WordPress Gutenberg -sisältölohkojen kehittäminen

Joona Mellin

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2023

Tiivistelmä

Tekijä(t)
Joona Mellin

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
WordPress Gutenberg -sisältölohkojen kehittäminen

Sivu- ja liitesivumäärä
50 + 2

Tämä toiminnallinen opinnäytetyö selvittää, miten WordPress Gutenberg -sisältölohkoeditorille kehitetään uusia sisältölohko -komponentteja. Opinnäytetyö on rajattu siten, että lukijan oletetaan tuntevan WordPress-sisällönhallintajärjestelmä sekä sen kehityksen perusteet. Opinnäytetyössä kuvataan lohkokehityksen prosessi suunnitelmasta lopputuotteeseen asti, käsitellen prosessin aikana kehitysympäristön perustaminen, lisäosan ja lohkokategorian luominen, sekä itse lohkojen rekisteröiminen.

Tietoperustan ensimmäisessä luvussa kuvaillaan Gutenberg -lohkoeditorin toimintalogiikkaa, käyttöä, taustoja sekä tulevaisuutta. Luvun aikana tarkastellaan myös editorin mukana tulevia sisältölohkoja. Tietoperustan toinen luku käsittelee lohkokehityksessä käytettäviä teknologioita, ja paneutuu tarkemmin erityisesti React-ohjelmakirjastoon sekä sen käyttämään JSX-syntaksiin. Lisäksi toisessa luvussa käsitellään niitä WordPressin sisäisiä rajapintoja, joita lohkokehityksessä hyödynnetään.

Opinnäytetyöprosessin aikana toteutettiin kaksi sisältölohkoa, keskenään eriävien menetelmien avulla. Ensimmäinen lohko on verkkolehdistä tuttu kainalojuttulohko, joka hyödyntää palvelimen puolella tulostamista, ja tukee muiden lohkojen lisäämistä kainalojutun sisälle. Toinen lohko on palvelimen puolen tulosteella toimiva artikkelinostolohko, joka nostaa artikkeleita ylläpitäjän valitseman määrän tämän niin ikään valitseman kategorian perustella. Lohkoja varten rekisteröitiin lisäksi oma lisäosa, jonka avulla artikkelit voi lisätä mihin tahansa WordPress-asennukseen, sekä lohkokategoria, jonka avulla ylläpitäjä löytää lisäosan luomat lohkot helpommin editorissa.

Työ saavutti sille asetetut tavoitteet, ja molemmat lohkot toimivat niille suunniteltuun tapaan. Työn aikana selvisi, että kehitystyötä voidaan tehdä monipuolisesti eri teknologioiden avulla. Yksinkertaisimmillaan lohko sisältää vain yhden JavaScript-tiedoston, joka rekisteröi lohkon muutaman perustietokentän avulla. Rajoitteena lohkon käyttämille teknologioille toimii oikeastaan ainoastaan lohkon rekisteröinti, joka tulee tehdä JavaScript-tiedoston avulla, tai palvelimen puolella. Tuloste voidaan siis tehdä vapaavalintaisella verkkokehitysteknologialla.

Asiasanat
Sisältölohko, sisältölohkoeditori, sisällönhallintajärjestelmä, client-side rendering, server-side rendering

Sisällys

1 Johdanto

Maailman suurin sisällönhallintajärjestelmä WordPress käy parhaillaan läpi suurinta murrosvaihetta elinkaarensa aikana. Verrattain yksinkertaisesta blogialustasta ponnistanutta sisällönhallintajärjestelmää muutetaan parhaillaan dynaamisia verkkosisältöjä jakavaksi alustaksi. Uusi editori on ollut osana WordPressiä jo vuodesta 2019, mutta vuoden 2022 aikana se korvaa aiemman editorin lopullisesti, ja mullistaa samalla WordPressin toimintalogiikan laaja-alaisesti. Lisäksi uutena asiana WordPressiin on tullut uudet lohkoteemat, jotka nojaavat lähtökohtaisesti nimenomaan lohkoeditorin sisältölohkoihin. WordPressin viralliset Twentytwentytwo- sekä Twentytwentythree-teemat ovat ensimmäiset WordPressin tyhjän asennuksen mukana tulleet lohkoteemat, ja yhteisön tavoitteena on siirtyä nojaamaan pääasiassa lohkoteemoihin.

Monipuolisten sisältöjen ohella WordPressin uusi Gutenberg editori on onnistunut jakamaan mielipiteet. Virallisella lisäosalla on tämän johdannon kirjoittamishetkellä (6.2.2022) 3505 arvostelua, joista reilusti yli puolet (2310 kpl) ovat antaneet lisäosalle vain yhden tähden viidestä mahdollisesta. Keskiarvoisesti arvosanaksi muodostuu vaatimattomat 2,1. Voimakkaasta vastustuksesta huolimatta WordPress.org on julkaissut aikomuksensa muuttaa koko sisällönhallintajärjestelmän uuteen lohkopohjaiseen malliin vuoden 2022 aikana. Samalla aiotaan poistaa tuki klassiselle editorille.

Maailmanlaajuisesti kymmenet tuhannet WordPress -kehittäjät pohtivat tahoillaan tämän tutkimuksen tutkimuskysymyksenä olevia aiheita. Ne ovat: Miten sisältöä kehitetään lohkoeditorille? Miten luon yksinkertaisimman mahdollisen toimivan lohkon (MVP, Minimum viable product)? Miten luon tyypillisiä sisältölohkoja hyödyntäen moderneja verkkokehityskirjastoja? Miten hyödynnän lohkoille tarkoitettuja rajapintoja ja komponentteja?

Tutkimuksessa ei oteta laajemmin kantaa klassisen editorin ja uudemman Gutenberg -editorin eroihin. Tarkoituksena ei ole myöskään tutkia syitä tai oikeutuksia uuden editorin vastustukselle. Syynä taustojen tarkemman tarkastelun rajaamiselle pois on aiheen irrallisuus toiminnallisten lohkojen kehittämisestä. Gutenberg lohkokehitykseen on rakennettu suhteellisen suosittuja ”starttipaketteja”, joista tunnetuimpia ovat create-guten-block sekä wp-create-block. Tämä tutkimus keskittyy lohkojen luomiseen Webpack -konfiguraatiolla eikä huomioi edellä mainittuja aloituspakkauksia. Aloituspakkauksille on paikkansa, mutta on vaikea ennustaa pakkausten pidempiaikainen tuki. Itse ympäristöt luomalla voin rajoittaa itse riskiä riippuvuuksien vanhenemisesta. Lisäksi tutkimuksen lähtökohtana oletetaan, että lukijalla on perusymmärrys WordPress -ympäristön toimintaan ja kehittämiseen, eikä WordPress -ytimen toiminnallisuuksia siten esitellä pintaa syvemmältä muuten kuin aiheeseen suoraan liittyviltä osin. Rajaus on tarpeellinen sillä opinnäytetyön laajuus ei

yksinkertaisesti riitä käsittelemään kaikkia WordPressin laajempaan toimintaan liittyviä tietoja. Tutkimus keskittyy nimenomaan WordPress.org julkaisuihin perustuviin ympäristöihin, ja kaupallinen tuote WordPress.com on rajattu tutkimuksen ulkopuolelle. Rajaus on tehty sillä suurin osa kehittäjistä ja digitoimistoista suomessa käyttävät sivustokehityksessä nimenomaan WordPress.org asennuksia eikä alkuperäisen kehittäjän kaupallista palvelua.

Opinnäytetyön keskeiset käsitteet ovat sisältölohko, sisältölohkoeditori, sisällönhallintajärjestelmä, client-side rendering sekä server-side rendering. Tässä työssä sisältölohkolla tarkoitetaan toistettavaa ohjelmakomponenttia, joka mahdollistaa sisällön näyttämisen eri puolilla sivustoa lohkoeditorin avulla. Lohkoeditori on sisältölohkoihin perustuva sisällönmuokkauseditori, jossa ylläpitäjäkäyttäjä suunnittelee ja lisää sivustolla näytettävää sisältöä valmiiden komponenttien avulla. Sisällönhallintajärjestelmällä tarkoitetaan ohjelmaa, joka tarjoaa ylläpitäjäkäyttäjille erilaisia sisällön luomiseen ja hallitsemiseen keskittyviä graafisia työkaluja, joiden pohjalta tämä muodostaa verkkosivusisältöä, ilman että tämän tarvitsee osata ohjelmoida. Client side rendering:illä tarkoitetaan tilannetta, jossa sivustoa käyttävän lukijan päätelaite laskee sisällön asetteluita tai toiminnallisuuksia, tyypillisesti palvelimen sille lähettämän JavaScript-koodin avulla. Server side rendering puolestaan on sisällönjakomenetelmä, jossa palvelin muodostaa sisällön näyttämiseen valmista HTML- ja CSS-sisältöä. Opinnäytetyön muita lyhenteitä sekä käsitteitä on avattu tarkemmin liitteessä 1.

2 Gutenberg-lohkoeditori

Gutenberg on koodinimi, jolla viitataan WordPressin uuteen lohkoeditoriin. Lohkoeditori on saanut nimensä Johannes Gutenbergin mukaan, sillä editorin tarkoituksena on mullistaa WordPress-sisällönhallintajärjestelmän julkaisukokemus samaan tapaan kuin Gutenberg mullisti kirjojen painamisen. (WordPress.org s.a. a.)

Lohkoeditorissa artikkelin sisältö koostetaan erilaisista sisältölohkoista. Kappaleita, otsikoita, mediaa sekä upotettua mediaa käsitellään editorissa lohkokomponentteina, joiden tiedot tallennetaan tietokantaan. (WordPress.org s.a. a.)

Lohkoeditori lisättiin alun perin lisäosana WordPress-kauppaan, mutta se on versiosta 5.0. alkaen sisällytetty osana WordPressin asennusta. Versiossa 5.0. lohkoeditori määritettiin oletuseditoriksi uusille artikkeleille ja sivuille. (WP Beginner 2022.)

WordPress Classic Editor muutettiin sittemmin lisäosaksi, joka voidaan lisätä WordPress-asennukseen, mikäli sitä halutaan käyttää lohkoeditorin sijasta. Classic Editorin tuki loppuu vuoden 2022 aikana, jolloin lohkoeditorista tulee ainoa virallisesti tuettu editori WordPress-ympäristössä. WordPressin kehittäjäyhteisö suosittaa uuteen editoriin siirtymistä, mutta Classic Editor on edelleen käytössä viidessä miljoonassa aktiivisessa asennuksessa. (Sears 2021.)

