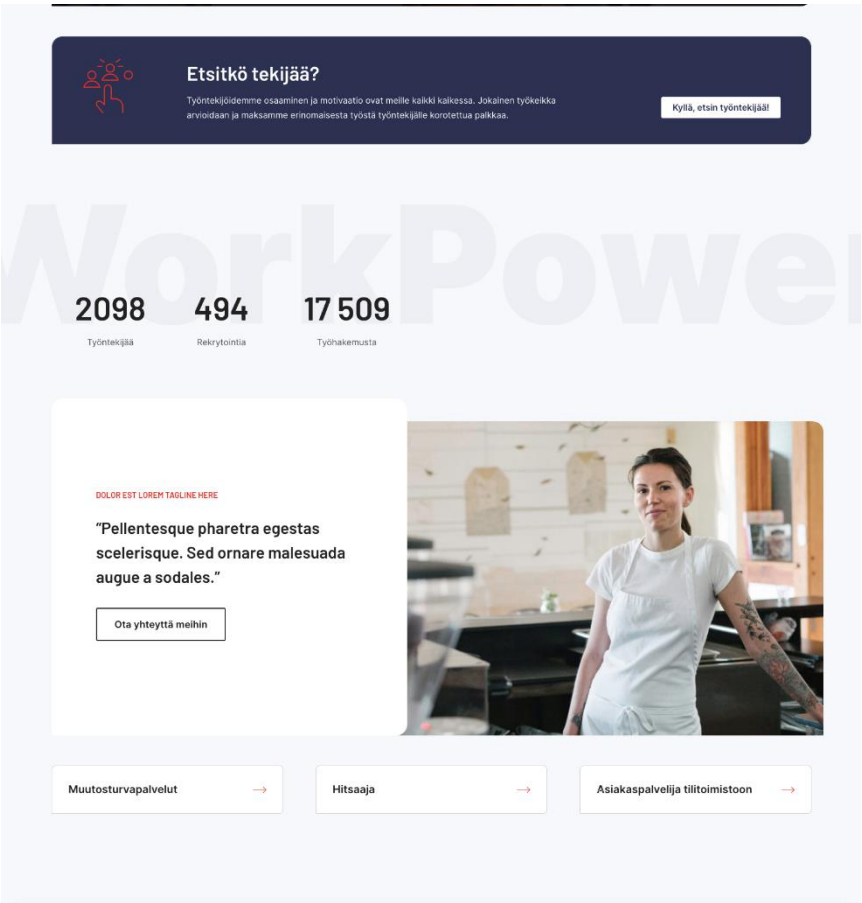
Front-end-kehittäjän näkökulmasta sivujen onnistuneisuutta voi mielestäni arvioida muutamalla eri tavalla, jotka ovat: koodin laatu, käyttäjän kokemus ja ulkoasu.

Koodin laatua määritteleviä osa-alueita ovat jatkokehitettävyys, tehokkuus ja modernien teknologioiden käyttö. Tässä projektissa noudatettiin mielestäni hyvin tietoperustassa käytyjä koodaamiseen liittyviä hyviä käytänteitä kuten modularisoimista ja uusimpia CSS-tekniikoita. Koodin määrä pidettiin minimissään modernien tekniikoiden avulla. Näiden perusteeella koodilla on hyvät valmiudet mahdollisiin jatkokehitysprojekteihin.

Käyttäjän kokemusta on vaikeampi arvioida, koska varsinaisia käyttäjiä sivuilla ei ole vielä ollut ja harvoin varsinaisilta käyttäjiltä, asiakasta lukuun ottamatta, kysytään mielipidettä sivuista. Kehittäjä voi tehdä parhaansa saavutettavuuden, responsiivisuuden ja yleisen käytettävyyden kannalta, mutta lopullinen arvio on aina käyttäjän päässä. Koen kuitenkin itse tehneeni parhaani miellyttävän kokemuksen suhteen noudattamalla tietoperustaani.

