jamk.fi

TailwindCSS:n integrointi Gutenberg-kehitykseen

Jere Pyhäjärvi

Opinnäytetyö
Marraskuu 2019
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Pyhäjärvi, Jere	Opinnäytetyö, AMK	Marraskuu 2019
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
TailwindCSS:n integrointi Gutenberg kehitykseen

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Kari Niemi

Toimeksiantaja(t)
Arto Saksola

Tiivistelmä

Työn tavoitteena oli kartoittaa uuden Gutenberg-editorin toimintoja, sen räätälöinnin vaativuutta ja sitä kautta kannattavuutta. Työn lopullinen tarkoitus oli kehittää usein käytetyistä lohkoista prototyyppejä ja tutkia, onko lohkojen kehitys React.js:n keinoin kannattavaa myös tulevaisuudessa.

Työ oli kehittämistutkimusta, jota tehtiin 2 – 4 viikon sykleissä. Syklien välillä tarkasteltiin tuloksia ja sen perusteella päätettiin tulevan syklin tehtävät ja kehityskohteet sekä mihin suuntaan kehitystä jatketaan. Syklien aikana sekä kehitettiin lohkoja että tutkittiin editorin ja itse lohkojen toimintaa. Tutkittiin, miten valmiita toimintoja voitaisiin käyttää hyväksi mahdollisimman paljon, jotta kuluja kehityksestä saataisiin minimoitua.

Kehitystyön tulokset osoittivat, ettei TailwindCSS-kehystä kannata sulauttaa lohkoihin liian syvälle, vaan lohkoihin kannattaa ottaa käyttöön TailwindCSS-kehyksen käyttämä komponenttimalli. Kyseisen mallin lisäksi lohkojen asetuksiin luotiin valintoja, joihin haettiin tiedot TailwindCSS-kehyksen asetustiedostosta. Näin lohkot ovat tulevaisuudessa helppo muokata tukemaan toista kehystä, jos se on tarpeen. Lohkot olivat myös tarpeeksi muokattavia, että keskikokoisissa projekteissa voidaan säästää arviolta 10 – 20 % työajasta. Lohkoilla on mahdollista säästää työajassa myös ylläpidossa, koska ne tarjoavat muuttumattomia rakenteita eri projektien välillä, jolloin muutosten tekeminen nopeutuu. Kehityksen aikana saatiin myös selville, että omien lohkojen kehittäminen joissain tapauksissa on välttämätöntä, mutta uusia lohkoja tulvaisuudessa kehitettäisiin vasta kun niitä tarvittaisiin.

Avainsanat (asiasanat)
Gutenberg, TailwindCSS, integrointi,

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Pyhäjärvi, Jere	Bachelor's thesis	November 2019
		Language of publication: Finnish
	Number of pages: 45	Permission for web publication: x

Title of publication

Integrating TailwindCSS to Guteberg development

Degree programme

Information and Communication Technologies

Supervisor(s)

Niemi, Kari

Assigned by

Saksola, Arto

Abstract

The objective of the development was to gather information about the new Gutenberg editor and its capabilities and possible ways of customizing it for the company's own purposes. The end goal was to develop a minimum of five blocks to be used frequently in most of the WordPress projects and see if developing custom blocks with React.js is at all worthwhile in the future.

The work consisted of theory and development based on the theory. The development was split into two- to four-week cycles, after which the results of cycle were reviewed and the new direction for next development cycle was settled. The cycles itself consisted of gathering a base of theory and then applying that theory into practice in the development of the blocks. Readymade functionalities of Gutenberg were also under the scope during the cycles to save up some development time.

The results of the development showed that integrating TailwindCSS framework too deeply into the blocks was not a good idea for the future. Instead, the direction should lean heavily towards using the component-based principles of TailwindCSS framework. The settings were made into the blocks' own control panels that pulled their data straight from the config-file of the framework. This was performed so it is easy to remove any associations between the block and the framework if the developers end up not using it in the future. The blocks were also customizable enough to save approximately 10 – 20 % of working time on medium-sized projects. The blocks can save time in maintenance as well as they provide constant structures between projects, which in turn speeds up the work. Another note during the development was not to build custom blocks from ground up unless necessary as it in some cases is.

Keywords/tags

Gutenberg, TailwindCSS, integration

Miscellaneous (Confidential information)

Sisältö

Kuviot

Sanasto

Blob – Binary Large Object on tallennusmuoto, jota käytetään tietokannoissa binaari datan tallennuksessa.

CSS – Cascading Style Sheets on kieli, jolla kuvataan HTML tiedoston tyylejä.

Core – WordPress-alustan ja Gutenberg-editorin mukana tulevia ydin ominaisuuksia.

HTML – Hypertext Markup Language on kieli, jolla rakennetaan verkkosivuja.

JavaScript – Ohjelmointi kieli, jota käytetään usein verkkosivujen ja palveluiden kehityksessä

JSON – JavaScript Object Notation on tapa esittää dataa tekstipohjaisena.

JSX – ExtendScript JavaScriptin laajennus, jota käytetään etenkin React-kehityksessä.

PHP – Palvelinohjelmointikieli, jolla on mahdollista tulostaa HTML-dokumentteja

React.js – JavaScript kirjasto erityisesti käyttöliittymien rakentamiseen.

Renderöinti – Lopullisen tiedostomuodon prosessointi

Skeema – Tietokantamalli, joka kuvastaa esimerkiksi tietokantataulukon rakennetta.

Webpack – Moduulien hallinta ja rakennus työkalu.

WordPress – Verkkosivujen sisällönhallintajärjestelmä

1 Työn lähtökohdat

1.1 Taustaa ja toimeksiantaja

Toimeksiantaja on web-palveluita kehittävä yritys Trimedia Oy, joka toimii Jyväskylässä ja Helsingissä. Trimedian asiakkaina on erikokoisia yrityksiä, yhdistyksiä ja säätiöitä. Trimedia keskittyy suurimmaksi osaksi kehittämään palveluita WordPress-julkaisujärjestelmää hyödyntäen, mutta tekee myös täysin räätälöityjä palveluita.

Suuri osa web-kehityksestä Trimedialla tapahtuu WordPress-alustalla, joten alustan uusi ominaisuus Gutenberg-editori aiheutti hämmennystä. Oli epäselvää, miten kehitys uudella editorilla toimisi, sillä editorin sisältämät lohkot olivat hyvin vieras käsite monille. Gutenberg tuo myös haasteita koskien vanhojen sivustojen ja palveluiden uudistamista tukemaan sitä.

1.2 Tehtävät ja tavoitteet

Päätavoitteena oli yhdistää yrityksellä käytössä olevat jo aiemmin hyväksi todetut prosessit uuteen Gutenberg-editorin kanssa työskentelyyn ja sitä kautta kartoittaa, mikä kaikki sen kanssa on mahdollista. Kun saadaan kerättyä tarpeeksi tietoa itse Gutenberg-editorista ja sen räätälöinnistä eri menetelmillä, voidaan päättää, mihin keskitytään tulevaisuudessa vai keskeytetäänkö sisäinen kehitys kokonaan.

Tarkoitus oli räätälöidä editoria ja lohkoja yrityksen omiin työtapoihin sopivaksi. Kehityksen aikana pysyttiin kuitenkin kriittisinä räätälöinnin suunnan kanssa, sillä varsinaista suuntaa ei vielä ollut pystytty lyömään lukkoon.

Kehityksen aikana oli tarkoitus rakentaa prototyyppi, jonka avulla tutkitaan räätälöityjen lohkojen käyttöä oikeissa projekteissa peruseditorin ja lohkojen kera. Näin selvitettiin, kannattaako editorin räätälöintiä jatkaa. Minimi vaatimukset prototyypille oli viisi lohkoa, joita voitaisiin hyödyntää lähes jokaisessa projektissa.

Tutkimuksen piiristä on rajattu pois WordPressin ja React.js:n tarkempi käsittely, jotta tutkimus ei paisuisi liian suureksi. WordPressin toimintoja käsitellään Gutenbergin käsittelyn yhteydessä tarpeen mukaan. React.js:n käsittelyssä tarkastellaan ainoastaan erot itsenäisen React.js-applikaation ja Gutenberg-applikaation välillä. Myös perinteiset HTML- ja CSS-teknologiat jätetään erikseen käsittelemättä.

1.3 Tutkimusmenetelmästä

Tutkimusmenetelmänä toimi kehittämistutkimus, ja käytännössä tämä tarkoitti sitä, että yrityksellä oli tarve muokata tuotetta ja tutkia sen ominaisuuksia kehityksen kannalta. Tuotetta lähtökohtaisesti pyritään aina muokkaamaan paremmaksi, joko itselle tai asiakkaille. Tässä tapauksessa tarkoitus oli parannella WordPress-kehittäjien tekemää avoimen lähdekoodin tuotetta omaan käyttöön sopivaksi. Kehittämistutkimus tässä tapauksessa tapahtui sykleittäin. Syklit eivät olleet ennalta määrätyn mittaisia, mutta niiden lopussa tutkittiin tuloksia ja päätettiin seuraavan syklin tavoitteet.

Tutkimusongelma oli kiteytettynä uuden editorin käyttöönotto ja räätälöinti. Tutkimusongelmasta voidaan johtaa tutkimuskysymykset, joiksi valikoituivat seuraavat kysymykset:

- Mikä Gutenberg editori on?
- Voiko Gutenbergiin sulauttaa TailwindCSS:n?
- Onko lohkojen räätälöinti kannattavaa?

2 Gutenberg

2.1 WordPressin uusi suunta

WordPress on avoimen lähdekoodin julkaisujärjestelmä, joka on kehitetty jokaisen tarpeisiin sopivaksi. Kuka tahansa voi kehittää järjestelmää eteenpäin tai ottaa järjestelmän pohjaksi omaan projektiinsa ja muokata siitä tarpeisiinsa sopivan. (Democratize Publishing n.d.)