2.1 Syyt lohkoeditorin kehitykseen ja käyttöönottoon

Aiempi Classic Editor koetaan yleisesti yksinkertaisempana käyttökokemukseltaan, mutta monimutkaisempien asetteluiden luominen sillä vaati HTML-syntaksin muokkaamista ja siten ymmärrystä HTML-merkintäkielestä. Lohkoeditori tarjoaa modernin ja yksinkertaisemman käyttöliittymän, jossa monipuolisempienkaan asetteluiden luominen ei vaadi ohjelmointitaitoja. Lohkoeditori mahdollistaa monimutkaisten asetteluidenluomisen myös sellaisille ylläpitäjille, joilla ei ole kehittäjätaustaa. (Dessign 2022.)

Toimivuudestaan huolimatta WordPressin aikaisempi niin kutsuttu klassinen editori ei tarjonnut käyttäjilleen täyttä WYSIWYG-kokemusta (What you see is what you get). Erityisesti monimutkaisimpia sivustoja tehneet kehittäjät joutuivat nojaamaan lyhytkoodeihin (shortcode) tai html-koodiin, eikä käyttäjä siten nähnyt visuaalista esikatselua ennen sisällön tallentamista. WordPressin kehitystiimi loi lohkoeditorin vastauksena kolmansien osapuolien kehittämille kehittyneemmille editoreille, kuten esimerkiksi Diville ja Elementorille. (Lefebre, alaluku Extending the block editor.)

Ennen lohkoeditoria ylläpitäjät pystyivät asettamaan ohjelmakoodia lyhytkoodien avulla. Lohkoilla pyritään ratkaisemaan vastaava ongelma kuin lyhytkoodeilla aiemmin. (Kinsta 2022a.)

Lyhytkoodit lisättiin osaksi WordPressiä vuonna 2008 julkaistussa versiossa 2.5. Lyhytkoodien avulla käyttäjät voivat lisätä monimutkaista sisältöä sivustolle ilman että heidän tarvitsee luoda monimutkaisia HTML- tai upotusratkaisuja. Lyhytkoodeja voi hyödyntää Gutenberg-editorissa käyttämällä ”Shortcode” -lohkoa. WordPressin asennuksen mukana tulee 6 lyhytkoodia, mutta näiden lisäksi sovelluskehittäjät voivat luoda omia lyhytkoodejaan, joita ylläpitäjät voivat hyödyntää sivustoillaan liittämällä lyhytkoodin sisältöihin. Lyhytkoodit esitellään osana teeman ”functions.php” -tiedostoa, tai osana lisäosaa. (Kinsta 2021.)

2.2 Sisältölohkot

Sisältölohkot ovat jonkin yksittäisen sisältötyypin tulostamiseen luotuja verkkokomponentteja. Sivuston ylläpitäjät lisäävät sisältölohkojen avulla sisältöä verkkosivulle. Tällaista sisältöä ovat muun muassa tekstikappaleet, kuvat sekä listat. (WordPress.org s.a. b.)

Koko verkkosivun sisältöosuus luodaan lohkoeditorissa sisältölohkoina. Lohkot itsessään voivat olla monen muotoisia, mutta lohkoeditori tarjoaa jokaisen lohkon muokkaamiseen yhtenäisen käyttöliittymän. Kaikille lohkoille on esimerkiksi yhteiset lisäämiseen, siirtämiseen, kopiointiin, vetämiseen, muuttamiseen sekä poistamiseen tarkoitetut toiminnallisuudet. Lohko voidaan myös tallentaa sisältöineen uudelleenkäytettäväksi, jolloin se voidaan esimerkiksi kopioida muihin artikkeleihin. (WordPress 2022d.)

WordPressin mukana tulee versiossa 5.9. kymmeniä erilaisia lohkoja, jotka on tarkoitettu erilaisten sisältöjen näyttämiseksi verkkosivulla. Esimerkiksi kappalelohkon avulla sivulle voidaan lisätä tekstikappale. Kuvalohkon avulla sivustolle voidaan lisätä kuva omalta päätelaitteelta tai vaihtoehtoisesti sivuston mediakirjastosta. Upotuslohkoilla voidaan lisätä muilta verkkosivustoilta tuotavia verkkosisältöjä, esimerkiksi YouTube-lohkolla voidaan upottaa video. (WordPress.org s.a. b.)

Sisältölohkoja voi muokata lohkon asetuksia hyödyntämällä. Lohkolle on voitu ohjelmoida esimerkiksi sisällön värejä, tekstimuotoiluja tai asettelua muuttavia asetusparametreja, joilla sisältöä muokkaava ylläpitäjä pääsee muokkaamaan lohkoa mieleisekseen tai paremmin tarkoitukseen sopivaksi. (WordPress.org s.a. c.)

Lohkon asetuksia muokataan sivun oikeassa laidassa olevassa lohkoasetus -sivupalkissa, sekä lohkon työkalurivillä, joka avautuu aktiivisen muokattavan lohkon yläpuolelle editorissa. Jokaiselle lohkotyypille on rekisteröity omat asetuksensa, ja nämä asetukset ovat lohkokohtaisia. Yhden tekstikappalelohkon asetusten muuttaminen ei siis vaikuta muihin tekstikappalelohkoihin. (WordPress.org s.a. c.)

Osalle lohkoista on lisätty tyylejä, joiden väliltä ylläpitäjä voi valita tarpeisiinsa sopivimman. Lohkon tyylit ovat muotoiluasetuksista koostuvia kokonaisuuksia, joilla muokataan lohkon ulkoasua. Verkkokehittäjät voivat lisätä lohkoille uusia tyylejä sekä poistaa olemassa olevia tyylejä ”register_block_style” sekä” unregister_block_style” funktioiden avulla. (A white pixel 2020.)

Lohkolle voi myös lisätä HTML ankkurin, jota voi hyödyntää hyppylinkkien muodostamiseksi (Ultimate Blocks 2021). Lohkolle voi lisätä myös CSS-lisäluokkia, joilla lohkoon voidaan kohdistaa muotoiluja (Gutenberg Hub s.a. a).

2.3 WordPress Coren sisältölohkot

Gutenberg-editorin mukana tulee useita sisältölohkoja, esimerkiksi seuraavaksi luetellut. Kappalelohko (Paragraph) on tekstieditori, joka mahdollistaa tekstikappaleiden lisäämisen ja muotoilemisen. Esimerkiksi tekstin kokoa, väriä sekä asettelua pystyy muuttamaan kappalelohkon asetuksista. Otsikkolohko (Heading) mahdollistaa sivun jakamisen osioihin. Lohko tukee eri otsikkokokoja H1 – H6 välillä. Kuvalohko (Image) mahdollistaa kuvan lisäämisen omalta koneelta, URL-osoitteella tai tietokannasta. Luettelolohko (List) tukee numeroituja sekä järjestämättömiä luetteloita. Editorin mukana tulee myös klassisen editorin lohko (Classic), joka muistuttaa klassisesta editorista tuttua editoria tekstimuotoilu ominaisuuksineen. (Tuca 13.1.2023.)

2.4 Editorin käyttäminen

Uusi artikkeli lisätään WordPress-hallintapaneelissa, artikkelit-sivulla. Uusi artikkeli lisätään sivun yläreunassa olevan ”lisää uusi” -painikkeen kautta, jolloin WordPress avaa uuden artikkelin muokattavaksi lohkoeditoriin. Ensimmäinen editorissa muokattava kenttä on sivun otsikko, jonka lisäksi editoriin aukeaa oletuksena yksi kappalelohko. (WP Beginner 2023.)

Lohkoeditori on jaettu kolmeen pääosaan – yläpalkkiin, sisältöalueeseen sekä asetukset -sivupalkkiin, jossa näytetään sivun artikkelin tai lohkon asetukset. Yläpalkista löytyy muun muassa lohkojen lisääjä, jonka avulla käyttäjä pystyy lisäämään lohkoja, ja sen avulla voi julkaista artikkelin tai avata artikkelin esikatselun. (WordPress.org s.a. g.)

Kuva 1. Lohkoeditorin jakautuminen sen eri pääosioihin

Kuvassa 1 kuvataan lohkoeditorin jakaantumista sen eri pääosioihin, joihin viitataan tämän opinnäytetyön eri vaiheissa. Valkoisella esitetty alue on sisältöeditori, sininen alue kuvaa lohko- ja sivuasetusten asetuspalkkia, ja vihreä alue on editorin yläpalkki.

Lohkoeditorin asetuspalkissa voidaan muuttaa sivun, artikkelin tai valitun lohkon asetuksia. Sivun ja artikkelin asetukissa voidaan muuttaa artikkelin näkyvyyttä julkisen, yksityisen tai salasanasuojatun välillä. Artikkelin voi myös julkaista, joko ajoitettuna tai heti, tai muuttaa luonnokseksi sivupalkin avulla. Näiden asetusten ohella voidaan muokata muun muassa artikkelin osoitetta, katkelmaa, esittelykuvaa, sivupohjaa tai kategoriaa. (WordPress.org s.a. f.)

Lohkoeditorin yläpalkin vasemmassa reunassa on painikkeet uuden lohkon lisäämiselle, artikkelin yleistietojen tarkastelemiselle sekä lohkonavigaation eli kaikkien asetettujen lohkojen listaukseen. Lisäksi palkista löytyy painikkeet edellisen toiminnon perumiselle ja uudelleen suorittamiselle. Yläpalkin oikeassa reunassa puolestaan on sivun esikatselun avaava painike, sivun julkaiseva painike, sivuasetukset avaava painike sekä sivupalkin näyttävä ja piilottava painike. (Tuca 13.1.2023.)

Uuden sisältölohkon voi lisätä joko yläpalkin painikkeesta, tai minkä tahansa lohkon alapuolella tai oikealla puolella olevasta lohkon lisäävästä painikkeesta. Lohkoja pystyy selaamaan kategorioittain, tai hakutoiminnolla. Lohkovalitsimen voi myös avata aloittamalla uuden kappalelohkon kauttaviivamerkillä, ja alkamalla kirjoittamaan lohkon nimeä. (WP Beginner 2023.)

Lohkon asetuksia pystyy muuttamaan joko lohkon kohdalle aukeavan työkalurivin, tai oikean sivupalkin lohkoasetusosion kautta. Esimerkiksi kappalelohkon työkaluriviltä voi lihavoida tai kursivoida tekstiä, tai asettaa hyperlinkkejä. Lohkon voi myös siirtää ylemmäs tai alemmas sivulla työkalurivin kautta, tai raahaamalla lohkoa (hiiren avulla). (WP Beginner 2023.)