Yhtenä henkilökohtaisena onnistumisen kriteerinä pidän sitä, kuinka hyvin pystyn jäljittelemään suunnittelijan suunnittelemaa ulkoasua. WorkPower-projektissa onnistuin tässä lähes pikselin tarkkuudella, osittain kiitos siitä Google Chromen ”Perfect Pixel”-lisäosalle, joka mahdollistaa kuvan lisäämisen malliksi selaimeen.

Kokonaisuudessaan koen henkilökohtaisesti onnistuneeni sivujen toteutuksessa. Mutta koska sivuja ei tehdä itseä varten, vaan asiakkaan, projektin varsinainen onnistuminen mitataan asiakkaan tyytyväisyydessä. Tätä opinnäytetyötä kirjoittaessa projektia ei olla viety vielä julkaisuun asti, mutta toistaiseksi asiakas on pitänyt tuotoksesta.



Kuvio 36. Neljä valmista lohkoa

Pelkästään CSS/Sass-tyylittelyn maailma on laaja ja sen läpikäynti yhdessä opinnäytetyössä on lähes mahdotonta. Koen kuitenkin onnistuneeni kuvaamaan modernin tyylittelyn kokonaiskuvan hyvin. Pieniä asioita jää pakostakin mainitsematta, joten ne jäävät lukijan itse opittavaksi. Mediatekniikan kursseilla käydään läpi pintapuolisesti kaikki web-kehitykseen liittyvä, mikä on harmillista, koska opittavaa on niin paljon. Toisaalta kurssien tarkoitus on antaa katsaus jokaiseen web-kehityksen osa-alueeseen. Täten opiskelija voi päättää mikä osa-alueista on mieluista ja aloittaa itse opiskelemaan aiheesta lisää. Tässä opinnäytetyössä onkin käyty läpi etenkin tyylittelyyn liittyviä asioita, mitä en itse aikoinaan kurssilla oppinut tai joita ei käyty läpi.

WorkPower-projekti oli viimeinen projekti, jossa Dudella lohkot tehtiin tässä opinnäytetyössä kuvatulla tavalla. Lohkot siirrettiin Gutenberg-editorin sisälle seuraavaan projektiin mentäessä. Gutenberg-editorin koettiin viimein olevan riittävän stabiili tämän toteuttamiseen. Jatkossa jokaisessa Duden tekemässä sivustoissa voi kaiken sisällön lisätä Gutenbergin kautta. Useille asiakkaille editori on jo entuudestaan tuttu, tehden sivujen muokkauksesta entistä yhdenmukaisemman.

Tämän projektin aikana tuli myös flexboxiin aiemmin mainittu suuri päivitys ”gap”, joka oikeastaan muutti todella paljon, kuinka CSS:ää kirjoitetaan. Tämä ja aiempi muutos Duden tavassa tehdä lohkoja ovat hyviä esimerkkejä alan jatkuvasta kehityksestä ja siitä, kuinka tärkeää kehittäjänä on pysyä kehityksen aallonharjalla.

Lähteet

A11Y Checklist. N.d. A11Y Project. Viitattu 2.5.2021. https://www.a11yproject.com/checklist/

Baghel, A. S. 2018. Software Design Principles DRY and KISS. Viitattu 28.4.2021. https://dzone.com/articles/software-design-principles-dry-and-kiss

Can I use. 2021. Web technology support table. Viitattu 28.4.2021. https://caniuse.com/css-grid

Can I use. 2021. Web technology support table. Viitattu 28.4.2021. https://caniuse.com/flexbox-gap

Can I use. 2021. Web technology support table. Viitattu 28.4.2021. https://caniuse.com/?search=flex

Can I use. 2021. Web technology support table. Viitattu 28.4.2021. https://caniuse.com/?search=css%20variables

Child themes. N.d. Wordpress for developers. Viitattu 18.4.2021. https://developer.wordpress.org/themes/advanced-topics/child-themes/