WordPress on rakennettu alun perin PHP:n päälle, mutta tulevaisuudessa sen koodipohja hallinnan puolella muuttuu enemmän React.js painotteiseksi. Hyvä esimerkki tästä on nimenomaan Gutenberg-editori, joka korvaa aikaisemman PHP-pohjaisen editorinäkymän Reactia hyödyntävällä editorilla. WordPress oli jäämässä alakynteen staattisten sivugeneraattorien (static site generator SSG) takia, koska staattiset sivut tarjoavat nopeammat latausajat ja paremman tietoturvan kuin PHP-pohjaiset ratkaisut. (Gowans 2016.)

2.2 Toiminta

Koko Gutenbergin ydin perustuu lohkoihin. Lohkoista muodostuu kokoelma elementtejä, joiden ennalta määrityt rakenteet korjaavat ongelmia perinteisen sisällöntuotannon monitulkintaisuuden kanssa. Lohkot yleisesti sisältävät ominaisuuksia, joiden avulla on helpompaa ja järjestelmällisempää manipuloida lopullista HMTL-dokumenttia. (Key Concepts n.d.)

Gutenberg korvaa vapaamuotoisen tekstin lohkoilla, joita käyttämällä voidaan luoda monipuolisia ja mukautuvia sivustoja. Lohkot käsittelevät otsikot, paragrafit ja muun

median komponentteina, jotka voivat sisältää kuvia ja upotuksia. (Block Editor Handbook n.d.)

Lohkot ovat yksikköjä, joiden kanssa voidaan järjestellä ja luoda sisältöä sivustolle. Lohkot seuraavat hierarkiaa, eli lohko voi olla lapsi tai vanhempi. Yksi lohko voi sisältää useita lapsilohkoja, esimerkiksi palstat-lohko voi sisältää 2-8 palstaa. Lohkot voivat olla joko staattisia tai dynaamisia. Staattiset lohkot sisältävät renderöityä sisältöä ja Attribuutti-objektin, jonka muutosten perusteella lohkoa uudelleen renderöidään. Lohkoista on mahdollista tehdä myös uudelleen käytettäviä, siten että muutokset näkyvät kaikilla sivuilla, joissa lohko on käytössä. (Key Concepts n.d.)

Lohkot ovat työkalu monipuolisen sisällön muokkaamiseen, mutta ne itsessään eivät vaikuta kuin editointiprosessiin. Lohkot ja niiden tuottama sisältö ovat aina samanlaiset, mutta eri lohkorakenteilla voidaan tuottaa samanlainen lopputulos. Tämä tarkoittaa siis sitä, että lopullisesta prosessoidusta HTML-dokumentista ei voida suoraan johtaa sen luonutta lohkoa, vaikkakin sen rakenne on jokseenkin näkyvissä. Lohko voi olla huomattavasti monimutkaisempi rakenteeltaan, kuin sen tuottama HTML, joten lohkoja voidaan ajatella myös HTML-laajennuksena. (Key Concepts n.d.)

Ajoaikana lohkoja pidetään muistissa, joten niin sanottu “Gutenberg post” eli Gutenberg lohkojen kokoelma, säilytetään HTML:n sijaan objektina, joka sisältää lopulliseen prosessointiin tarvittavan datan. Näin saadaan pidettyä editori ja lopullinen HTML-näkymä erillään toisistaan. Objekti, johon lohkot tallennetaan, on HTML:n tavoin solmuja sisältävä puu. (Key Concepts n.d.)

2.3 Lohkojen rakenne

Lohkojen tallentuessa tietokantaan voidaan ajatella, että niistä tallennetaan “vedos”, joka sisältää lohkolle asetetut ominaisuudet, varsinaisen sisällön ja lohkon nimen.

Tätä vedosta voidaan käyttää, joko editorissa asettamaan lohkot oikeaan paikkaan sen ominaisuuksilla ja sisällöillä tai itse sivuston kasaan parsimisen aikana oikean laisen HTML-rakenteen tekemiseen. Lohkon “vedos” tietokannassa koostuu HTML-kommenteista ja elementeistä. (ks. kuvio 1.) HTML-kommenttien sisältä löytyy itse attribuutit, joita käytetään editorin puolella luomaan lohkon rakenne tallennetun kaltaiseksi. (Key Concepts n.d.)

```
<!-- wp:cgb/singlecol -->
<div class="wp-block-cgb-singlecol alignfull undefined">

    <!-- wp:heading {"align":"center","level":1,"className":"is-style-default"} -->
    <h1 class="has-text-align-center is-style-default">Otsikko - lapsi lohkolle singlecol</h1>
    <!-- /wp:heading -->

    <!-- wp:paragraph {"className":"is-style-default"} -->
    <p class="is-style-default">Paragrafi, lapsi lohkolle singlecol, sisar lohkolle heading.</p>
    <!-- /wp:paragraph --></div>

<!-- /wp:cgb/singlecol -->
```

Kuvio 1. Sisäkkäiset lohkot artikkeleissa tallennettuna tietokantaan

Lohkot tallennetaan tällä tavalla, jotta voidaan luottaa, että tallennettu tieto on todenmukaista ja ajantasaista, sekä koko WordPress-alustan luettavissa missä tahansa näkymässä, jossa lohkoja tulkitaan. Jos rakenne olisi tallennettuna erilliseen tiedostoon, voisi syntyä synkronointi ongelmia itse artikkelin sisällön ja rakennetta ohjaavan tiedoston välillä. (Key Concepts n.d.)

Koska lohkojen tiedot tallennetaan suoraan artikkelin sisällöksi, niiden tuottamaa sisältöä voidaan käyttää myös WordPress-ympäristöissä, jotka eivät tue Gutenbergiä. Gutenbergin tuen puute kuitenkin aiheuttaa sen, ettei kaikkea pysty näyttämään tarkoitetulla tavalla. Esimerkiksi dynaamiset lohkot, eivät toimi ilman Gutenbergin tukea, mutta perus HTML-elementeiksi prosessoitavat lohkot pystytään silti tulkitsemaan oikein. (Key Concepts n.d.)

2.4 Lohkojen ominaisuuksien parsiminen

Lohkojen ominaisuudet kuljetetaan HTML-kommenttien sisällä, koska kommenteilla on selkeä rakenne. Kommentti alkaa aina “<!--”-merkinnällä ja loppuu ensimmäiseen sitä seuraavaan “-->”-merkintään. Kommenttien selkeän rakenteen lisäksi, voidaan kommentin sisälle kirjoittaa lähestulkoon mitä tahansa ilman vaikutusta varsinaiseen HTML:ään, jota selain tulkitsee. Näin on mahdollista kuljettaa JSON-objekteja ja muuta lohkoon liittyvää dataa kommenteissa, joista ne parsitaan editorin käyttöön, ilman vaikutuksia selaimen tulkintaan. HTML-elementtien radikaali ero HTML-kommentteihin mahdollistaa pelkkien lohkojen ja datan keräämisen, jolloin ei tarvitse huolehtia itse HTML-elementtien pätevyydestä. (Key Concepts n.d.)

Selkeät rajat parsiessa auttavat myös eristämään virheitä parsimisen aikana, ettei virheet yhden lohkon parsinnassa vaikuta muihin lohkoihin. Näin estetään, ettei yksi virhe aiheuta koko dokumentin rikkoutumista. (Key Concepts n.d.)

2.5 Gutenberg ja teemojen kehitys

Editorin tyylit ajetaan Gutenbergissä parsijan läpi, joka automaattisesti muuttaa tehdyn CSS-tyylitiedoston lohkoeditorille sopivaksi. Esimerkiksi CSS-valitsin “body” muuttuu parsimisen tuloksena “.editor-styles-wrapper” valitsimeksi. Koska parsin kohdentaa automaattisesti perinteiset valitsimet lohkoeditorin vastaaviin valitsimiin, alkuperäiseen CSS-tiedostoon ei ole suotavaa asettaa valitsimia, joita editorissa esiintyy. (Theme Support n.d.)

Gutenberg “core”-lohkoja voidaan käyttää suoraan minkä tahansa teeman kanssa, kunhan itse Gutenbergiä ei ole otettu pois käytöstä. Teemojen kehittämisessä tulee ottaa huomioon teeman tuet, joita voidaan lisätä tarpeen vaatiessa. Gutenberg “core"-lohkoille on annettu perus-CSS-asettelut, joita voidaan yliajaa, jos halutaan

tehdä teemasta räätälöidympi. Teemalle voidaan lisätä ominaisuus tukea laajaa valikoimaa erilaisia edistyneempiä toimintoja, väripaletista lohkon asetteluun tai vaikka tekstikokojen valintaan. Esimerkiksi rekisteröimällä oman väripaletin teemalle, kehittäjän on mahdollista rajoittaa editorissa käyttäjälle näkyviä värejä. (ks. kuvio 2.) (Theme Support n.d.)

```
add_theme_support( 'editor-color-palette', array(
  array(
      'name'  => __( 'Blue', 'froggy' ),
      'slug'  => 'blue',
      'color' => '#59BACC',
  ),
  array(
      'name'  => __( 'Green', 'froggy' ),
      'slug'  => 'green',
      'color' => '#58AD69',
  ),

) );
```

Kuvio 2. Oman väripaletin lisääminen teemaan

Värillä on aina nimi (name), slugi (slug) ja värikoodi (color). (ks. kuvio 2.) Nimi on periaatteessa etiketti värille, joka on pääasiassa tarkoitettu ihmisen luettavaksi. Slugin avulla luodaan luokat CSS-määrittelyitä varten. Värikoodi on värin heksadesimaali. Funktiolla “add_theme_support” voidaan lisätä myös muut lisäominaisuudet. (Theme Support n.d.)