2.5 Lohkoeditorin tulevaisuus ja kehityssuunta

Lohkoeditorin kehitys on jaettu neljään päävaiheeseen. Projektin ensimmäinen vaihe keskittyy helpompaan editointikokemukseen. Toinen vaihe keskittyy muokattavuuteen. Kolmas vaihe keskittyy sisällön muokkaamiseen yhteistyössä. Neljäs, ja viimeinen vaihe keskittyy monikielisyyden tukemiseen. Vuonna 2023 kehitystyö siirtyy projektin kolmanteen vaiheeseen, joka puolestaan on jaettu kolmeen julkaisuversioon (6.2, 6.3 sekä 6.4). (WordPress.org s.a. m.)

2.6 Artikkelin asetukset ja julkaiseminen

Artikkelin (tai sivun) olennaiset asetukset löytyvät editorin oikeasta sivupalkista. Asetusten tarkka sisältö voi vaihdella asennettujen teemojen ja lisäosien mukaan, mutta asetuksissa on tyypillisesti seuraavaksi luetellut osiot. Yhteenveto- osiossa voidaan muuttaa artikkelin julkisuustyyppiä yksityisen, julkisen ja salasanasuojatun välillä. Osiossa voidaan myös esimerkiksi ajastaa artikkelin julkaisu, muuttaa artikkelin osoitetta, vaihtaa sivupohjaa, muuttaa artikkelin kirjoittajaa tai siirtää artikkeli roskakoriin. Kategoriat-osiossa artikkelille voidaan lisätä joko uusia tai aiemmin luotuja kategorioita. Asiasanat-osiossa artikkelille voidaan liittää artikkeliin liittyviä olemassa olevia tai uusia asiasanoja. Artikkelikuva-osiossa voidaan liittää artikkelin yhteydessä käytettävää kuvaa, joka voidaan joko ladata uutena kuvatiedostona tai valita tietokannasta. Ote-osiossa artikkelille voidaan antaa lyhyt kuvausteksti. Kommentointi-osiossa voidaan hallita artikkelin yhteydessä mahdollisesti näytettävän kommenttiosion asetuksia. (WordPress.org s.a. i.)

2.7 Lohkoteemat

Lohkoteemojen myötä sivuston ylläpitäjälle annetaan oletuksena enemmän asetuksia hallintaansa kuin aiemmin on ollut tyypillisesti tapana. Aiemmin ylläpitäjän muokattavissa oli ne asetukset, jotka kehittäjä oli kytkenyt muokattavaksi. Lohkoteeman aktivoinnin myötä ylläpitäjä pystyy muuttamaan teeman sivupohjia sekä tyylejä uuden sivustoeditorin kautta. WordPressin perinteinen customizer-työkaluvalikko, sekä vimpaimien ja valikoiden hallintanäkymät poistuvat käytöstä. Vimpaimet poistuvat kokonaan käytöstä, kun taas navigaatiot on muutettu navigaatiolohkoiksi. (Tadlock 2022.)

3 Lohkokehityksen teknologiat

WordPress tallentaa avoimeen lähdekoodiin perustuvaan MySQL-tietokantaan käyttäjien ja ylläpitäjien tallentamia tietoja. Tietokantaan tallennetaan muun muassa artikkelien ja sivujen sisällöt, käyttäjätiedot sekä erilaiset sivuston, teemojen ja lisäosien asetukset. Tietokanta on jaettu tietokantatauluihin tiedon tyypin mukaan suorituskyvyn ja tiedon löytämisen helpottamiseksi. (Kinsta 2022b.)

WordPress tarvitsee toimiakseen verkkoserverin (Web server) joka pystyy prosessoimaan PHP-tiedostoja sekä ylläpitämään tietokantaa kuten MySQL tai MariaDB. Yleisimmät verkkoserverit WordPressin käyttämiseksi ovat NGINX sekä Apache, jotka toimivat keskenään eri tavoilla, mutta ovat molemmat WordPressin tukemia. (FullHost 2022.)

WordPressissä on kahdenlaisia hook-funktioita, action:eita sekä filter:eitä. Hook:it mahdollistavat ohjelmakoodin muuttamisen tai sen kanssa vuorovaikuttamisen ennalta määritetyissä kohdissa WordPress Coren suorituksessa. Action hook -funktiot mahdollistavat datan lisäämisen ja WordPressin toiminnan muuttamisen. Action hook -funktion kutsumaan funktioon voidaan laittaa esimerkiksi tulostuksia tai tietokantaan kirjoittamista. Filter hook -funktiot puolestaan mahdollistavat datan muokkaamisen suorituksen eri vaiheissa. Niiden kutsumat funktiot muuttavat niille välitettyä tietoa ja palauttavat muunnetun tiedon niitä kutsuville funktioille. (WordPress.org s.a. h.)

Sivupohjien (template) avulla voidaan muuttaa sivun sisällön esitystapaa. Ne voidaan kiinnittää yksittäiseen sivuun, sivuosioon tai sivutyyppiin. Sivupohja voidaan joko liittää ohjelmallisesti tiedoston nimen avulla, tai tarjoamalla sivupohja sivuja muokkaavien ylläpitäjien erikseen päälle kytkettäväksi vaihtoehdoksi. Sivupohjien avulla voidaan esimerkiksi luoda erilainen pohja etusivulle, tai näyttää artikkelistaus tai artikkelikuva sisällön ohessa. (WordPress.org s.a. j.)

3.1 Lohkokehityksen tyypilliset teknologiat

3.1.1 React

React on Facebookin kehittämä JavaScript-ohjelmistokirjasto, jota käytetään käyttöliittymäkomponenttien luomiseen. Se julkaistiin avoimena lähdekoodina vuonna 2013, ja on sittemmin noussut kaikkein laajimmin käytetyksi ohjelmistokirjastoksi verkkokehittäjien keskuudessa. (Noble Desktop 2022.)

Elementin tilaa muuttaessa React muuttaa selaimessa ainoastaan kyseistä elementtiä, esimerkiksi muuttaessa taulukossa olevaa solua, React päivittää lähtökohtaisesti vain muuttunutta solua. React luo DOM-puusta kevennetyn version, Virtual DOM:in, jossa esitetään muutoksen haluttu

tulos. Vertailemalla selaimen DOM-puuta sekä muutoksen Virtual DOM-puuta, React selvittää miten muutos voidaan toteuttaa mahdollisimman vähillä DOM-operaatiolla, joita pidetään yleisesti hitaina pullonkauloina laajemmissa JavaScript sovelluksissa. (Stoyan 2015, 55–56.)

Komponenttien käyttäytymismalli sekä esittämistapa voivat muuttua riippuen komponenteille annetuista ominaisuuksista (properties). Lohkon ominaisuuksia voidaan käyttää sen ”this.props” -objektin kautta. Komponentin ei tulisi muuttaa ominaisuuksiaan, vaan sitä tulisi käyttää ulkoisten komponenttien antamien ominaisuuksien tuomiseen. (Stoyan 2015, 13.)

Reactin etu verrattuna vanhempien kirjastojen DOM-manipulaatioon perustuu elementtien tilan eli staten hallintaan. State on dataa, jota komponentti käyttää itsensä piirtämiseksi. Kun State-dataa muutetaan, React piirtää (render) komponentin uudelleen ilman erillistä käskyä. Kehittäjän ei siten tarvitse miettiä niinkään komponenttien muuttamista käyttöliittymän puolella, vaan tämä voi keskittyä datan päivittämiseen. Statea voi muokata komponentin ”this.state” -objektin kautta. (Stoyan 2015, 18.)

Komponentin ominaisuuksia ja tilaa käytetään hieman eri näkökulmista. Ominaisuuksia tulisi muuttaa lähinnä ulkoisista lähteistä, eli komponentin käyttäjä muuttaa propsien avulla komponentin esittämistapaa. Komponentin tilaa taas muutetaan komponentin sisällä. (Stoyan 2015, 24.)

3.1.2 JSX

JSX on JavaScript syntax extension, jota käytetään React-komponenttien tekemiseen. Se muistuttaa syntaksiltaan XML-syntaksia, ja sisältää yleensä JavaScript-koodia. JSX-syntaksi on muutettava puhtaaksi JavaScriptiksi selainta varten. JSX-syntaksin käyttö ei ole pakollista, mutta on suositeltavaa erityisesti isompien komponenttien luomisen helpottamiseksi. (Corvalan, Sengupta & Singhal 2016, 13–15.)

Tyypillisesti HTML ja JavaScript on eritelty viitekehyksissä (Framework). MVC-mallia (Model View Controller) hyödyntävissä sovelluksissa malli, näkymä ja käsittelijä on eriytetty toistaan selkeämmän luettavuuden ja ylläpidettävyyden vuoksi. JSX-syntaksilla kirjoitettu React ohjaa kehittäjiä erilaiseen toimintamalliin, jossa komponenttiin liittyvät asiat esitellään yhdessä paikassa, ja niihin tarvittaessa viitataan muualta. JSX avulla on kolmansien osapuolien komponenttien käyttäminen sekä omien komponenttien julkaiseminen on yksinkertaista. (Corvalan ym. 2016, 16–17.)

3.1.3 JSX syntaksin muuttaminen JavaScriptiksi

JSX-syntaksin muuttaminen JavaScriptiksi vaatii huomattavaa laskentaa asiakkaan puolella (Client Side Render), jonka vuoksi sitä ei kannata tehdä tuotantoversioissa asiakkaan puolella. Sen sijaan meidän tulisi tehdä muunnos kehittäessä kääntäjän (compiler) avulla. Tarkoitukseen voi käyttää esimerkiksi Babel:ia, Webpack:ia, Grunt:ia tai Gulp:ia. (Corvalan ym. 2016, 18.)

Webpack on moduuliyhdistin (module bundler) jota käytetään JavaScript-sovellusten käyttöönottamiseen. Webpack ottaa syötteenä moduuleja, joilla voi olla riippuvuuksia, ja muodostaa niiden pohjalta staattisia resursseja (static assets). (Corvalan ym. 2016, 157.)

Webpackille määritetään syötteenä sovelluksen alkupiste (entry point), sekä määränpää ja tiedostomuoto, johon sen halutaan tulostavan käsitelty ohjelmakoodi. Koodin käsittelemiseen Webpack hyödyntää erilaisia lataajia (loader). (Corvalan ym. 2016, 162.)

Webpackin etuna verrattuna osaan kilpailevista moduulinmuuntimista voidaan pitää saatavilla olevia lisäosia sekä erilaisia lataajia (Corvalan ym. 2016, 167.).