Choose a mobile configuration. 2021. Google for developers. Viitattu 3.5.2021. https://developers.google.com/search/mobile-sites/mobile-seo

Clement, J. 2021. Share of global mobile website traffic 2015–2021. Viitattu 18.4.2021. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/

Gash, D. 2017. CSS Variables – No, really! Viitattu 28.4.2021. https://medium.com/dev-channel/css-variables-no-really-76f8c91bd34e

House, C. 2021. A Complete Guide to Grid. Viitattu 28.4.2021. https://css-tricks.com/snippets/css/complete-guide-grid/

Hughes, J. 2018. A Simple Guide to WordPress Starter Themes. Viitattu 25.4.2021. https://torquemag.io/2018/04/wordpress-starter-themes/

Laukkarinen, R. 2021. Native CSS variables vs. SCSS variables. Viitattu 28.4.2021. https://rolle.design/native-css-variables-vs-scss-variables

Lawrence, M. 2019. What is a CSS Framework? Viitattu 28.4.2021. https://medium.com/html-all-the-things/what-is-a-css-framework-f758ef0b1a11

Media queries. 2020. Mozilla web docs. Viitattu 28.4.2021. https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries

Ogidan, B. 2018. History of CSS Grid and CSS Flexbox. Viitattu 28.4.2021. https://medium.com/@BennyOgidan/history-of-css-grid-and-css-flexbox-658ae6cfe6d2

Responsive Web Design. 2021. Google for developers. Viitattu 3.5.2021. https://developers.google.com/search/mobile-sites/mobile-seo/responsive-design

Sass Basics. N.d. Sass documentation. Viitattu 28.4.2021. https://sass-lang.com/guide

Sass Basics. N.d. Sass documentation. Viitattu 9.5.2021. https://sass-lang.com/guide

Template files. N.d. Wordpress for developers. Viitattu 18.4.2021. https://developer.wordpress.org/themes/basics/template-files/

Tutkimus- ja kehittämistoiminta. N.d. Tilastokeskus. Viitattu 11.5.2021. https://www.stat.fi/meta/kas/t_ktoiminta.html

Usage statistics of content management systems. N.d. W3Techs web technology surveys. Viitattu 28.4.2021. https://w3techs.com/technologies/overview/content_management

Using Media Queries for Accessibility. 2021. Mozilla web docs. Viitattu 28.4.2021. https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_Media_Queries_for_Accessibility

What is a Theme? N.d. Wordpress for developers. Viitattu 18.4.2021. https://developer.wordpress.org/themes/getting-started/what-is-a-theme/

Zahra, S. A. 2019. Accessibility Priciples. Viitattu 28.4.2021. https://www.w3.org/WAI/fundamentals/accessibility-principles/