Lohkoille voidaan antaa myös sisäänrakennettu leveyden valitsin. (ks. kuvio 3.) Tämänkin ominaisuuden lisääminen tapahtuu käyttämällä funktiota “add_theme_support”. Ensimmäinen parametri määrittää kyseisessä funktiossa sen mitä tukea ollaan lisäämässä teemaan. Mahdollinen toinen parametri puolestaan ottaa vastaan datan, jota kyseisen ominaisuuden rakentamiseen käytetään. (Theme Support n.d.)

```
// Add support for full and wide align blocks.
add_theme_support( 'align-wide' );
```

Kuvio 3. Lohkojen leveysasetusten käyttöönotto

Molemmissa tapauksissa editorille annetaan mahdollisuus lisätä lohkoille luokkia, värien tapauksessa esimerkkiluokka olisi “has-blue-background-color”. Lohkolle annettava väriluokka alkaa aina “has-”, jota seuraa värin slugi “blue-” ja viimeiseksi määritetään, mille väri annetaan “background-color”. Lohko, jonka teksti on sininen, sisältää luokan “has-blue-color” ja niin edelleen. (Theme Support n.d.)

2.6 Lohkon rakenne

2.6.1 Rekisteröintifunktio

Lohkojen kehitys alkaa aina itse lohkon rekisteröinnistä, joka tapahtuu funktiolla “registerBlockType”. Funktiolle annetaan kaksi parametriä, joista ensimmäinen on lohkon nimi ja toinen on asetusobjekti. Lohkon nimen tulee olla yksilöllinen ja tekstimuodossa pienillä kirjaimilla kirjoitettu. Lohkon asetukset ovat laajempi kokonaisuus, jossa on pakollisia ja valinnaisia ominaisuus ryhmiä. (Block registration n.d.)

2.6.2 Pakolliset asetukset

Lohkoasetusten välttämättömät ominaisuudet rekisteröinnin kannalta ovat otsikko (title) ja kategoria (category). Otsikko on ihmisen luettavaksi tarkoitettu lohkon nimi. Kategorialla asetetaan lohko valikossa oikeaan paikkaan löytämisen helpottamiseksi. Valmiita kategorioita ovat yleiset (common), formatointi (formatting), asettelu (layout), vimpaimet (widgets) ja upotukset (embed). (Block registration n.d.)

2.6.3 Valinnaiset asetukset

Lohkolla on useampi valinnainen asetus, osa hyvin yksiselitteisiä, kuten kuvaus (description), avainsanat (keywords) ja ikoni (icon), osa ominaisuuksista on laajempia. Tyylit (styles) on asetus, jolla voidaan rekisteröidä lohkolle eri tyylejä. Lohkotyyleihin perehdytään tarkemmin luvussa 2.10. Ominaisuudet (attributes) on lohkon tallennuspaikka sen datalle, ominaisuuksien toimintaa tarkastellaan 2.7. Esimerkki (example) -asetuksen avulla voidaan rakentaa lohkolle lohkovalikossa näytettävä malli lohkosta. Muunnokset (transforms) -asetuksella voidaan määrittää lohkot, joiksi rekisteröitävä lohko voidaan editorissa muuttaa. Vanhempi (parent) -asetuksella voidaan estää lohkon lisääminen muualle kuin nimetyn lohkon sisälle. Lapsi (child) -asetuksella määritetään, mitä lohkoja rekisteröitävän lohkon sisään voi asettaa. Tuet (supports) -asetuksella saadaan lisättyä lohkolle tuki Gutenbergin sisäänrakennettuihin ominaisuuksiin. Sisäänrakennettuja ominaisuuksia käsitellään luvussa 2.8. (Block registration n.d.)

2.6.4 Muokkausfunktio

Muokkausfunktio (edit) kuvastaa lohkoa hallintapaneelin puolella esimerkiksi artikkelin muokkausnäkymässä. Funktion tuottama ulkoasu voi poiketa lohkon lopullisesta ulkoasusta paljonkin. Funktio ottaa parametrinä lohkon ominaisuudet sekä luokat ja mahdollisesti myös muita parametrejä ja piirtää niiden perusteella lohkon editoriin. (Edit and save n.d.)

2.6.5 Tallennusfunktio

Lohkon tallennusfunktio (save) käyttää samoja parametrejä kuin muokkausfunktio, joiden perusteella se palauttaa lohkon lopullisen rakenteen, joka tallentuu artikkelisisältöön. Tallennusfunktio ei voi hakea tietoa ulkopuolelta, vaan sen tulee käyttää ai-

noastaan muokkausfunktiolta saatua dataa validoinnin vuoksi. Rajoitetta voidaan kuitenkin kiertää dynaamisten lohkojen avulla, joista lisää luvussa 2.12. (Edit and save n.d.)

2.7 Lohkon ominaisuudet

2.7.1 Ominaisuuksien lähteet (source)

Jos ominaisuuksille ei erikseen määritetä lähdettä, ne tallennetaan lohkon kommenttierottimeen HTML-esitysmuotoon. Ominaisuudet luetaan kyseisestä erottimesta oletuksena tulkintavaiheessa. Ominaisuuksien määritykset tallennetaan avainkohtaisesti, ominaisuuksien avaimet voi nimetä haluamikseen. (Attributes n.d.)

2.7.2 Ominaisuuksien valitsimet (selector)

Ominaisuuksille voidaan antaa valitsin, jolla kerrotaan, mistä ominaisuuden data haetaan. Valitsimen ollessa tyhjä data haetaan lohkon juurisolmusta, muussa tapauksessa se haetaan valitsinta vastaavasta elementistä lohkon sisältä. Valitsin voi olla HTML-tagi, -id tai -luokka, mikä tahansa millä voidaan tehdä kysely. (Attributes n.d.)

2.7.3 Ominaisuuksien tyypit ja tallennus

Wordpress.org sivuston kehittäjäkäsikirjan mukaan ominaisuuden tulee olla jokin seuraavista tietotyypeistä:

- Null
- Boolean
- Object
- Array
- String
- Number
- Integer

(Attributes n.d.)

Ominaisuudet tallentuvat JSON-muotoon niin sanottuihin blobeihin, jotka käyttävät skeemoja (schema) validoidakseen datan oikean tietotyypin. Skeemat ovat tärkeitä datan validoinnissa, sillä datan vastaan ottava sovellus ei ole tietoinen datan tyypistä. (REST API Handbook n.d. Schema)

2.7.4 Esimerkki toiminnasta

Ominaisuuksia voidaan hakea parametrien ”selector” ja ”source” perusteella, jotka ovat HTML-elementin osia. (ks. kuvio 4.) Näin sidotaan data editorista ja sivustosta toisiinsa. (Attributes n.d.)

```
{
    content: {
        type: 'string',
        source: 'text',
        selector: '.hero-text',
    }
}
```

Kuvio 4. Ominaisuus, jolla tyyppi, lähde ja valitsin.

2.8 Lohkon tuet (block supports)

Lohkolle on mahdollista asettaa Gutenberg-editorin mukana tulevia tukia, joilla lohkon toiminnallisuutta voidaan laajentaa. Tuet eli ”supports”-asetus on tyypiltään objekti, jolla on ennalta määrättyjä avainten ja arvojen pareja, joilla toiminnallisuudet saadaan käyttöön. (Block registration n.d.)

2.8.1 Tasaus ja leveys -asetus

Tasaus ja leveys (align) asetus ottaa vastaan listan eri tasauksista ”left”, ”right” ja ”center” sekä leveyksistä ”full” ja ”wide”. Jos lohkolle on annettu mikä tahansa näistä arvoista, näytetään lohkon työkalurivillä valikko, josta asetuksia voi vaihtaa. (Block registration n.d.)

2.8.2 Lohkoa rajoittavat asetukset

Tuilla on myös mahdollista poistaa lohkolta ominaisuuksia. ”Inserter”-tuen avulla voidaan estää lohkon näkyminen lohkolistalla, kun arvoksi annetaan ”false”, jolloin se voidaan lisätä ainoastaan koodin kautta. ”Reusable”-tuella määritetään, saako lohkosta luoda uudelleenkäytettävän. Oletuksena lohkoista saa tehdä uudelleenkäytettäviä, asetus on tällöin ”true”. (Block registration n.d.)

2.8.3 Lohkon yksilöimistä helpottavat asetukset

Lohkolle voidaan oletuksena antaa myös lisää luokkia, mutta tämäkin on mahdollista estää asettamalla ”customClassName”-tuki epätodeksi. Lohkolle voi asettaa myös ankkurin ”anchor”, jolla voidaan antaa lohkolle uniikki id. Näin voidaan sivulla luoda linkki, joka johtaa kyseiseen lohkoon. (Block registration n.d.)

2.9 Lohkomuunnokset (transforms)

Lohkoille on mahdollista määrittää lista lohkoista, joiksi ne voidaan muuttaa lennosta käyttämällä ”createBlock”-funktiota, joka tulee ”wp-blocks”-paketista. Lohko voidaan muuttaa tai siksi voidaan muuttua. Esimerkiksi paragrafi on oletuksena mahdollista muuttaa otsikoksi ja päinvastoin. Tällöin lohkossa olevat ominaisuudet siirretään lohkolta toiselle edellä mainitulla ”createBlock”-funktiolla. ”createBlock”-funktioon

voidaan yhdistää ”isMatch”-funktio, jolla voidaan tarkastaa ominaisuuksien yhteensopivuus lohkojen välillä. Jos ”isMatch” ei palauta arvoa ”true”, lohkon muunnosmahdollisuutta ei näytetä käyttäjälle. (Block registration n.d.)