3.1.4 Ohjelmistopaketit ja paketinhallinta

NPM (Node Package Manager) on laajalti käytetty paketinhallinta-alusta, jonka kautta kehittäjät voivat jakaa sekä käyttää avointa lähdekoodia paketteina. Paketit koostuvat JavaScript-moduuleista tai kirjastoista. Jokainen NPM projekti sisältää package.json -tiedoston, johon projektiin kuuluvien pakettien versionumerot ja muu projektiin liittyvä metatieto tallennetaan. NPM sisältää komentorivikäyttöliittymän, jonka kautta kehittäjät voivat ladata ja poistaa projektin paketteja, sekä muuttaa ladattujen pakettien versioita. (Hostinger 2022.)

WordPress tarjoaa kehittäjille useita JavaScript-paketteja sekä -työkaluja. Niitä voidaan käyttää joko kutsumalla globaalin ”wp”-muuttujan kautta, tai lataamalla ne Node Package Managerin avulla. Käyttäessä globaalia muuttujaa tulee tarvittava kirjasto ensin lisätä sivulle lisäämällä paketti koodikatkelman jonoon rekisteröimisen yhteydessä riippuvuutena (dependency). NPM kautta käytettäessä paketti tulee ensin ladata ympäristöön käytettäväksi, ja sitä voidaan sen jälkeen käyttää lataamalla komponentit JavaScriptin import-toiminnon avulla. (WordPress.org s.a. m.)

3.2 WordPress lohkojen asetukset ja rajapinnat

Hyvä keino lohkojen käytettävyyden parantamiseen on jatkaa lohkojen asetuksia. Sivupalkin lohkoasetuksia voidaan jatkaa hyödyntämällä WordPressin editor-kirjastosta löytyvää InspectorControls-komponenttia. Komponentin sisälle lisättävä sisältö näytetään lohkoa muokatessa sivupalkissa, ja

on tarkoitettu koko lohkoa koskevien asetusten muokkaamiseen – rivillä muokkaamiseen tarkoitetut asetukset tulisi sijoittaa työkalupakkiin. Yhtenäisen ulkoasun saavuttamiseksi asetusten jäsentelemiseen sivupalkissa voi hyödyntää WordPressin components-kirjastosta löytyviä PanelBody- sekä PanelRow-komponentteja. (JavaScript for WP 2019.)

Lohkoille voidaan luoda ennalta editorissa valittavia lohkotyylejä, joilla lohkoille voidaan lisätä yleensä yksinkertaisia tyylivariaatioita. Nykyisellään lohkotyylejä voidaan valita yksi kerrallaan. Lohkotyylin voi valita graafisesta ”styles” asetuspaneelista lohkon asetuksissa, mutta käytännössä asetus liittää lohkolle ylimääräisen CSS-luokan. Itse muotoilu lisätään CSS-koodina joko osana lohkon rekisteröintiä (inline CSS) tai osoittamalla tyylin rekisteröinnissä lähde ulkoiselle tyylitiedostolle, joka voi olla esimerkiksi CSS- tai JS-tiedosto. Lohkotyylejä voi rekisteröidä joko backendin puolella PHP koodina, tai frontendissä JavaScriptillä. (Nymark 19.2.2022.)

Lohkotukien rajapinta (Block Supports API) mahdollistaa tiettyjen yhtenäisten asetusten määrittämisen lohkoille. Tällaisia tukia ovat esimerkiksi asettelu (align), väri (color), typografia (typography) sekä skaalaus (scaling). Lohkotukien kytkeminen lisää lohkon attribuutteihin tukia vastaavat asetukset, sekä luo niiden muokkaamiseen käyttöliittymät. Jotta lohkotuet toimisivat, tulee lohkon ympäröivään elementtiin editorissa lisätä useBlockProps-funktio. Editorin tallenteessa puolestaan tulee käyttää useBlockProps-funktion alta löytyvää save-funktiota. (WordPress.org s.a. l.)

Sisältölohkot on jaettu editorissa kategorioittain. Oletuksena WordPressin mukana tulee teksti-, media-, suunnittelu-, widget-, teema- sekä upotuskategoriat, mutta näiden lisäksi kehittäjät voivat luoda omia kategorioitaan. Kategorian rekisteröinnillä lohko saadaan erottumaan lohkovalikoimasta paremmin, ja sillä voidaan kontekstoida lohkojen rekisteröityneen esimerkiksi jonkin tietyn lisäosan tai teeman mukana. Toisaalta lohkot voi kategorisoida jonkin toiminnallisuuden perusteella. Kategoriat lisätään ’block_categories_all’ filtterin avulla, muuntamalla filtterille annettavaa ja sen palauttamaa kategorialistaa. Kategoria tarvitsee parametreikseen ainakin uniikin viittauksen (slug) sekä otsikon. (Gutenberg Hub s.a. b.)

3.3 Lohkojen rekisteröiminen

Lohkojen rekisteröimiseen asiakkaan puolella käytetään registerBlockType-funktiota. Funktio tulee lohkoeditoriin osana wp.blocks kirjastoa, joka sisältää lohkojen kehittämisen kannalta hyödyllisiä funktioita sekä WordPressin mukana tulevat lohkot (Gordon 2017). Rekisteröitävälle lohkolle on annettava nimi muodossa ”nimiavaruus/lohkonnimi”, jossa nimiavaruutena toimii lisäosan tai teeman nimi. Lohkolle annetaan myös otsikko, joka näytetään lohkojen lisääjän yhteydessä. Lohkolle määritetään kategoria, jotta käyttäjien olisi helpompi löytää lohko. Vapaaehtoisesti lohkolle voidaan

määrittää lisäksi kuvaus, avainsanoja, kuvake, lohkotyylejä, attribuutteja, esimerkkidataa, variaatio, tuetut ominaisuudet, muutokset sekä vanhempana toimiva lohko. (WordPress.org s.a. e.)

Asiakkaan puolella rekisteröinnin ohella lohko on rekisteröitävä myös palvelimen puolella. Rekisteröinti tapahtuu register_block_type-funktiolla, jonka ensimmäiseksi argumentiksi annetaan sama nimi kuin frontendin rekisteröinnissä. Yksinkertaisemmillaan tämä riittää, jos lohkon editorin tuloste sekä sivustolla näytettävä tuloste on molemmat tallennettuna lohkon rekisteröinnin yhteydessä JavaScript tiedostossa. Jos lohkolla on attribuutteja, tai lohkon tuloste halutaan toteuttaa palvelimen puolella PHP-koodilla, tulee palvelimen puolella olevalle rekisteröintifunktiolle antaa lisäargumentteja toisena parametrina välitettävän listan muodossa. Attribuutit annetaan listana, joka sisältää attribuutin nimen sekä tyypin. Palvelimen puolella tehtävän tulosteen palauttavan funktion nimi puolestaan annetaan merkkijonona avaimella ”render_callback”. (WordPress VIP 2023.)

3.4 Lohkoille rekisteröitäviä lisäominaisuuksia

Lohkon tunnistamisen helpottamiseksi lohkoille voidaan antaa rekisteröinnin yhteydessä kuvake. Kuvakkeen voi lisätä joko WordPress Dashicons -kirjastosta, tai vaihtoehtoisesti mukautettuna SVG-elementtinä. Ikonille voi antaa värin sekä taustavärin. Lohkon ikoni väreineen näytetään muun muassa lohkoeditorin lohkojen valitsimessa. (WordPress.org s.a. e.)

Lohkoista voidaan muodostaa lohkotemplaatteja, joiden avulla lista lohkoja asetuksineen voidaan syöttää esitäytettyinä. Lohkolle voidaan esitäyttää sen attribuutit tai mallisisältöä. Lohkotemplaatin voi esittää lohkolle joko backendin puolella post_type_object-objektia laajentamalla, tai frontendissä lohkon rekisteröimisen yhteydessä. Lohkotemplaatin voi asettaa myös mukautettuun artikkelityyppiin sen rekisteröinnin yhteydessä. Lohkotemplaatin voi myös lukita lohkoittain tai kokonaisuudessaan, mikäli sen muokkaaminen editorissa halutaan estää. (WordPress.org s.a. k.)

Lohkoille voidaan lisätä WordPressin InnerBlocks-komponentin avulla muokkausalueita, joissa lohkon sisään voi liittää muita lohkoja. Lohkon sisällä saatavilla olevia lohkotyyppejä voi lisäksi rajoittaa InnerBlocks-komponentin allowedBlocks-parametrin avulla, syöttämällä parametriksi lista sallituista lohkoista. (Watson 2021)

3.5 Lisäosan rekisteröiminen

WordPress lisäosan luominen aloitetaan luomalla tiedostoja WordPressin plugins-kansioon (lisäosat). Lisäosan voi luoda joko suoraan plugins-kansioon, tai tyypillisemmin lisäosakohtaiseen alakansioon. WordPress etsii kansiosta PHP-tiedostoja, ja se tunnistaa lisäosan tiedoston alkuun

liitettävästä kommentista. WordPress lukee kommenteista lisäosan tyyppitietoja, kuten nimen, kuvauksen, lisenssin sekä tekijän. (Lefebre 2022, alaluku Creating a plugin and a header.)

4 WordPress Gutenberg -sisältölohkojen kehittäminen

4.1 Projektisuunnitelma

Toteutan tutkimuksen tuotoksena, eli kehitän itse komponentteja, ja teen yhteenvetoa kehityksen ja sen lopputuloksen perusteella. Toteutan tutkimuksen toiminnallisesti, eli kehitän itse komponentteja, ja teen yhteenvetoa kehityksen ja sen lopputuloksen perusteella. Oletuksena on, että tutkimuksessa saadaan luotua toimivia ja verkkokehityksessä hyödyllisiä komponentteja.

Ensimmäisenä aion perustaa kehitysympäristön tutkimusta varten. Kehitysympäristö kattaa käytännössä projektin ohjelmointiympäristösovelluksessa (Microsoft Visual Studio Code), joka sisältää kokonaisen WordPress-projektin. Ohjelmointiympäristöön luon tämän lisäksi konfiguraatiotiedostot Node Package Managerille, jonka avulla hyödynnettäviä kirjastoja hallinnoidaan, sekä Webpackille, jolla muunnan kehittämistäni sisältölohkoista JavaScript paketteja.

Seuraavaksi varmistan asennuksen toimivuuden käynnistämällä projektin lokaalissa kehitysympäristössä MAMP-ohjelman tarjoaman tietokannan ja palvelimen päällä. Perustan WordPress ympäristön ja luon sinne jonkin verran testidataa, jotta tutkimuksessa kehitettäville lohkoille löytyy ympäristöstä haettavaa dataa.