Liitteet

Liite 1. Air-light teemarakenne

Air-light kansiorakenne

```
themes/your-theme-name/             # → Root of your air-light based theme
├── 404.php                         # → Default "not found" page
├── archive.php                     # → Default archive template
├── bin/                            # → Scripts
│   ├── air-move-in.sh              # → A script for moving all dev files back to the theme
│   ├── air-move-out.sh             # → A script for moving all dev files out of the theme for testing with Theme Check plugin
│   ├── air-pack.sh                 # → A script that makes a package for WordPress Theme Directory
│   ├── newtheme.sh                 # → The start script for creating YOUR own theme out of air-light
├── comments.php                    # → Default comments template (can be deleted if not needed)
├── css/                            # → CSS files for production (never edit)
│   ├── dev/                        # → Unminified stylesheet files for debugging (never edit)
│   └── prod/                       # → Minified stylesheet files for production (never edit)
├── fonts/                          # → Your webfont files (woff, woff2, ttf needed)
├── footer.php                      # → Site footer
├── front-page.php                  # → Demo front-page template (not included in wordpress.org version)
├── functions.php                   # → Set up your theme basic settings
├── gulp/                           # → Gulp related settings and tasks
├── gulpfile.js                     # → Core gulpfile for air-light development
├── header.php                      # → Site header
├── images/                         # → Your theme images, for example default featured images and placeholders
├── inc/                            # → Theme core PHP
│   ├── hooks/                      # → Hook functions
│   ├── includes/                   # → Non-template features
│   ├── template-tags/              # → Template functions and helpers
│   ├── post-types/                 # → Custom Post Types
│   ├── taxonomies/                 # → Custom Taxonomies
│   ├── hooks.php                   # → All hooks the theme runs are here
│   ├── includes.php                # → Include non-template features
│   ├── template-tags.php           # → Include template functions and helpers
├── js/                             # → JavaScript files
│   ├── dev/                        # → Unminified script files for debugging (never edit)
│   ├── prod/                       # → Minified script files for production (never edit)
│   └── src/                        # → Script files for development (edit these)
│       ├── navigation.js           # → Accessible multi-level navigation (from 3.4.5)
│       ├── front-end.js            # → Theme core JavaScript file (from 1.0.0, before: scripts.js)
│       └── sticky-nav.js           # → Sticky nav functionality (optional)
├── package.json                    # → Node.js dependencies and scripts
├── page.php                        # → Default page template
├── phpcs.xml                       # → PHPCodeSniffer/WordPress Theme Coding Standards settings
├── sass/                           # → CSS files for development
│   ├── base/                       # → Theme base styles
│   │   ├── _accessibility.scss     # → Accessibility
│   │   └── _normalize.scss         # → Browser reset
│   ├── global.scss                 # → Core CSS file that calls all the modular files
│   ├── gutenberg.scss              # → Core CSS file for Gutenberg editor and blocks
│   ├── components/                 # → Add your style components to this folder
│   ├── features/                   # → Fuctionality styles
│   │   ├── _breadcrumbs.scss       # → Styles for hybrid breadcrumbs
│   │   ├── _gallery.scss           # → Default WordPress gallery feature styles
│   │   ├── _gravity-forms.scss     # → Defaults for Gravity Forms + WCAG 2.0 form fields for Gravity Forms
│   │   ├── _lazyload.scss          # → Styles for air-helper lazyload feature (lazyload.js needed)
│   │   ├── _top.scss               # → Back to top styles
│   │   ├── _pagination.scss        # → Numbered pagination styles
│   │   ├── _sticky-nav.scss        # → Sticky nav styles (not included by default)
│   │   └── _slick.scss             # → Styles for slick-carousel (not included by default)
│   ├── helpers/                    # → Helper mixins and functions
│   │   ├── _animations.scss        # → Animations and effects
│   │   ├── _aspect-ratio.scss      # → A mixin for aspect ratio
│   │   ├── _general.scss           # → Mixins for general use, or helpers of other mixins
│   │   ├── _grid.scss              # → CSS Grid helper mixin
│   │   └── _typography.scss        # → Typography style mixins
│   ├── layout/                     # → Fuctionality styles
│   │   ├── _demo-content.scss      # → Styles for demo, start script will delete this file
│   │   ├── _forms.scss             # → Styles for general forms and Gravity Forms
│   │   ├── _site-footer.scss       # → Footer styles
│   │   ├── _site-header.scss       # → Header styles
│   │   ├── _typography.scss        # → Defaults for typography and fonts
│   │   └── _gutenberg.scss         # → Site-side styles for Gutenberg (pratically for single.php)
│   ├── navigation/                 # → Navigation styles
│   │   ├── _burger.scss            # → Burger styles and animations
│   │   ├── _nav-desktop.scss       # → Desktop navigation styles and dropdowns
│   │   └── _nav-mobile.scss        # → Navigation styles for mobile and touch devices
│   ├── variables/                  # → Configurations
│   │   ├── _breakpoints.scss       # → Widths from mobile to TV screens
│   │   ├── _colors.scss            # → All the colors of the theme
│   │   ├── _fonts.scss             # → Font settings
│   │   └── _spacings.scss          # → Margins and paddings
│   ├── views/                      # → Templates, archives, pages and views go here
│   │   ├── _blog.scss              # → General blog archive and post styles
│   │   ├── _comments.scss          # → Comment styles (optional)
│   │   ├── _front-page.scss        # → Front page styles (demo content, optional)
│   │   └── _page.scss              # → Default single page styles
├── screenshot.png                  # → Theme screenshot for WP admi
├── search.php                      # → Default search view
├── sidebar.php                     # → Default sidebar (optional)
├── single.php                      # → Default single article or CPT view
├── style.css                       # → Theme meta information
├── svg/                            # → Your theme SVG graphics and icons
└── template-parts/                 # → WordPress template parts. Modules go under this folder.
    ├── header/                     # → Header modules
    │   ├── branding.php            # → Site branding
    │   ├── navigation.php          # → Site navigation
    ├── content-modular.php         # → ACF flexible content
    └── hero.php                    # → Default hero
```