2.10 Lohkotyylit

2.10.1 Rekisteröinti ja poisto (JavaScript)

Lohkotyylien avulla voidaan mahdollistaa olemassa oleville lohkoille vaihtoehtoisia tyylejä. Lohkotyylit antavat lohkon ylimmälle HTML-elementille eli niin sanotulle ”kääreelle” uuden luokan. Funktio ”wp.blocks.registerBlockStyle” ottaa kaksi parametriä, ensimmäinen on lohkon rekisteröintinimi. Toinen parametri on objekti, joka sisältää lohkon nimen, nimikkeen ja ”isDefault”-valinnan. (ks. kuvio 5.) Nimen perusteella luodaan uusi luokka, nimikettä käytetään käyttöliittymässä, ”isDefault”-valinnalla määritetään tyylin käyttö oletuksena. (Block filters n.d.)

```
wp.blocks.registerBlockStyle( 'cgb/hero', {
    name: 'hero-style-1',
    label: 'Hero Style Default',
    isDefault: true,
} );
```

Kuvio 5. Tyylin rekisteröinti.

Lohkoilta on myös mahdollista poistaa tyylejä käyttämällä funktiota ”wp.blocks.unregisterBlockStyle”. Kyseinen funktio ottaa kaksi parametria. (ks. kuvio 6.) Ensimmäinen parametri on lohkon rekisteröintinimi. Toinen parametri on poistettavan tyylin nimi. Lohkon tyylejä poistaessa käytöstä halutaan poisto-skripti ajaa viimeisenä, jotta

ei tapahdu ylikirjoitusta. Lohkon poiston voi kääriä ”wp.domReady” -funktioon, joka ajaa koodin vasta, kun sivun rakenne on ladattu. (Block filters n.d.)

```
wp.domReady( () => {
    /* Painikkeiden tyylit */
        wp.blocks.unregisterBlockStyle( 'core/button', 'default' );
        wp.blocks.unregisterBlockStyle( 'core/button', 'outline' );
        wp.blocks.unregisterBlockStyle( 'core/button', 'squared' );
```

Kuvio 6. Nappi-lohkon tyylien poisto käärittynä “wp.domReady”-funktioon.

2.10.2 Rekisteröinti ja poisto (PHP)

Rekisteröinti ja poisto on mahdollista suorittaa myös palvelimen puolella, jolloin käytetään funktioita ”register_block_style” ja ”unregister_block_style”. Tyylin poisto palvelimen puolella toimii kuten JavaScript-funktiokin, mutta sillä ei voi poistaa JavaScriptillä rekisteröityjä tyylejä käytöstä. (Block filters n.d.)

Rekisteröinti funktio ottaa vastaan samat parametrit, objekti on tässä tapauksessa taulukko. Taulukossa kulkeutuu yksi uusi arvopari ”inline_style” tai ”style_handle”, joilla määritetään lohkolle sisäinen tyyli tai tyylitiedoston nimi, jolla se on ladattu aiemmin. (ks. kuvio 7.) (Block filters n.d.)

```
register_block_style(
    'core/button',
    array(
        'name'         => 'dark-border-only',
        'label'        => __( 'Dark Border Only' ),
        'inline_style' => '.wp-block-button.is-style-dark-border-only { border: solid 2px #222; }',
    )
);
```

Kuvio 7. Lohkon elementin riville sijoittuvan tyylin rekisteröinti.

2.11 Editorin tyylit

Editorin tyylit tulevat samasta tiedostosta kuin itse sivuston tyylit, mutta ne kulkeutuvat kääntäjän läpi, joka muokkaa tietyt CSS-valitsimet Gutenberg-editorin vastaaviksi elementeiksi. Esimerkiksi valitsin ”body” kääntyy ”.editor-styles-wrapper” valitsimeksi. Näin editori pystyy käyttämään hyväksi sivuston tyylejä ilman, että editori rikkoutuu käyttökelvottomaksi. Sivuston tyylitiedosto täytyy viedä editorille käyttämällä funktiota ”add_action(”enqueue_block_editor_assets”)”, joka vie sen editorille kääntäjän kautta. (ks. kuvio 8.) (Theme Support n.d.)

```
// Add backend styles for Gutenberg.
add_action( 'enqueue_block_editor_assets', 'tailwind_to_editor' );
/**
 * Load Gutenberg stylesheet.
 */
function tailwind_to_editor() {
    wp_enqueue_style(
        'froggy-gutenberg',
        get_theme_file_uri( '/public/bundle.css' ),
        false
    );
}
```

Kuvio 8. Oman tyylitiedoston vieminen editorille.

2.12 Dynaamiset lohkot

2.12.1 Käyttötarkoitus

Dynaamisille lohkoille on kaksi pääasiallista käyttötarkoitusta. Lohkot, joiden sisällön tulee olla ajan tasalla, vaikkei itse artikkelia tai sivua olekaan päivitetty, ovat ensimmäinen tapaus. Toinen käyttötapaus koskee lohkoja, joiden koodipäivityksien tulee olla näkyvillä heti asiakaspäädyssä. Dynaamisilla lohkoilla voidaan varmistua, että lohko näyttää asiakkaille juuri siltä, kuin pitääkin heti päivityksen jälkeen. Normaalit lohkot jouduttaisiin päivittämään jokaiseen artikkeliin ja sivuun käsin, ennen kuin muutokset näkyisivät asiakaspäädyssä. (Creating dynamic blocks n.d.)

2.12.2 Toiminta

Lohkot, joiden asiakaspääty on dynaaminen palauttavat tyhjän ”null” tallennusfunktiossaan, jolloin lohkosta tallennetaan ainoastaan attribuutit tietokantaan. Nämä attribuutit viedään palvelimen puolella tehtävälle renderöintifunktiolle, jossa lohkon lopullinen ulkoasu kirjoitetaan auki PHP-kielellä. Lohkosta voidaan tallentaa HTML-versiokin, jota käytetään vian ilmetessä palvelimella renderöinnissä. (Creating dynamic blocks n.d.)

2.13 Vanhentuneet lohkot

Päivitettäessä jo käytössä olevien lohkojen rakennetta tai toimintoja, vanhoille lohkoille täytyy antaa jokin väylä päivittyä ja säilyttää toiminnallisuutensa eri versioiden välillä. Mahdollisia tapoja välttää lohkojen rikkoutuminen ovat lohkon uuden version rekisteröinti eri nimellä, jolloin vanhaan lohkoon ei kohdistu muutoksia, tai antaa uudelle versiolle ”deprecated”-ominaisuus. (Deprecated blocks n.d.)

Lohkoille voidaan asettaa useita vanhentuneita käsittelymalleja, joita käytetään silloin, kun lohkon validointi ei mene läpi tai ”isEligible”-tarkistus palauttaa arvon ”true”. Lohkon ”deprecated”-objektille voidaan antaa normaaliin tapaan ominaisuudet (attributes), tuet (supports) sekä tallennusfunktio (save) sekä lisäksi siirto (migrate) ja sopivuuden määritys (isEligible). Ominaisuudet, tuet ja tallennus eivät periydy automaattisesti, vaan ne täytyy määrittää käsin, jotta vältytään jäsentelyn virheiltä. (Deprecated blocks n.d.)

Jos uudella ja vanhalla lohkolla on eroavaisuuksia ominaisuuksien käsittelyssä, mutta ominaisuuksien tyyppi pysyy samana, voidaan suorittaa ominaisuuksien siirto (migrate). Tällöin saadaan jäsenneltyä vanhat ominaisuudet uuteen malliin sopiviksi. Ominaisuuksien siirtäminen pätee myös sisäisiin lohkoihin (innerBlocks). (Deprecated blocks n.d.)

2.14 Lohkopohjat (Block templates)

Lohkopohjat ovat taulukoita lohkoista, joille voidaan ennalta määrittää ominaisuuksia ja ne voivat olla sekä staattisia että dynaamisia. Lohkojen pohjia voidaan määritellä sekä JavaScriptillä että PHP:llä, ja molemmat noudattavat samankaltaista kaavaa. Päätaulukko sisältää lohkovanhemmat, jotka itsessään voivat sisältää taulukon lapsilohkoista. Lohkojen määrää pohjissa ei ole rajoitettu. (Templates n.d.)

Artikkelityypeille on mahdollista rekisteröinnin yhteydessä luoda lohkopohjia. (ks. kuvio 9.) Uusi “template”-määritys ottaa vastaan lohkopohjan PHP-muodossa ja “template_lock”-määrityksellä saadaan lukittua pohja joko kokonaan tai osittian. Arvolla ”All” voidaan pohja lukita kokonaan, jolloin uusia lohkoja ei voi lisätä, liikuttaa tai poistaa. Arvolla “Insert” voidaan poistaa mahdollisuus lisätä uusia lohkoja, jolloin olemassa olevia lohkoja saa kuitenkin siirrellä ja poistaa. Oletuksena lohkoja saa lisätä poistaa ja siirtää. (Templates n.d.)

```php
function froggy_register_example_post_type() {
    $args = array(
        'public' => true,
        'label'  => 'Examples',
        'show_in_rest' => true,
        'template' => array(
            array('core/image', array(
                'align' => 'left',
            )),
            array('core/columns', array(), array(
                array('core/column', array( 'width' => 25 ), array(
                    array('core/image', array() ),
                )),
                array('core/column', array( 'width' => 75 ), array(
                    array('cgb/bodytext', array() ),
                )),
            )),
        ),
    );
    register_post_type('example', $args);
}
```

Kuvio 9. Lohkopohjan määrittely rekisteröitävälle postityypille.