Seuraavaksi toteutan erilaisten lohkojen kehitystutkimuksen, jossa selvitän, miten lohkot voidaan kehittää Gutenberg-editorille, ja mitä erityishuomioita tulisi ottaa huomioon niin tehdessä. Käytän lohkojen kehittämiseen React-ohjelmakirjastoa, ja sen kirjoittamiseen JSX-syntaksia.

Kehitän kaksi lohkoa, joista ensimmäinen on asiakaan puolen renderöintiä hyödyntävä, ja jälkimmäinen puolestaan renderöidään palvelimen puolella. Molemmat lohkot hyödyntävät lohkoasetuksia WYSIWYG-ajattelua mukaillen, jolloin asetukset on sijoitettu omaan sivupalkkiosioonsa, ja editorin tulostealueella näytetään reaaliaikainen esikatselu sivun lopullisesta tulosteesta.

Lopuksi esittelen opinnäytetyön aikana toteutuneet toiminnalliset tuotokset, ja teen pohdinnan tutkimuskysymyksiin nojaten.

4.1.1 Kainalojuttulohko

Lehdissä yleinen elementti kainalojuttu on jalkautunut verkkolehtiin ja verkkosivuille. Kainalojuttu on artikkeliin liittyvä, mutta sen etenemisestä erillinen kontekstoiva artikkeliosa. Haluan että lohkon avulla ylläpitäjä pystyy lisäämään artikkeliin osion, jolla on muusta artikkelista poikkeava tyylittely. Lohkon sisälle tulee pystyä lisäämään muuta sisältöä lohkoina. Osiossa käyn läpi lohkon rekisteröimisen, sisäkkäiset lohkot, lohkokategoriat sekä lohkoasetukset.

Kuva 2. Kainalojuttulohkon rautalankasuunnitelmat

Kuvassa 2 esitetään rautalankasuunnitelma kainalojuttulohkon tulosteille ja asetuksille editorissa sekä tulosteelle julkaistulla sivulla. Tavoitteena on, että editorin sisältöeditorissa oleva tuloste vastaa mahdollisimman tarkkaan lukijoille näkyvää julkaistua sivua. Lohkon asetukset on siksi suurelta osalta lisätty lohkoasetuspalkkiin eikä editorin sisältöalueelle.

4.1.2 Artikkelinostolohko

Kehitän seuraavaksi artikkeleita kategorian perusteella esittävän dynaamisen sisältölohkon. Osiossa käsittelen lohkon attribuutit, palvelimen puolella renderöimisen sekä WP_Query-rajapinnan.

Kuva 3. Artikkelinostolohkon rautalankasuunnitelmat

Kuva 3 esittää artikkelinostolohkon editoria ja lohkon tulostetta julkaistulla sivulla. Lohkon sisältö pohjautuu sivustolla oleviin artikkeleihin, mutta lohkon asetuksia voi muuttaa sivupalkkiasetuksissa. Itse lohkon tuloste editorissa vastaa julkaistulla sivulla näytettävää tulostetta, jotta ylläpitäjä saa jo muokkausvaiheessa realistisen käsityksen sivun ulkoasusta. Jotta ylläpitäjä saa välittömän palautteen tekemistään asetusmuutoksista, tulisi tulosteen päivittyä vastaamaan ylläpitäjän asettamia muutoksia reaktiivisesti.

4.2 Toteutus

4.2.1 Kehitysympäristön perustaminen

Aloitan lohkokehityksen toteutuksen perustamalla kehitysympäristön. Tarkoituksenani on luoda toiminnallisia lohkoja, joiden ulkoasu noudattaa kulloisenkin aktiivisen teeman määrittelemiä ulkoasusääntöjä. Ulkoasun ollessa tausta-alalla, päädyin käyttämään WordPressin mukana automaattisesti asennettavaa Twentytwentytwo-teemaa.

Kehitysympäristön on oltava ratkaisevilta ominaisuuksiltaan vastaava, kuin kehitettävien lohkojen tarkoitettu kohdeympäristö. Mikäli esimerkiksi käytössä oleva WordPress-teema vaikuttaa ratkaisevasti lohkon ulkoasuun tai käyttäytymiseen, kannattaa kyseinen teema ottaa käyttöön myös kehitysympäristössä, jotta kehitettävien lohkojen oikea toiminta ja ulkoasu voidaan todentaa. Jos kehityksen yhteydessä taas ei ole tarvetta testata esimerkiksi palvelimeen liittyviä ominaisuuksia, kehitysympäristö voi poiketa tuotantoympäristöstä siltä osin. Tyypillisesti esimerkiksi verkkosivujen kehityksessä kehitysympäristö asetetaan toimimaan lokaalilla palvelimella tai kehittäjäpalvelimilla, eikä välttämättä samanlaisessa palvelinympäristössä kuin tuotantoversio.

Uuden WordPress ympäristön asentaminen aloitetaan hakemalla WordPress.org -sivustolta asennuspaketti. Tutkimuksen kirjoitushetkellä se löytyy osoitteesta ”www.wordpress.org/download/”, ja se ladataan omalle tietokoneelle ”Download WordPress 6.1.1” -painikkeella. Asennuspaketti ladataan pakattuna kansiona, jonka voi purkaa tietokoneelle vapaasti haluamaansa paikkaan, ellei valittu palvelinympäristö erikseen vaadi tiettyä sijoituspaikkaa. Puran pakatun kansion sopivaan paikkaan tietokoneeni kovalevyllä.

Käytän tutkimuksen kehitysympäristönä MAMP-ohjelman tarjoamaa MySQL-tietokantaa sekä palvelinta. Lohkojen kehitykseen voidaan käyttää useita erilaisia palvelun isännöintiratkaisuja, mutta valitsin opinnäytetyöprojektiin MAMP:in yksinkertaisen käyttöönoton perusteella. Konfiguroin ensimmäiseksi MAMP:in perusasetukset, valitsemalla asetuksista palvelintyypiksi Apachen, ja juurikansioksi kansion, jonne aiemmin purin WordPress aloituspaketin. Loin lisäksi uuden MySQL-

tietokannan sovelluksen mukana tulevan phpMyAdmin-sovelluksen avulla. Asetin tietokannan nimeksi ”kehitys”, ja kollaatioksi oletusasetuksen ”utf8_general_ci.”

Jotta WordPress saadaan käynnistettyä, tulee WordPress-asennuspakkauksessa tulleeseen WordPress-kansioon luoda ”wp-config.php” -tiedosto. Asennuspaketin mukana tulee tätä silmällä pitäen wp-config-sample.php -tiedosto, jota voidaan käyttää pohjana konfigurointitiedostolle. Jotta asennus saa muodostettua tietokantayhteyden, tulee tietokannan asetukset asettaa kyseisessä tiedostossa osoittamaan valittuun kehitysympäristöön. Tein kopion tiedostosta, ja asetin kopion nimeksi ”wp-config.php”. Konfiguraation asetukset voi muokata vapaasti valitsemassaan tekstieditorissa, itse käytän Microsoftin Visual Studio Code -sovellusta. Osoitan tietokantataulun nimeksi ”kehitys”, ja käyttäjätunnukseksi sekä salasanaksi MAMP-sovelluksen oletusarvot (root), jolloin yhteys otetaan aikaisemmassa kappaleessa perustamaani MySQL-tietokantaan. Konfiguraatiotiedostoon tulee vielä lisätä suolat, jotka generoin tiedoston mukana tulleen linkin kautta. Kuvassa 4 on esitetty tietokanta-asetukset, jotka tiedostoon on asetettava MAMP-sovellukseen yhdistämiseksi.

```
21  // ** Database settings - You can get this info from your web host ** //
22  /** The name of the database for WordPress */
23  define( 'DB_NAME', 'kehitys' );
24
25  /** Database username */
26  define( 'DB_USER', 'root' );
27
28  /** Database password */
29  define( 'DB_PASSWORD', 'root' );
30
31  /** Database hostname */
32  define( 'DB_HOST', 'localhost' );
33
34  /** Database charset to use in creating database tables. */
35  define( 'DB_CHARSET', 'utf8' );
36
37  /** The database collate type. Don't change this if in doubt. */
38  define( 'DB_COLLATE', '' );
39
```

Kuva 4. Asennuksen tietokantaan yhdistävät asetukset wp-config.php -tiedostossa

Valinnaisena asetuksena konfiguraatioon voi asettaa päälle virhelokin, joka auttaa erilaisten virheiden selvittämisessä. Lokin saa päälle määrittämällä muuttumattomien WP_DEBUG- sekä WP_DEBUG_LOG-parametrien arvon todeksi. Asetan virhelokin päälle mahdollisten virhetilanteiden selvittämisen helpottamiseksi. Asetusten ollessa nyt määritettyinä, käynnistän paikallisen palvelimen

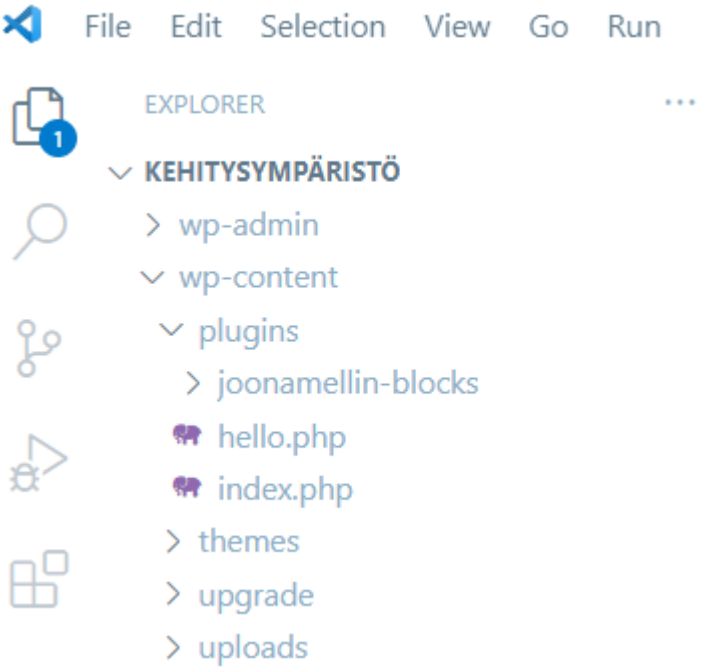
MAMP-sovelluksen etusivulta. Palvelimen käynnistettyä siirryn selaimella osoitteeseen ”http://localhost”, jossa WordPress asennusta isännöidään.

WordPressin ensimmäisen käynnistyksen yhteydessä aukeaa ympäristön asennustyökalu, jossa valitaan asennuksen kieli ja nimi, sekä ylläpitäjäkäyttäjän nimi, sähköpostiosoite sekä salasana. Tietojen tallentamisen jälkeen ympäristön kirjautumissivu aukeaa, ja WordPress-asennus on valmiina käytettäväksi.