Liite 2. _rectangle-box.scss

_rectangle-box.scss

```scss
@mixin rectangle-box($image: true, $aspect-ratio: true, $width: 600px, $height: 587.38px,
$corners: 'hero-style', $blur: true, $background-effect: true, $border: 1px solid var(--color-
border), $padding: 56px 56px, $background-color: var(--color-white)) {
  overflow: visible;

  @if $image == true {
    border-radius: 2rem 2rem 2rem 0;

    > div {
      background-repeat: no-repeat;
      background-size: cover;
      max-height: $height;
      max-width: $width;
      position: relative;
      z-index: 1;

      @if $aspect-ratio == true {
        @include aspect-ratio($width, $height);
      }

      @if $aspect-ratio == false {
        height: 100%;
        max-height: 100%;
        max-width: 100%;
        min-height: $height;
        width: 100%;
      }

      @if $corners == 'hero-style' {
        border-radius: 2rem 2rem 2rem 0;
      }
    }
  }

  @if $image == false {
    background-color: $background-color;
    border: $border;
    padding: $padding;

    @media (max-width: $container-mobile) {
      padding: 3rem;
    }

    @if $corners == 'hero-style' {
      border-radius: 2rem 2rem 2rem 0;
    }
  }

  @if $image == true {
    @if $blur == true {
      &::before {
        background-color: var(--color-blur-background);
        border-radius: 44px;
        content: '';
        display: block;
        filter: blur(100px);
        height: 290px;
        left: -6rem;
        position: absolute;
        top: -6rem;
        width: 247px;
        z-index: 0;
      }
    }

    @if $background-effect == true {
      .blurred-image {
        margin: 17px;
        opacity: .5;
        position: absolute;
        top: 0;
        z-index: 0;
      }

      .blurred-image > div {
        filter: blur(200px);
        height: $height;
        position: absolute;
        width: $width;

        @if $corners == 'hero-style' {
          border-radius: 2rem 2rem 2rem 0;
        }
      }
    }
  }
}
```