2.15 Uudelleenkäytettävät lohkot

Uudelleenkäytettävät lohkot mahdollistavat lohkon tai lohkoryhmän monistamisen, ja muutokset uudelleenkäytettäviin lohkoihin vaikuttavat kaikkiin lohkosta monistettuihin lohkoihin. Uudelleenkäytettävät lohkot tallennetaan piilotettuun artikkelityyppiin ja ovat dynaamisia lohkoja, jotka viittaavat lohkon artikkeli id:seen, palauttaen artikkeli sisällön kyseiselle lohkolle. (Key Concepts n.d.)

2.16 Lohkokohtaiset editorityökalut

2.16.1 Työkalupalkki

Kun käyttäjä valikoi lohkon, lohkon päälle tulee näkyviin työkalupalkki, josta on mahdollista esimerkiksi vaihtaa tekstin asettelua lohkokohtaisesti. Osa asetuksista kuten

tekstin asettelu on joillain lohkoilla esimerkiksi paragrafilla automaattisesti käytössä. Jotkin asetukset vaativat erikseen niiden käyttöönoton aiemmin mainitun ”add_theme_supports” funktion avulla. (Block toolbar and settings sidebar n.d.)

2.16.2 Sivupalkki

Sivupalkkiin asetetaan asetuksia, joita yleisesti käytetään vähemmän tai jotka vaativat enemmän näyttötilaa. Myös sivupalkissa olevien asetusten tulisi olla ainoastaan lohkokohtaisia. Asetukset, jotka muokkaavat lohkon sisältöä eivät kuulu sivupalkkiin. (Block toolbar and settings sidebar n.d.)

3 TailwindCSS

3.1 Mikä tailwindCSS on?

TailwindCSS, myöhemmin Tailwind, on hyvin muokattava CSS-kehys, joka antaa kaikki perusrakennuspalikat sivun ulkoasujen kehittämiseen. Tailwind ei anna valmiita tyylikomponentteja, vaan se tarjoaa valmiita yleishyödyllisiä CSS-luokkia, joilla on mahdollista luoda monimutkaisia ulkoasuja. (TailwindCSS n.d.)

Tailwind-kehystä käyttäessä luokat ovat ennalta määrättyjä ja niitä yhdistämällä saadaan kasattua monipuolisia tyylittelyjä, ilman omia CSS-määrityksiä. Muutoksien tekeminen muuttuu myös turvallisemmaksi, sillä CSS itsessään ei muutu. CSS-tiedoston koko pysyy huomattavasti hillitympänä, koska kaikelle ei tule omaa luokkaansa. Ulkoasun kehitys siirtyy tilanteeseen räätälöidystä tyyleistä käyttämään ennalta määritettyä design-systeemiä, joten ulkoasut ovat yhtenäisempiä. (Utility-First n.d.)

3.2 Toiminta

Tailwindin toiminnan ymmärtämiseksi on hyödyllistä ymmärtää hieman PostCSS toiminnasta, sillä Tailwind on periaatteessa PostCSS-lisäosa. PostCSS on kirjasto, joka muuttaa CSS:ää JavaScriptiksi ja JavaScritpiä CSS:ksi. CSS muunnetaan JavaScriptiksi, koska näin saadaan käyttöön laajempi logiikka, joka CSS:ltä puuttuu, esimerkiksi silmukat. (Vega 2019)

Lopulta JavaScriptiksi muunnettu tyylitiedosto, jonne kehittäjä on voinut asettaa omia silmukoita ja lainalaisuuksia, muunnetaan se takaisin CSS-tiedostoksi noudattaen JavaScriptiltä tulevaa logiikkaa. (Vega 2019)

CSS-tiedostoa voidaan ajatella puuna, jossa jokainen valitsin (selector) on oma haaransa, jonka sisällä on deklaraatioita omina haaroinaan. PostCSS muuntaa CSS:ää edellä mainitun kaltaiseksi puuksi, jota PostCSS voi silmukoida läpi tarvittaessa ja näin lisätä uusia tyylimääritteitä tietyin ehdoin. Tailwind sisältää vastaavia ehtoja, joiden mukaan PostCSS rakentaa sen luokkia. Luokat rakentuvat Tailwindin asetustiedoston mukaan, jonne kehittäjän on mahdollista kirjoittaa omia luokkamalleja. (Vega 2019)

3.3 Responsiivisuus

Kaikki Tailwindin luokat on mahdollista aktivoida ruutukokojen pysäytyspisteiden (breakpoints) mukaan. Oletuksena pysäytyspisteitä on neljä ja ne noudattavat yleisimmin käytössä olevien laitteiden ruutukokoja. Oletuksena Tailwind käyttää niin sanottua ”mobile first”-periaatetta, jonka mukaan käyttöliittymä suunnitellaan ensin puhelin koossa, jonka jälkeen siirrytään askel kerrallaan kohti työpöytänäkymää. (Responsive Design n.d.)

Tailwindin lähestymistapa on hyvin samanlainen, kuin Bootstrapissa tai Foundationissa. Kun luokalla ei ole etuliitettä se on toiminnassa kaikilla ruutukoilla, mutta jos luokalla on etuliite, se toimii kyseisestä koosta ylöspäin. (Ks. kuvio 9.) (Responsive Design n.d.)

```
flex flex-col md:flex-row align-center justify-center
```

Kuvio 10. Tailwind luokan vaihto ruudun leveyden mukaan ja esimerkki etuliitteestä.

3.4 Pseudoluokkien käsittely

Pseudoluokkien eli focus, hover ja niin edelleen käyttö Tailwindissä toimii samalla tavalla kuin pysäytyspisteidenkin käyttö. Oletuksena kehys ei sisällä kaikille luokille pseudoluokkia tiedostokoon takia, mutta yleisimmät käyttökohteet on otettu huomioon. Pseudoluokat, joita kehys ei tue suoraan on mahdollista lisätä kirjoittamalla lisäosia. Pseudoluokkia ja responsiivisuus asetuksia voidaan myös yhdistää. (Pseudo-Class variants n.d.)

3.5 Tyylikomponentit

Tyylikomponentteja suositellaan käytettäväksi, kun samoja elementtejä toistetaan useaan otteeseen sivuilla. Näin vältytään pitkien luokka jonojen kirjoittamiselta ja myös ylläpito helpottuu, kun muutokset voidaan ajaa yhdestä komponentista useaan paikkaan. Sivuston elementtejä suositellaan jakamaan esimerkiksi JavaScript-komponentteihin, joita toistetaan ja joita voidaan muokata helposti. (Extracting Components n.d.)

Tailwind tarjoaa käyttöön ”@apply”-toiminnon, jolla voidaan omalle luokalle antaa Tailwind-luokkien ominaisuuksia. (ks. kuvio 11.) Tällaiset luokat tulee asettaa heti Tailwindin komponenttien määrittämisen jälkeen, jotta parsinnassa ei synny virheitä. (Extracting Components n.d.)

```
@tailwind base;

@tailwind components;

.tns-nav{
    @apply w-1/2 mx-auto flex justify-center;
    button{
        @apply bg-gray-700 p-2 mx-1 rounded-full;
    }
    .tns-nav-active{
        @apply bg-primary;
    }
}

@tailwind utilities;
```

Kuvio 11. TailwindCSS @apply toiminto sekä parsittavan CSS-tiedoston rakenne.

4 React.js Gutenberg kontekstissa

4.1 Koodin parsinta Gutenbergille

Kuten normaalin React.js-projektin kanssa, täytyy React-pohjainen JavaScript pyörittää Babel-kirjaston kautta. Babel muuntaa modernin ecmascript 6 (ES6)-syntaksia käyttävän JavaScriptin ecmascript 5 muotoon, joka yleisesti on paremmin tuettu. Kuitenkin Gutenberg-ympäristössä Babel-kirjastolle täytyy erikseen kertoa, että elementit käännetään WordPress-elementeiksi eikä React-elementeiksi. Määritys saadaan

käyttöön asettamalla “pragma” parametrille arvo “wp.element.createElement” .bablerc-tiedostoon. (Bell 2019.)

4.2 Create-Guten-Block

Create-Guten-Block (CGB) on valmis aloituspaketti Gutenberg-lohkojen kehittämiseen Reactilla. CGB on Ahmad Awaisin kehittämä paketti, joka sisältää tarvittavat säädökset lohkon kehittämiseen. Paketti liitettään WordPressiin lisäosana. (Schenck 2019)

CGB-paketti mahdollistaa kehittämisen heti, ettei tietämystä kaikkeen ympäröivään teknologiaan, kuten Webpackiin, tarvita aloittaakseen Gutenberg-lohkojen kehityksen. Näin voidaan aloittaa ensin lohkokehitys ja vasta tarpeen vaatiessa paneutua ympäröiviin teknologioihin. (Schenck 2019)

5 Froggy-blocks, Gutenberg kohtaa TailwindCSS:n

5.1 Yleistä

Froggy-blocks -lohkot rakennettiin Greate-Guten-Blockin tarjoaman kehitysympäristön päälle. Ennen varsinaisen lisäosan kehitystä kehitettiin vanhasta Trimedian pohjateemasta Gutenbergiä tukeva versio. Froggy-blocksin lohkot kehitettiin aluksi Create-Guten-Block pohjan perusteella, jonka jälkeen siirryttiin käyttämään Gutenbergin koodipohjan toimintoja mahdollisimman paljon. Kehityksessä keskityttiin Gu-

tenbergin opiskelun jälkeen hyvin paljon TailwindCSS:n sulauttamiseen suoraan lohkoihin, mutta loppua kohden kehitys ajautui enemmän tukemaan Tailwindin komponentteja, jotka kirjoitetaan käsin projektin tarpeiden mukaan. Lähes kaikki luotettava ja ajantasainen tieto, jota kehityksen aikana käytettiin, löytyi joko WordPress.org sivustolta löytyvästä Block Editor Handbookista tai WordPress-tiimin Gutenberg GitHubista lähdekoodista.