4.2.2 Lisäosan rekisteröiminen

Lohkoja voidaan rekisteröidä WordPress-ympäristöön teeman tai lisäosan avulla. Tarkoituksenani on luoda yleisesti hyödyllisiä toiminnallisia lohkoja, joita voidaan hyödyntää useissa projekteissa, joten rekisteröin lohkot lisäosan avulla. Tällöin lohkot voidaan ottaa helposti käyttöön eri asennuksiin lisäämällä luomani lisäosa WordPressin lisäosan asennustyökalulla.

Aloitan lisäosan tekemisen siirtymällä asennuksen wp-content-kansiossa olevaan plugins-kansioon, jossa lisäosat sijaitsevat. Lisään lisäosalle oman kansion, jonka nimen tulee olla uniikki, jotta useita lisäosia sisältävissä WordPress-asennuksissa ei muodostuisi konfliktitilanteita eri lisäosien välille. Lisäosan olisi hyvä olla tunnistettavissa kansion nimen perusteella, jotta sovelluskehittäjien olisi helpompi selata eri lisäosien tiedostoja. Nimesin oman lisäosani oman nimeni sekä lisäosan käyttötarkoituksen perusteella, jotta kansion nimi olisi sekä uniikki että kuvaava. Kuvassa 5 on esitetty lisäosan sijainti osana WordPress-asennuksen kansiorakennetta.



Kuva 5. Luomani lisäosa WordPress-asennuksen kansiorakenteessa

Luon kansion juureen PHP-tiedoston, jonka nimeksi asetan lisäosan nimen mukaisesti joonamellin-blocks. Jotta WordPress tunnistaa tiedoston lisäosan koodin suorittamisen aloittavaksi tiedostoksi, tulee tiedosto aloittaa kommenttiosiolla. Kommenttiosioon tulee vähintään asettaa lisäosan nimi, mutta hyvä käytäntö on asettaa myös lisäosan tekijän nimi sekä lisäosan kuvaus ja versio. Lisäosalle voi rekisteröidä muitakin tietoja, mutta nämä riittävät hyvin omaan käyttöön tarkoitetulle lisäosalle. Lisäosan käynnistävän tiedoston nimi on vapaaehtoinen, mutta sen kannattaa selkeyden nimissä olla viittaus lisäosan nimeen. Kuva 6 esittää lisäosalle rekisteröimiäni perustietoja.

Kuva 6. Lisäosan perustiedot rekisteröivä kommentti

Jotta lisäosan koodi ajettaisiin, se tulee vielä kytkeä päälle WordPressin hallintapaneelissa. Lisäosat löytyvät oman otsikkonsa alta hallintapaneelin valikosta, ja ne kytketään päälle painamalla lisäosalistauksessa halutun lisäosan kohdalta ”aktivoi” -painiketta.

4.2.3 Ohjelmistopakettien lisääminen ja moduuliyhdistimen käyttöönotto

Tulen käyttämään lohkokehitykseen muutamia ulkoisia kirjastoja ja ohjelmistopaketteja, jotka esittelen sitä mukaa kun otan ne käyttöön. Kirjastojen hallintaan käytän Node-paketinhallintaa (Node Package Manager, NPM), joka on ladattu tietokoneelleni globaalisti. Muutan lisäosan kansion Node projektiksi komentokehotteen kautta, kirjoittamalla lisäosan juuressa komennon ”npm init”, joka alustaa projektin pyytämiensä lisätietojen perusteella. Käytän itse asennusohjelman oletustietoja, ja hyväksyn paketin alustuksen, jolloin asennusohjelma lisää lisäosan juureen Node-pakettien hallintaan käytettävän ”package.json”-tiedoston. Node-paketinhallinta käyttää tätä tiedostoa ladattujen pakettien versiotietojen hallitsemiseen sekä mahdollisten prosessien käynnistämiseen.

WordPress-lohkoeditori, sekä sen kehittämiseen liittyvät kirjastot ja ohjelmistopaketit voidaan asentaa Node-paketinhallinnan avulla. Tämä ei kuitenkaan ole välttämätöntä, sillä tarvittavat JavaScript-komponentit ladataan WordPress-editorin mukana selaimeen globaaleiksi muuttujiksi. Komponentin voi siksi ladata joko kirjaston kautta, tai koodissa voi viitata komponentin löytyvän

esimerkiksi window.wp.blocks -paketin alta, jolloin Webpack asettaa viittauksen ikkunamuuttujaan sen sijaan että kasaisi tarvittavat komponentit Node-kirjaston tiedostoista. Tämä vaikuttaisi ainakin nopeuttavan tiedostojen käsittelyä, ja toisaalta tarkoittaa sitä, että komponentista käytetään aina ajantasaista, kulloisenkin WordPress-version mukana tulevaa ja yhteensopivaa versiota. Lataan Node-kirjastot projektiin komentokehotteen avulla komennolla ”npm install @wordpress/block-editor @wordpress/blocks @wordpress/components @wordpress/data @wordpress/dom-ready @wordpress/editor @wordpress/element”. Aion käyttää komponenteissa globaaleja muuttujia, mutta kirjastojen ollessa ladattuina näen koodieditorissa vihjeet eri funktioiden käyttötarkoituksista.

Haluan kehittää lohkoja React:illa, käyttäen JSX-syntaksia, joka pitää muuttaa natiiviksi JavaScript-koodiksi ennen kuin sitä voidaan käyttää selaimessa. Tiedostojen muuntamiseen voidaan käyttää useita eri työkaluja, mutta käytän itse suosittua Webpack-moduuliyhdistintä aiemman tottumuksen vuoksi. Asennan sen projektiin npm-komennolla ”npm install webpack webpack-cli” komentokehotteessa. Tämän lisäksi Node-projektin juuressa tulee olla webpackin konfiguraatiotiedosto ”webpack.config.js”, jonka lisään projektin juureen uutena tyhjänä tiedostona. Konfiguraatiotiedostoon syötetään tieto siitä, mistä tiedostoja haetaan käsiteltäväksi (entry), ja mihin ne tulee tallentaa muutoksen myötä (output). Itse konfigurointiin käytetään module.exports -muuttujaa, jonka arvoksi asetetaan parametrisointitiedot sisältävä objekti. Lisään tiedoston alkuun vakion ”path”, joka osoittaa kansion projektin tiedostojen juuren. Lisään sen lisäksi module.exports -objektin, ja alustan sille tiedostojen sisääntulon sekä ulostulon. En ole vielä luonut lohkoja, mutta tiedän millä nimellä aion ne luoda, joten lisään sisääntuloiksi lohkon nimet, sekä lähteeksi niiden sijainniksi projektin juureen luomani ”block” kansion. Ulostuloksi määritän, että juuren alle tulee luoda kansio ”dist”, jonne sisääntulossa nimetyt JavaScript-tiedostot luodaan. Resolve-parametriksi annetaan lista tiedostomuotoja, joita webpackin toivotaan käsittelevän, tässä tapauksessa ainakin JSX. Kuvassa 7 on esitetty Webpackille rekisteröimäni asetukset kokonaisuudessaan.

wp-content > plugins > joonamellin-blocks > webpack.config.js > ...

```
const path = require('path');
const { ESBuildMinifyPlugin, default: esbuildLoader } = require('esbuild-loader');

module.exports = {
    entry: {
        'kainalojuttu' : path.resolve(__dirname) + '/block/Kainalojuttu.jsx',
        'ViimeisimmatArtikkelit' : path.resolve(__dirname) + '/block/ViimeisimmatArtikkelit.jsx'
    },
    externals: {
        'lodash' : 'lodash'
    },
    mode: 'production',
    module: {
        rules: [{
            exclude: /node_modules/,
            loader: 'esbuild-loader',
            options: {
                loader: 'tsx',
                target: 'es2015'
            }
        }]
    },
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename:'[name].js'
    },
    optimization: {
        minimize: true,
        minimizer: [new ESBuildMinifyPlugin({target: 'es2015'})],
    },
    resolve: {
        extensions: ['.js','.jsx']
    },
    target: 'web',
    watch: true,
}

```

Kuva 7. Webpack-moduuliyhdistimen asetukset

Asetan vielä parametrin ”watch” todeksi, jolloin Webpack ajaa päällä ollessaan tiedostojen muunnoksen aina kun lähdetiedostoihin tallennetaan muutoksia. Webpack käynnistetään komentokehotteen kautta komennolla ”npx webpack”, ja sen voi tarvittaessa sammuttaa komentokehotteessa näppäinyhdistelmällä ”Ctrl + C”. Alkuun suoritus antaa virheen, mutta suorituksen tulisi käynnistyä onnistuneesti luomalla lohkoille tiedostot.

4.2.4 Kainalojuttulohko

Kainalojuttu on painetussa mediassa artikkeliosio, joka ei ole osa laajempaa artikkelin leipätekstiä, mutta liittyy artikkelin käsittelemään aiheeseen ja tuo aiheeseen kontekstia. Haluan luoda vastaavan tapaisen sisältöä korostavan lohkon, jota voitaisiin hyödyntää esimerkiksi verkkolehdessä tai osana verkkosivuston artikkelia. Suunnitelmanani on toteuttaa lohko siten, että sen sisälle voidaan

lisätä muita sisältölohkoja, esimerkiksi kuvia, tekstiä ja listoja. Ideana on, että kainalojuttulohko muuttaa sisällään olevien lohkojen tyylisääntöjä siten, että ne erottuvat laajemmasta artikkelista omana kokonaisuutenaan, ja ovat samalla keskenään yhtenäisiä eri artikkelien kesken.