Liite 3. upcoming-events.php

upcoming-events.php

```php
<?php

namespace Air_Light;

$title_upper = get_sub_field( 'title_upper' );
$title = get_sub_field( 'title' );
$button = get_sub_field( 'button' );

if ( empty( $title ) ) {
  return;
}

$events = [];
$events_query = new \WP_Query( [
  'post_type'      => 'event',
  'posts_per_page'  => 5,
  'meta_query'      => [
    [
      'key'     => 'start',
      'value'   => wp_date( 'Ymd' ),
      'compare' => '>=',
      'type'    => 'DATE',
    ],
  ]
] );

if ( ! $events_query->have_posts() ) {
  return;
}

while ( $events_query->have_posts() ) {
  $events_query->the_post();

  $event = get_event_details( get_the_id() );
  if ( empty( $event ) ) {
    continue;
  }

  $events[] = $event;
} wp_reset_query();
?>

<section class="block block-upcoming-events">
  <div class="container">
    <div class="cols">

      <div class="col col-title">
        <?php if ( ! empty( $title_upper ) ) : ?>
          <p class="prefix"><?php echo esc_html( $title_upper ); ?></p>
        <?php endif; ?>

        <h2><?php echo esc_html( $title ); ?></h2>

        <?php if ( ! empty( $button ) ) : ?>
          <p class="button-wrapper">
            <a href="<?php echo esc_url( $button['url'] ) ?>">
              <?php echo esc_html( $button['title'] ); ?>
            </a>
          </p>
        <?php endif; ?>
      </div>

      <div class="col col-events">
        <?php foreach ( $events as $event ) : ?>

          <div class="event">
            <a href="<?php echo esc_url( $event['permalink'] ); ?>" class="global-link" aria-
hidden="true"></a>
            <div class="cols">

              <div class="col col-date">
                <div class="wrapper">
                  <p class="date">
                    <?php echo esc_html( wp_date( 'j', strtotime( $event['dates']['start'] ) )
); ?>
                  </p>
                  <p class="month">
                    <?php echo esc_html( wp_date( 'F', strtotime( $event['dates']['start'] ) )
); ?>
                  </p>
                </div>
              </div>

              <div class="col col-info">
                <h3 class="description">
                  <?php echo esc_html( $event['title'] ); ?>
                </h3>

                <?php if ( ! empty( $event['location'] ) ) : ?>
                  <p class="location">
                    <?php echo esc_html( $event['location'] ); ?>
                  </p>
                <?php endif; ?>
              </div>

            </div>
          </div>

        <?php endforeach; ?>
      </div>

    </div>
  </div>
</section>
```

Liite 4. _upcoming-events.scss

_upcoming-events.scss

```scss
.block-upcoming-events {
  .container {
    @media (min-width: $container-mobile) {
      padding-bottom: calc(var(--padding-container-vertical) * 2);
      padding-top: calc(var(--padding-container-vertical) * 2);
    }
  }

  .container > .cols {
    @media (min-width: $container-ipad) {
      @include grid(2, $gutter_y: calc(174px - 6rem), $gutter_x: calc(174px - 6rem));
    }

    @media (max-width: 1200px) {
      grid-gap: 6rem;
    }
  }

  .col-title {
    @media (max-width: $container-ipad) {
      margin-bottom: 3rem;
    }

    h2 {
      margin-top: 1rem;
    }
  }

  .col-events {
    @include grid(1, $gutter_y: 1.5rem);
  }

  .col-date {
    align-items: center;
    background-color: var(--color-white);
    border-bottom: 1px solid var(--color-white);
    border-left: 1px solid var(--color-light-blue);
    border-radius: 6px 0 0 6px;
    border-right: 1px solid var(--color-light-blue);
    border-top: 1px solid var(--color-white);
    display: flex;
    padding: 2rem;

    p {
      margin: 0;
    }
  }

  .button-wrapper a {
    @include button();
    @include button-style-ghost();
  }

  .col-info {
    background-color: var(--color-white);
    border-bottom: 1px solid var(--color-white);
    border-left: 1px solid var(--color-light-blue);
    border-radius: 0 6px 6px 0;
    border-right: 1px solid var(--color-white);
    border-top: 1px solid var(--color-white);
    padding: 2rem 4rem;
  }

  .date {
    font-size: 36px;
    font-weight: var(--font-weight-semibold);
    letter-spacing: -2px;
    line-height: 36px;
    text-align: center;
  }

  .month {
    font-size: var(--font-size-14);
  }

  .description {
    font-family: var(--font-barlow);
    font-size: var(--font-size-20);
    font-weight: var(--font-weight-semibold);
    margin-bottom: .5rem;
  }

  .location {
    font-size: var(--font-size-14);
    margin-top: .5rem;
  }

  .event {
    position: relative;

    .cols {
      @include grid(2, $gutter_x: 0);
      grid-template-columns: 110px 1fr;
    }

    .col-date {
      align-items: center;
```