5.2 Teeman kokoaminen

Froggy-blocks Gutenberg-lisäosan kehitys aloitettiin selvittämällä mitä vaatimuksia Gutenbergillä on koskien pohjateemaa. Tarkoituksena oli ottaa vanha Trimedialla käytössä ollut pohjateema ja päivittää siitä Gutenberg-teema. Vanhasta teemasta löytyi Tailwind jo ennestään, joten pystyttiin keskittymään pelkästään Gutenbergin-vaatimusten täyttämiseen. Vanhasta teemasta perittiin myös sen kehitysympäristö, joka hoitaa Tailwindin rakentamisen ja paljon muuta. Pohjateemaa kehitettiin kolmen henkilön voimin, joiden kaikkien oli tarkoitus tutkia Gutenbergin tarjoamia mahdollisuuksia eri kanteilta.

Normaalisti lohkoja voitaisiin rakentaa suoraan teemaan, mutta tahdoimme pitää uuden teeman mahdollisimman kevyenä. Teemaa käytettäisiin paljon myös sivuilla, joissa Gutenberg-editori ei ole käytössä, joten lohkot jätettiin erilliseen lisäosaan. Tämä oli sopiva valinta myös sen takia, että toisinaan haluttiin käyttää vain tietyllä tekniikalla rakennettuja lohkoja.

Teemaa laajennettiin Gutenbergin sisäänrakennetuilla ominaisuuksilla “add_theme_support”-funktioiden avulla. Tärkeimpinä pohja oman väripaletin rekisteröintiin ja lohkojen leveysasetusten käyttöönotto sitä tukeville lohkoille.

Lohkon leveyksille kehitettiin oma CSS-malli, jota olisi helppo muokata tarpeen vaatiessa. Kyseinen CSS-malli käyttää hyväkseen funktiota “calculate”, jolla lasketaan erot koko näyttöpäätteen leveyden ja näyttöpäätteen käytettävän leveyden välillä. Koska “alignwide”-luokan on tarkoitus rikkoa ylemmältä elementiltä peritty leveys, täytyy leveys pakottaa sitä suuremmaksi käyttämällä näyttöpäätteestä laskettavaa leveyttä. Käytettäessä pelkästään leveyttä “100vw” esiintyy ongelmia, jos sivulla esiintyy samaan aikaan myös vierityspalkki, joten piti selvittää mikä vierityspalkin leveys on ja poistaa sen leveys koko näyttöpäätteen leveydestä. (ks. kuvio 11.) Sama toimenpide toistettiin myös marginaalille.

```
.alignwide, .alignfull {
    /* 50% of containing elements width - 50% of scr
    margin: 0 calc(50% - 50vw + (100vw - 100%) / 2);
    /* screen width - scrollbar width */
    max-width: calc(100vw - (100vw - 100%));
    width: calc(100vw - (100vw - 100%));
}
.alignfull{
    .alignfull{
        margin: 0 auto;
        width: 100%;
    }
}
@screen lg {
    .alignwide {
        margin: 0 auto;
        max-width: 1024px;
        width: 1024px;
    }
}
```

Kuvio 12. Gutenbergin oletus leveyden säätö CSS-luokat muokattuna omaan malliin.

5.3 Lohkokehitysympäristö

Lohkojen kehittämisympäristöksi asennettiin Create-Guten-Block paketti Node Package Managerista, jonka avulla voitettiin paljon aikaa ympäristön säädöiltä. Kehitysympäristö sisältää tarvittavat asetukset Webpackiin, jolla esimerkiksi Reactin käyttämä JSX käännetään selainten tukemaan muotoon JavaScript ES5-syntaksiin. Ympäristö sisältää myös tarvittavat määritykset ja tiedosto WordPress-lisäosan rekisteröintiin, jolloin sitäkään ei tarvitse tehdä itse.

Koko kehityksen ajan WordPressissä oli käytössä Gutenbergin kehitysversio, joka kulkee huomattavan paljon edellä loppukäyttäjille tarkoitettua versiota. Kehittäjäversiosta löytyi myös satunnaisesti virheitä, jotka estivät tiettyjen asioiden tekemisen, jonka takia ei pystytty tekemään kaikkia haluttuja toiminnallisuuksia. Kehittäjäversion käyttö kuitenkin omia lohkoja kehittäessä oli kannattavaa, jotta tiedettiin mitä lohkoja on tulossa, ettei niitä tarvitse rakentaa itse.

5.4 Lohkokehitys alussa

Lohkokehitys alkoi Create-Guten-Blockin tarjoaman lohko kehyksen päälle, johon alettiin ohjelmoimaan lohkon toimintoja melko sokeasti ensi alkuun. Lohkokehityksen alussa kollegat alkoivat tutkia toisia lohkokehitysmenetelmiä. Ensimmäiset lohkot kehitettiin ilman rajoitteita, sillä niitä ei ollut tarkoitus liittää mihinkään asiakasprojektiin. Niiden tarkoitus oli näyttää, mitä lohkoilla voi tehdä ja miten lohkoja rakennetaan. Samalla oli tarkoitus tutustua niiden toimintaan ja kerätä kysymyksiä, joiden avulla kehitystä vietäisiin eteenpäin.

Ensimmäisien lohkojen kehityksessä keskityttiin lähinnä itse lohkoihin ja editorin toiminnallisuuksiin. Tailwindin sulauttaminen jätettiin aluksi kokonaan pois kehityskuvioista, koska sen ei haluttu hidastavan Gutenbergin haltuunottoa.

5.5 Ensimmäiset lohkot asiakasprojekteihin

Asiakasprojekteihin tarkoitetut lohkot rakennettiin Tailwind-integrointi etusijalla. Ensimmäiset Tailwind-lohkot rakennettiin siten, että lohkon sivupaneeliin lisättiin vipuja ja valintoja, joilla vaikutettiin lohkon ominaisuuksiin. Näiden ominaisuuksien muuttumisen perusteella määritettiin lohkolle Tailwindin mukaisia luokkia. Luokat olivat suureksi osaksi yleispäteviä, esimerkiksi lohkon osien leveyksien määrittämisiä tai värien muuttamista.

Menetelmä oli erittäin hyvä juuri yleisien asetusten kannalta, mutta lisättäessä valinta jonkin tarkemman asetuksen määrittämiseksi alkoivat asiat mutkistumaan. Koska lohkoille määritettiin jo editorissa Tailwind-luokkia, joutui niitä yliajamaan esimerkiksi mobiili näkymässä. Tästä lopulta koitui enemmän haittaa kuin hyötyä, kun editorin CSS-määrityksiä piti jatkuvasti yli kirjoittaa erikoisemmissa taitoissa.

5.6 Spurtti Tailwind-integraatiota

Edellisten kokemusten perusteella päädyttiin kehittämään eteenpäin Tailwindin integrointia lohkoihin. Tailwindin asetustiedosto sisältää kaiken tarvittavan Tailwind-luokkien rakentamiseen, joten tätä asetustiedostoa alettiin käyttää lohkojen sivupaneelin asetusten perustana. Kaikkiin itse tehtyihin lohkoihin tehtiin yksi sivupaneeli, josta voitiin kutsua tarvittavia asetuksia.

Edellisestä asiakasprojektista oppineena lohkoille rakennettiin asetuksia, jotka tukivat mobiilikokoja ja responsiivisuutta paremmin. Edelleen tyylittelyn perustana oli lohkon ominaisuuksien kautta eri elementtien luokkien nimien muuttaminen. Näitä asetuspaneeleita alettiin kasaamaan yhteen react-komponenttiin, josta tehtiin yleinen sivupaneeli kaikille lohkoille. Komponenttia oli mahdollista kutsua lohkon omi-

naisuuksilla, jolloin komponenttiin rakennettu logiikka osasi palauttaa kaikki tarvittavat asetuspaneelit. Kehityksen loppuvaiheilla tämä ominaisuus kuitenkin poistettiin käytöstä ja kaikille lohkoille rakennettiin omat sivupaneelit.

Koska Tailwind todella haluttiin integroida suoraan lohkoihin, rakennettiin Reactilla parsimia eri asetusten hakemiseen lohkon sivupaneeliin. (ks. kuvio 12.) Parsijan ansiosta muutokset Tailwindin asetustiedostoon näkyivät suoraan lohkoihin. Esimerkiksi luotaessa uusi väri Tailwindiin, saatiin se heti käyttöön lohkoille. Sama tehtiin useimmille asetuksille kuten marginaaleille ja leveysasetuksille. Tämä ominaisuus säilyi muutamilla lohkoilla kehityksen loppuun asti supistettuna.

```
37  export function ColorClasses (prefix) {
38      return flatten(Object.keys(theme.colors).map( key => {
39          if(typeof(theme.colors[key]) == 'object'){
40              return Object.keys(theme.colors[key]).map( variationKey => {
41                  return variationKey
42              }).map(variation => {
43                  return prefix+'-'+key+'-'+variation
44              });
45          }
46          else{
47              return prefix+'-'+key
48          }
49      }))
50  }
```

Kuvio 13. Rekursiivinen funktio, joka palauttaa listan Tailwindin värien luokista.

Parsijan kohdalla kyse on rekursiivisesta funktiosta, jolla käydään läpi kaikki halutut Tailwindin asetustiedoston ominaisuudet. Funktio palauttaa listan, jolla ei ole sisäkkäisiä listoja. Tailwindin asetuksista tuleva lista voi sisältää loputtomasti sisäkkäisiä listoja, mutta yleisesti listojen syvyys on kaksi kerrosta.

Editoripuolella alkoi esiintyä ongelmia, sillä asetteluita sai laitettua huomattavasti enemmän, jolloin asettelut myös editorissa menivät helpommin rikki. Kun asettelut menivät editorissa rikki, koko editorin toiminta kärsi. Editorin tyylit ladattiin vielä tässä kehityksen vaiheessa täysin väärällä tavalla ilman WordPressiin sisäänrakennettua tyyliparsinta, joka muuttaa sivuston tyylitiedoston editorille sopivaksi. Asiasta ei tässä vaiheessa yksinkertaisesti tiedetty, eikä vielä osattu etsiä.