Aloitan lohkon kehittämisen luomalla sille JSX-tiedoston aiemmin luomaani blocks-kansioon. Kaikki WordPressissä käytettävät lohkot tulee rekisteröidä, joten lisään ensimmäiseksi viittauksen lohkon JavaScriptin ajon aikana rekisteröivään registerBlockType-funktioon, joka löytyy globaalista muuttujasta blocks-moduulin alta. Funktiota voisi kutsua myös suoraan koodissa viittaamalla ”window.wp.blocks.registerBlockType”, mutta yleisen selkeyden ja luettavuuden vuoksi lisään funktiot viittauksen avulla. Kainalojuttulohkolle halusin mahdollisuuden muiden lohkojen liittämiseksi lohkon sisälle, joten lisään globaalista muuttujasta blockEditor-moduulin alla olevat InnerBlocks- sekä useBlockProps- funktiot. Kuvassa 8 on esitetty tiedostoriippuvuuksien esittely lohkon rekisteröivän JSX-tiedoston yläosassa.

```
wp-content > plugins > joonamellin-blocks > block > Kainalojuttu.jsx > ...
1   const { registerBlockType } = window.wp.blocks;
2   const { InnerBlocks, useBlockProps } = window.wp.blockEditor;
3
```

Kuva 8. Kainalojuttulohkon lataamat tiedostoriippuvuudet

Aloitan lohkon rekisteröinnin lisäämällä registerBlockType-funktion. Sen ensimmäiseksi parametriksi asetetaan lohkon nimiavaruudesta ja nimestä yhdistetty merkkijono, tässä tapauksessa siis ”joonamellin-blocks/kainalojuttu”. Lisäksi funktiolle tulee antaa kontekstoivaa tietoa objektin muodossa. Asetan lohkon nimeksi ”Kainalojuttu”, ja lisään sille lyhyen kuvauksen sekä muutaman kuvaavan asiasanan, joiden avulla lohkoa voidaan etsiä editorissa. Lisään lohkolle myös ikonin, käyttäen WordPressin Dashicon-ikoneihin kuuluvaa ikonia. Lohkon kategoriaksi asetan alkuun ”text”, jolloin lohko näytetään tekstielementtien joukossa.

Näiden ohella lohkolle on lisättävä myös sen tulosteen ja toiminnallisuuden määrittävät funktiot, jotka asetan suoraan rekisteröintifunktion kolmanneksi ja neljänneksi parametriksi nuolinotaatiofunktioina. Edeltävä näistä on ”edit”, joka määrittää editorin tulosteen. Jälkimmäinen on ”save”, jonka pohjalta määritetään tietokantaan tallennettava, julkaistulla sivustolla näytettävä sisältö.

Aloitan lohkon editorifunktion lisäämällä sille tulosteen, jonka sisälle lisään yhden ympäröivän div-elementin, sillä React-komponenttien palautuksissa voi olla vain yksi juuritason elementti. Tämän sisälle lisään aiemmin viittaamani InnerBlocks -komponentin, joka mahdollistaa muiden lohkojen liittämisen tämän lohkon sisälle. Kyseinen komponentti lisää lohkon sisälle editorissa

lohkonlisäimen, jonka avulla ylläpitäjä voi valita sisälle liitettävän sisällön. Toimiakseen se vaatii, että ympäröivään div-elementtiin liitetään ”useBlockProps” -funktio. Tallennefunktio toimii muuten samoin, mutta InnerBlocks-komponentin sijaan viitataan funktion sisällä olevaan Content-metodiin. BlockPropseista käytetään vastaavasti funktion sisällä olevaa save-metodia. Kuvassa 9 on kuvattu kainalojuttulohkon rekisteröivä funktio lohkon JSX-tiedostossa.

```
registerBlockType('joonamellin-clientside-blocks/kainalojuttu', {
    title: 'Kainalojuttu',
    description: 'Luo kainalojuttuosion, jonka sisällä olevat lohkot näytetään eri tavalla muotoiltuna',
    category: 'joonamellin_blocks',
    icon: 'format-aside',
    keywords: [
        'kainalojuttu',
        'text',
        'article',
        'highlight'
    ],
    edit: () => {
        const blockProps = useBlockProps();

        return(
            <div { ...blockProps }>
                <InnerBlocks />
            </div>
        );
    },
    save: () => {
        const blockProps = useBlockProps.save();

        return(
            <div { ...blockProps }>
                <InnerBlocks.Content />
            </div>
        );
    }
});

```

Kuva 9. Kainalojuttulohkon rekisteröinti

Webpackin ollessa päällä, koodi on muutettu toimivaksi JavaScript-koodiksi dist-kansioon. Koodia ei kuitenkaan oletuksena ladata WordPressin suorituksen aikana. Jotta koodi välitettäisiin editorille, tulee tiedosto lisätä suorituksen latausjonoon. Lisään tiedoston lataamisen lisäosan suorituksen aloittavaan ”joonamellin-blocks.php”-tiedostoon, käyttäen tarkoitukseen lisättyä action-koukkua ”enqueue_block_editor_assets”. Lisään kyseiseen koukkuun anonyymin funktion, jonka sisällä kutsun koodikatkelmat rekisteröivää ja jonoon lisäävää ”wp_enqueue_script”-funktiota. Funktiolle on lisättävä parametreina jonoon lisättävän JavaScript-koodin nimi, joka on sama kuin mitä rekisteröitiin editorin registerBlockType-funktiossa. Lisäksi sille määritetään mistä ajettava JavaScript-tiedosto löytyy, tässä tapauksessa siis viittaus lisäosan juuressa olevaan dist-kansioon, ja siellä olevaan webpackin käsittelemään JavaScript-tiedostoon. Koska kirjoittamani tiedosto viittasi siinä

käytettyihin funktioihin globaalien muuttujien kautta, tulee muuttujien lataaminen varmistaa lisäämällä tarvittavat moduulit riippuvaisuksina kolmannessa parametrissa. Kuvassa 10 on kuvattu kainalojuttulohkon lisääminen latausjonoon riippuvuuksineen.

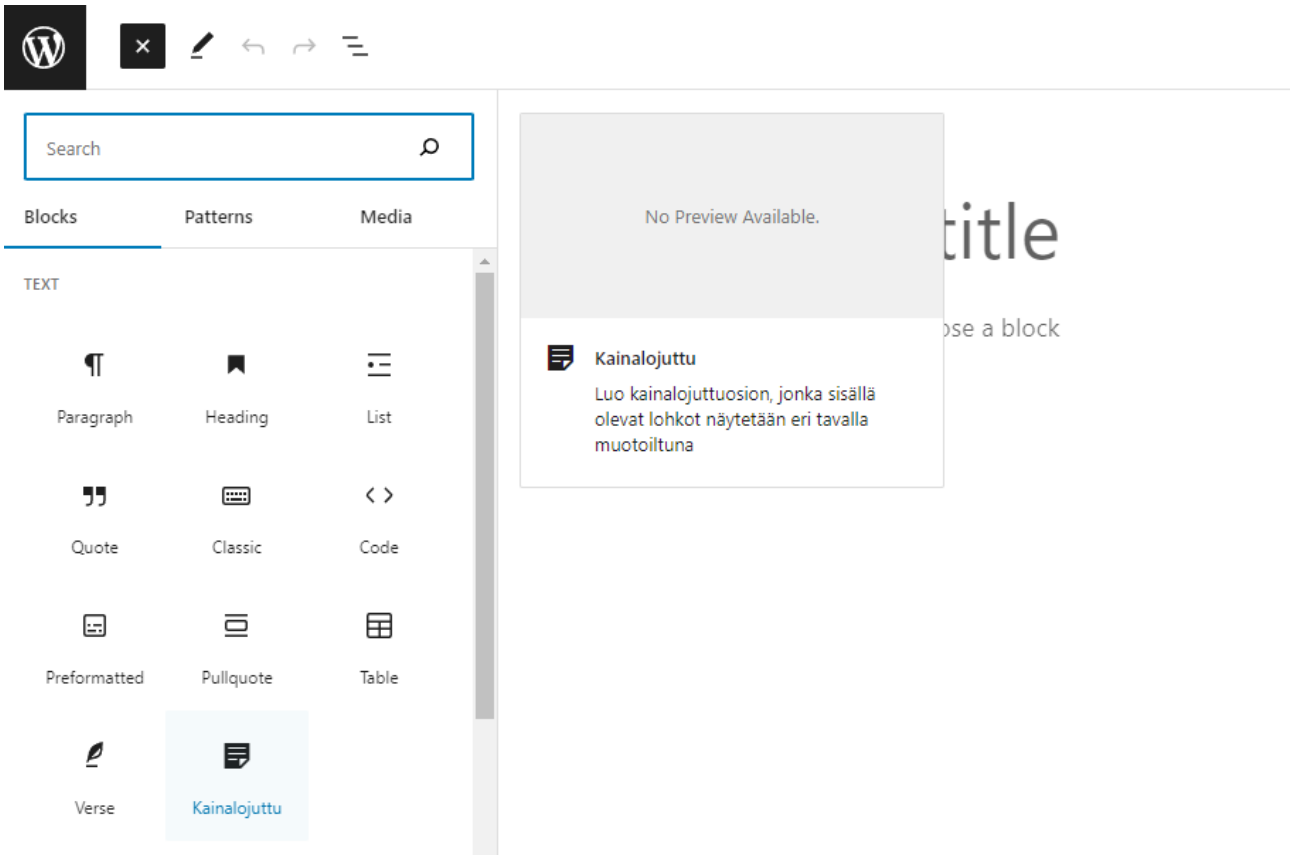
```
add_action('enqueue_block_editor_assets', function () {

    // Kainalojuttu
    wp_enqueue_script('joonamellin-blocks-kainalojuttu', plugins_url('/dist/kainalojuttu.js', __FILE__),
    ['wp-blocks','wp-element','wp-components','wp-i18n', 'wp-editor']);

});
```

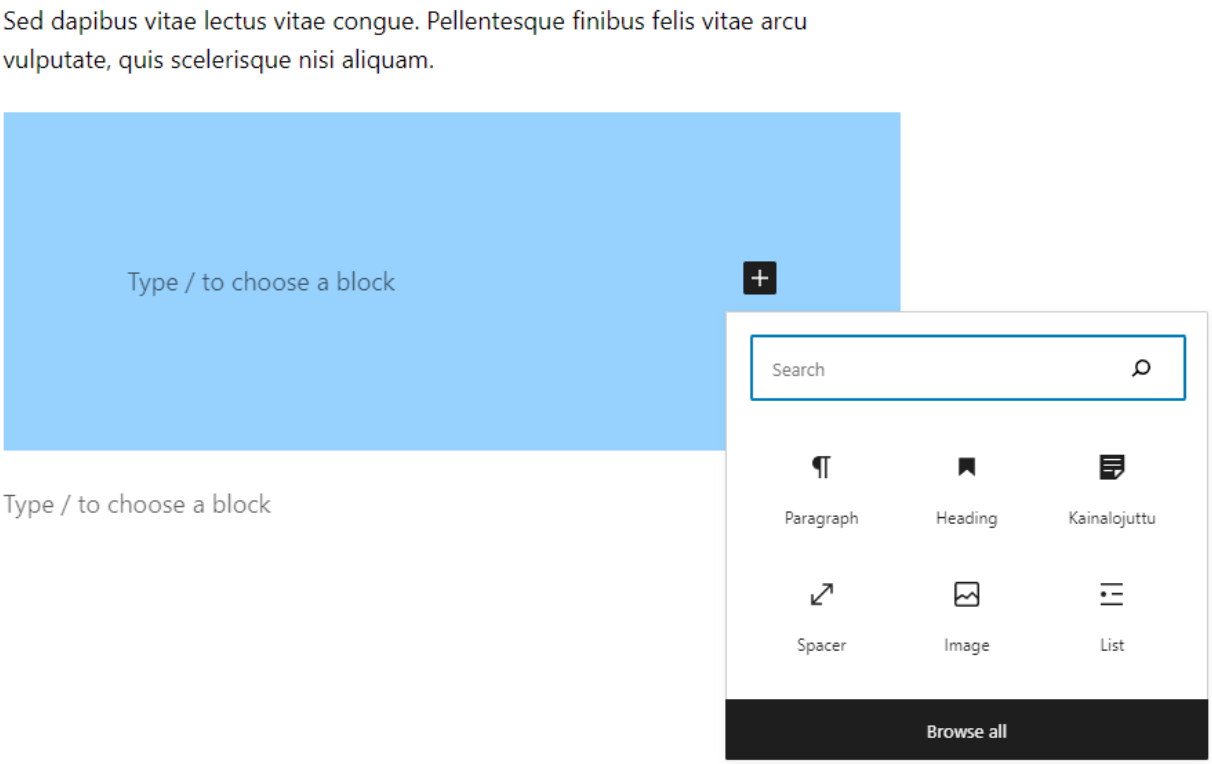
Kuva 10. Kainalojuttulohkon koodin lisääminen suorituksen latausjonoon