Lohkoja kehitettiin projektikohtaisesti ja toivottiin, että niitä voidaan käyttää vielä tulvaisuudessa. Valitettavasti perusteellisen suunnittelun puute johti siihen, että lohkot eivät olleet tarpeeksi yleispäteviä.

5.7 Kasvukipujen jälkeen

Kehityksessä kahden takaiskun jälkeen päätettiin pitää taukoa ja keskityttiin pelkästään Gutenbergin opiskeluun Block Editor Hanbookin avulla ja tutkimalla Gutenbergin omaa lähdekoodia ja lohkojen rakennustapaa Gutenbergin GitHubista. Tutkimusta tehtiin noin kaksi viikkoa, jonka jälkeen opittuja menetelmiä alettiin laittaa käytäntöön.

Tailwind-integraatio mietittiin kokonaan uudelleen. Enää Tailwind ei olisi suoraan yhteydessä lohkoon, vaan lohkot rakennettaisiin tukemaan Tailwindin komponentteja mahdollisimman hyvin. Tämä tarkoitti käytännössä sitä, että kaikilla lohkon elementeillä oli oma luokkansa. (ks. kuvio 13.) Lohkoissa vähennettiin Tailwindin luokkien käyttöä rankasti, jotta säästyttiin CSS:n ylikirjoittamiselta.

```
<li class={"automatik-card " +  widthClass}>
    <div className="card-content">
        {showImage &&(
            <div className="card-image" style={
            </div>
        )}
        {showName &&(
            <h3 className="card-name">
                {item.name.substring(0, 35)}
            </h3>
        )}
```

Kuvio 14. Tailwindin komponetti mallia tukeva lohko.

Koska lohkot käyttivät nyt Tailwindin komponenttimallia, niille oli mahdollista rakentaa teemaan valmiiksi komponenttirakenteet ja näin nopeuttaa taittajan työtä. (ks. kuvio 14.) Perustamalla taitto Tailwindin komponentteihin, ylikirjoittamista tarvitaan enää ainoastaan poikkeustilanteissa. Tämän muutoksen ansiosta voidaan lohkolle rekisteröidä tyyli, joko teemassa tai itse lohkoa rekisteröitäessä, jolla voidaan antaa lohkon ylimmälle elementille tyyliä vastaava luokka. Tämän luokan avulla voidaan koko lohkon tyylit muuttaa, siten että valmiilla sivustolla editorissa tyylin vaihtamiseen tarvitaan vain muutama hiiren painallus. Samalla tyylit ovat tiukasti taittajan käsissä, eikä sivuston muokkaaja vahingossa saa rikottua asetteluita.

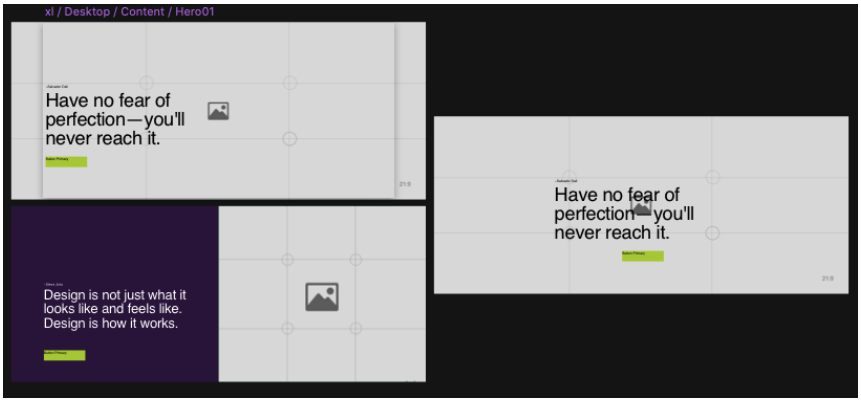
```
.automatik-card{
    @apply list-none inline-block p-4;
    .card-content{
        @apply bg-gray-300 shadow flex flex-col;
        .card-image{
            @apply h-64 bg-cover bg-center pb-2 block;
        }
        .card-name{
            @apply text-center pb-2;
        }
```

Kuvio 15. Tailwindin komponenttimallia käyttävän lohkon CSS-määritykset

5.8 Lohkojen rakentaminen design-malleista

Tutkivan kehityksen viimeinen vaihe oli luoda design-tiimin malleista lohkot, jotta saataisiin lisättyä yhtenäisyyttä tekniikan ja designin välillä. Kyseiset lohkot ovat sellaisia, joita käytetään lähes jokaisella uudella sivustolla. Samaan aikaan kehitettiin myös lohkoja, joissa yhdistetään yrityksen omakehitteinen verkkokauppa sekä Gutenberg.

Design-mallien pohjalta kehittämien tapahtui käytännössä siten, että design-tiimi kehitti ensin itselleen pohjia tietylle design elementille. Näistä pohjista rakennettiin yksi lohko, jonka tuli olla tyyliteltävissä kaikiksi mallin mukaisiksi elementeiksi. (ks. kuvio 16.) Tähän käytettiin juuri Tailwind-komponentteja ja eri renderöintimalleja lohkolle.



Kuvio 16. Design-tiimin mallit “hero”-lohkosta.

Lohkojen ollessa tarpeeksi yksinkertaisia kuten leipäteksti voitiin käyttää pelkkiä itse rekisteröityjä lohkotyylejä, mutta monimutkaisempien lohkojen kohdalla täytyi koko lohkon elementit järjestellä uudelleen. Sivun aloittava ”hero”-lohko esimerkiksi oli

parempi rakentaa siten, että eri tyylit muodostuivat sekä itse rekisteröidyistä lohkotyyleistä että lohkon rakenteen muutoksilla, koska turhia elementtejä ei haluttu jättää lohkoihin.

Lohkon elementtirakenteen muutokset tapahtuivat perinteiseen Reactin tyyliin, eli tarkastelemalla tiettyä muuttujaa, jonka perusteella palautetaan elementit datan kera halutulla tavalla. Renderöinnin ollessa muuttujan varassa piti ottaa huomioon myös lohkon tallennuksessa samojen renderöintien rakentaminen, jotta lohko näyttäisi samalta editorissa ja sivustolla.

6 Tulokset

6.1 Näkyvät tulokset

Tutkivan kehityksen konkreettisina tuloksina toimivat kehitetyt lohkot, joita kehitettiin lopulta seitsemän kappaletta. Näistä seitsemästä lohkosta kaikki eivät ole kehitetty viimeisimmän kehitysmenetelmän mukaan, vaan ne noudattavat vanhempia menetelmiä, koska ne toimivat tarpeeksi hyvin. Osa lohkoista on jälkeenpäin korvattu vastaavilla uusien Gutenberg-versioiden mukana tulleilla lohkoilla, kuten esimerkiksi ryhmälohkot.

6.2 Kerätty tietotaito

Kehitystyön aikana saatiin laajahko käsitys Gutenberg-editorista sekä sen toiminnasta. Gutenberg-editori pohjimmiltaan on React.js:n avulla toteutettu staattisten sivujen generaattori, jossa sisällönhallinta on toteutettu lohkoilla. Lohkoista kasattavan näkymän on tarkoitus mallintaa lopullista sivua mahdollisimman hyvin. Kehityksen

aikana selvitettiin myös, että sivun staattisuutta voidaan rikkoa käyttämällä dynaamisia lohkoja, joilla on mahdollista rakentaa esimerkiksi artikkeleista arkistoja yhden lohkon avulla.

TailwindCSS:n sulauttaminen Gutenberg-editoriin osoittautui mahdolliseksi monellakin eri tyylillä. Lopullisista lohkoista vain muutama käyttää hyväksi Tailwindin sulauttamista lohkoasetuksissa, sillä sen käytöllä on rajalliset hyödyt. Esimerkiksi, värien hakeminen Tailwindin asetustiedostosta osoittautui erittäin hyödylliseksi ominaisuudeksi, mutta sitä erikoisempia asetuksia ei juurikaan kannattanut sulauttaa lohkoon.

6.3 Nykyisen menetelmän hyötyjä

6.3.1 Säästöt kehityksessä

Vanhaan työtapaan verrattaessa Gutenberg ja siihen räätälöidyt lohkot keskikokoisissa projekteissa voivat karkeasti säästää noin 10 – 20 % työajasta totuttelun jälkeen. Vanha työtapa perustui täysin Advanced Custom Fields -lisäosan ympärille, jolla rakennettiin sivuille hieman lohkojen kaltaisia editori näkymiä. Vanhan tekniikan ongelma oli niiden projektikohtaisuus, sillä kentät rakennettiin aina vain yhdelle projektille, joten jokainen projekti alkoi ACF-kenttien luonnilla, jonka jälkeen piti rakentaa sivun renderöintilogiikka asiakaspäätyyn sen lisäksi.

Uuden menetelmän ansiosta sivun sisällöt voidaan asettaa valmiista lohkoista sivun editoriin, josta ne kääntyvät jo valmiiksi oikein myös asiakaspäätyyn. Tämän jälkeen työvaiheet jatkuvat melko samanlaisina molemmilla menetelmillä, tyylittely, sisällön syöttö ja tyylittelyn viimeistely.

Sisäisen verkkokaupan sulauttaminen Gutenberg-lohkoon säästää tulevaisuudessa huomattavan määrän työtunteja siihen liittyvissä WP-projekteissa. Lohkon käyttöönottaminen vaatii noin viiden minuutin työn, jolloin se on käyttövalmis näyttämään verkkokaupan tuotteita. Lohko täytyy tämän jälkeen vain taittaa oikean näköiseksi.

6.3.2 Säästöt sisällönsyötössä

Uusi menetelmä säästää aikaa myös sivuston omistajalta, koska lohkoja on luonnollisempaa muokata kuin valkoisia ACF-kenttiä, joilla ei ole mitään sidoksia lopulliseen ulkoasuun. Koska sivuston omistajan kynnys muokata sisältöä on pienempi, kehittäjät säästyvät kyseiseltä vaivalta, jolloin aika voidaan käyttää parempaan tekemiseen. Kyseiset säästöt riippuvat kuitenkin viimekädessä täysin sivuston omistajan halukkuudesta syöttää itse sisältöä, mutta kynnys siihen on asiakasprojektien perusteella madaltunut.

6.3.3 Ylläpito

Uutta menetelmää voidaan ylläpitää päivittämällä lisäosaa sitä mukaa kun uusia ominaisuuksia tai muokkauksia rakennetaan lohkoihin. Lohkojen muutokset tulevat käyttöön automaattisesti, jos mahdollista, muussa tapauksessa lohko pyydetään päivittämään tai luomaan uudelleen. ACF-menetelmässä ylläpito on erittäin saman kaltainen, mutta esimerkiksi uudet ominaisuudet pitää käydä itse liittämässä jo tehtyihin kenttäryhmiin.

Uuden menetelmän ylläpitäminen sisällön ja ulkoasun kannalta on kuitenkin helpompaa, koska lohkot ovat aina samaa kaavaa toistavia, toisin kuin ACF-kentät, joiden malli vaihtelee tekijäkohtaisesti. Kehittäjien on helpompaa lähteä tekemään esimerkiksi tyylimuutoksia, koska lohkot toistavat samaa kaavaa projektista toiseen, jolloin säästytään turhalta koodin tutkimiselta. Ainoat muuttuvat asiat projektikohtaisesti

ovat lohkojen sisältö ja tyylimääritykset, jotka eivät muunna lohkon varsinaista rakennetta.

7 Pohdinta

7.1 Tavoitteiden toteutuminen

Päätavoitteena oli yhdistää vakiintuneita hyväksi todettuja prosesseja Gutenberg-editorin kanssa työskentelyyn ja kartoittaa editorin tarjoamia mahdollisuuksia Käytännössä tämä tarkoitti editorin ja lohkojen kehittämistä tukemaan TailwindCSS-tyylikirjastoa. Kehityksen tuloksena toivottiin vähintään viittä Tailwindin kanssa sulautettua lohkoa, joita voitaisiin käyttää tulevissa projekteissa.

Tavoitteet olivat kohtuulliset ja lopulliset tulokset täyttävät nämä tavoitteet, vaikkei lopullisissa lohkoissa olekaan Tailwindiä sulautettu täysin lohkoihin. Kun tehtävä työlle tuli, kellään ei ollut aiheesta niin hyvää käsitystä etteikö tavoitteet olisi hieman muuttuneet kehityksen edetessä. Joka tapauksessa Tailwind on nyt olennainen osa kehitettyjä lohkoja ja lohkot tukevat Tailwindin tyylittelyn periaatteita.

7.2 Onnistumiset ja epäonnistumiset

Työn aikana onnistuttiin keräämään paljon tietotaitoa Gutenbergistä ja sen tarjoamista mahdollisuuksista. Suurin onnistuminen kehityksen aikana oli oivaltaa, ettei Tailwindiä edes lopulta kannata juurruttaa liian syvälle lohkon toimintaan, koska sen käyttö ei välttämättä ole kaikissa projekteissa optimaalista. Näin päädyttiin lopulliseen kehitysmenetelmään, jonka mukaan lohkot rakennetaan ensisijaisesti tukemaan

Tailwind-komponenttirakennetta ja toissijaisesti sisältämään Tailwindin asetustiedostosta noudettavia lohkoasetuksia. Kyseinen käänne on erityisesti tulevaisuuden kannalta potentiaalisesti merkittävä, sillä lohkot eivät ole enää riippuvaisia Tailwindistä vaan ne sisältävät sille parhaan mahdollisen tuen.

Työn suurimmat epäonnistumiset sijoittuvat ehdottomasti tutkimusakselille, sillä se jäi varsinkin aluksi aivan liian vähälle. Kehitystä puskettiin vain eteenpäin ilman syvempää paneutumista asiaan, jonka tuloksena menetettiin työtunteja. Asian voi myös kääntää koko työn suurimmaksi onnistumiseksi, sillä asian tajuamisen jälkeen tutkimus ja kehitys alkoivat kulkea hyvin tiukasti käsikädessä. Tämän tajuaminen tuntui kasvattavan ammattimaisuutta myös yleisesti, eikä pelkästään tämän projektin kohdalla.

7.3 Rajoitukset

Rajoitukset tutkimuksessa koostuivat enimmäkseen luotettavien lähteiden etsinnästä, toimeksiannon aihe on melko tuore ja siitä on vaikea löytää luotettavaa ja samaan aikaan ajantasaista tietoa. Ja teknologioiden sulauttamisesta ei löytynyt ainakaan kehityshetkellä mitään luotettavaa tietoa suoraan, vaan kaikki palaset piti poimia eri teknologioiden dokumentaatioista. Tästä syystä, valtaosa tutkimuksesta keskittyi teknologioiden omiin dokumentaatioihin, eikä niinkään alan kirjallisuuteen. Tämä oli toisaalta hyvä asia, että pystyi olemaan suhteellisen varma tiedon paikkansapitävyydestä, mutta toisaalta näkökulma jäi melko suppeaksi. Kuitenkin kehittäessä avoimen lähdekoodin tuotetta eteenpäin, tarkin ja ajantasaisin totuus löytyy lähdekoodista ja sen ymmärtämisestä.

7.4 Tuloksista

Kehityksen aikana tuloksia alettiin hyödyntää sitä mukaa kun niitä saatiin, valitettavasti tämä tarkoitti sitä, että joissain tapauksissa aiheutui lisää työtunteja, koska kehitetyt lohko eivät olleet kehitetty uuden mallin mukaisesti. Loppua kohden tulokset alkoivat olla jo siinä vaiheessa, ettei kehitetyt lohkot hidastaneet työtahtia. Lohkoilla on potentiaalia säästää työaikaa tulevaisuudessa melko paljon, koska niitä voidaan tyylitellä ennalta karkeasti ja antaa asiakkaalle käytettäväksi sisällön tuoton yhteydessä. Jonka jälkeen sivuston taittajalla on mahdollisuus lisätä lohkoille lopulliset tyylit sisällön ympärille.

Jatkossa kehityksessä keskitytään mahdollisimman paljon jo kehitettyjen lohkojen ja Gutenbergin valmiiden lohkojen kehittämiseen. Käytännössä kehitys muuttuu siten, ettei enää keskitytä niinkään uusien lohkojen rakentamiseen, vaan vanhojen parantaluun uuden kehitysmenetelmän mukaan ja samalla niin sanottujen ”core”-lohkojen laajentamiseen. Näin vältytään mahdollisimman paljon pyörän uudelleenkeksimiseltä, jota jo hieman ehti tapahtua joidenkin lohkojen kohdalla.

Lähteet

Bell, A. 2019. Learning Gutenberg: Setting up a Custom webpack Config. Viitattu 15.11.2019. https://css-tricks.com/learning-gutenberg-6-setting-up-a-custom-webpack-config/

Attributes. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/block-api/block-attributes/

Block filters. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/filters/block-filters/

Block Controls: Block Toolbar and Settings Sidebar. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/tutorials/block-tutorial/block-controls-toolbar-and-sidebar/

Block Editor Handbook. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/

Block registration. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/themes/block-registration/

Creating dynamic blocks. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/tutorials/block-tutorial/creating-dynamic-blocks/

Deprectated blocks. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/block-api/block-deprecation/

Edit and save. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/block-api/block-edit-save/

Key concepts. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/key-concepts/

Templates. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/block-api/block-templates/

Theme support. N.d. Block Editor Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/block-editor/developers/themes/theme-support/

Democratize Publishing. N.d. WordPress.org. Viitattu 07.11.2019. https://wordpress.org/about/

Extracting Components. N.d. TailwindCSS Documentation. Viitattu 07.11.2019. https://tailwindcss.com/docs/extracting-components

Gowans, J. 2016. The Genius of WordPress (and why it's doomed). Viitattu 07.11.2019. https://medium.com/@jasongowans/the-genius-of-wordpress-and-why-it-s-doomed-d9fd571712fe

Paschal, E. 2017. Introducing TailwindCSS. Viitattu 08.11.2019. https://scotch.io/@paschaldev/introducing-tailwindcss

Pseudo-Class variants. N.d. TailwindCSS Documentation. Viitattu 07.11.2019. https://tailwindcss.com/docs/utility-first/

Schema. N.d. REST API Handbook. Viitattu 07.11.2019. https://developer.wordpress.org/rest-api/extending-the-rest-api/schema/

Responsive Design. N.d. TailwindCSS Documentation Viitattu 07.11.2019. https://tailwindcss.com/docs/responsive-design

Schenck Lara. 2019. Learning Gutenberg: A Primer with create-guten-block. Viitattu 08.11.2019. https://css-tricks.com/learning-gutenberg-3-primer-with-create-guten-block/

Most CSS frameworks do too much. N.d. TailwindCSS. Viitattu 07.11.2019. https://tailwindcss.com/#what-is-tailwind

Utility-First. N.d. TailwindCSS Documentation. Viitattu 07.11.2019. https://tailwindcss.com/docs/utility-first/

Vega M. 2019. Parsing Open Source: Tailwind CSS. Viitattu 08.11.2019. https://dev.to/mariowhowrites/parsing-open-source-tailwind-css-39j7