Kytkin aiemmin lisäosan päälle, ja Webpack muuttaa JSX-syntaksissa kirjoitettua koodia taustalla natiiviksi JavaScript-koodiksi. JavaScript-tiedosto puolestaan asetettiin ladattavaksi editoriin. Lohkon pitäisi nyt siis näkyä editorissa lohkon lisäävässä työkalussa. Luon uuden sivun, ja avaan sen editorin. Lohkon rekisteröiminen on toiminut oikein, sillä lohko löytyy nyt valitsimesta. Kuvassa 11 on esitetty kainalojuttulohko lohkojen lisäämisen valikossa.



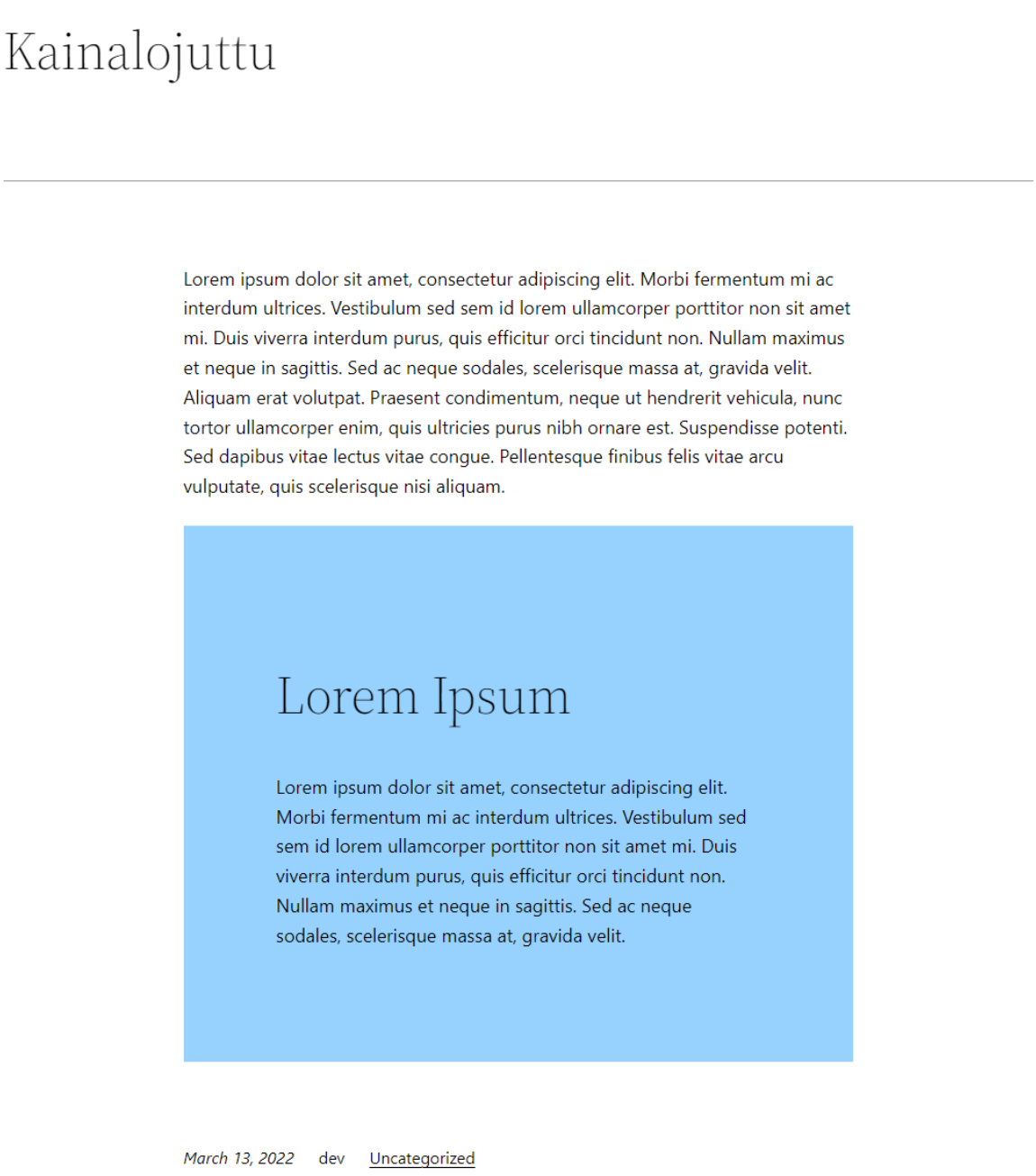
Kuva 11. Kainalojuttulohko lohkojen lisäämisen valikossa

Kun lohkon lisää editorista käyttöön, se aukeaa käytettäväksi määritettyjen teemasääntöjen mukaan. Esimerkkikuvassa teemaan on määritetty vaaleansininen taustaväri sekä marginaalia lohkon editori- ja tallennusosioissa käytetylle ”kainalojuttu” CSS-luokalle. Lohkon sisällä on uuden lohkon lisäämiseen tarkoitettu tyhjä tekstikappalelohko, jolla lohkoon voidaan luoda teksti- ja muuta sisältöä. Lisään lohkolle esimerkkitekstiksi painoissa ja sittemmin verkkosuunnittelussa usein käytettyä pseudosisältöä, lorem ipsumia. Kuvassa 12 on esitetty kainalojuttulohko muun sisällön lomassa lohkoeditorissa.



Kuva 12. Kainalojuttulohko editorissa

Julkaisun jälkeen lohko näkyy myös Front end -käyttöliittymän puolella, eli valmiilla verkkosivulla. Lohkon sisään on lisätty otsikko- sekä kappalelohkot, ja muotoilu ottaa huomioon teemassa määritetyn lisäluokan muotoilun. Kuvassa 13 on esitetty kainalojuttulohkon tulostetta lopullisella sivulla, sisältäen esimerkkisisältöä.



Kuva 13. Kainalojuttulohkon tuloste sivustolla

Koska ylläpitäjälle tarjotaan mahdollisuus lisätä lohkon sisälle muita lohkoja, voivat oletusasetuksilla sisälle asetettavan sisällön muoto ja koko vaihdella merkittävästi. Haluan varmistaa, että lohko on helppo käyttää, ja että sen sisälle voi asettaa vain itse toimivaksi validoimiani lohkoja. Tämän mahdollistamiseksi InnerBlocks-komponentille on luotu ”allowedBlocks”-parametri, joka vastaanottaa listan sallittuja lohkoja. Lohkot tunnistetaan kullekin lohkolle yksilöllisen nimikkeen avulla. Haluan rajata käyttäjän käytössä olevat lohkot WordPressin mukana tuleviin otsikko-, tekstikappale-, tilan lisääjä- sekä listalohkoihin. Etsin lohkojen nimikkeet dokumentaatiosta, ja asetan niistä

koostamani listan allowedBlocks parametrille. Nyt uuden lohkon lisääjä ei anna tämän lohkon sisällä syöttää muita kuin käytettäväksi asettamiani lohkoja.

Jotta käyttäjän olisi helpompi nähdä miten lohkoa on suunniteltu käytettävän, lisään lohkolle sapluunan InnerBlocks-komponentin template-parametrin avulla. Parametri ottaa vastaan listan listoja, joista sisemmissä määritetään sapluunassa käytettävän lohkon nimike, ja toisena listan osana parametreista koostuva objekti. Käytännössä haluan lisätä parametrina placeholder-tekstin, joka näkyy editorissa mallina syötettävästä sisällöstä. Lisään sapluunaan kainalojutun otsikon sekä yhden tekstikappaleen. Käyttäjä voi halutessaan poistaa ne, mutta koska oletan että suurimmassa osassa tapauksia ne aiotaan syöttää lohkon alussa, koen mielekkääksi tulostaa ne heti muokattavaksi editoriin. Kuvassa 14 on kuvattu lohkon editoritulostetta, ja siinä esitetään sallittujen lohkojen ja lohkosapluunan muodostaminen, sekä niiden asettaminen InnerBlocks-komponentille.

```
edit: () => {
    const blockProps = useBlockProps();

    const allowedblocks = [
        'core/heading',
        'core/paragraph',
        'core/spacer',
        'core/list',
    ];

    const blocktemplate = [
        ['core/heading', {
            placeholder: 'Your heading here'
        }],
        ['core/paragraph', {
            placeholder: 'Add the main paragraph here. If you need multiple paragraphs,
        }],
    ];

    return(
        <div { ...blockProps }>
            <InnerBlocks
                allowedBlocks={ allowedblocks }
                template={ blocktemplate }
            />
        </div>
    );
},
```

Kuva 14. Kainalojuttulohkon editorin tuloste

Haluan tarjota ylläpitäjälle mahdollisuuden lohkon tyylien muuttamiseksi, ilman että tämän tarvitsee muuttaa lohkon koodia tai lisätä ylimääräisiä CSS-luokkia. Tämän mahdollistamiseksi lohkoille on mahdollista lisätä lohkotukia (block supports). Nämä kytketään päälle lisäämällä lohkon

rekisteröivään JavaScript-funktioon supports parametri, joka ottaa sisältönä vastaan objektin päälle kytkettäviä lohkotukia. Lisään ensimmäiseksi tueksi värin (color), jonka parametreina asetan todeksi eli muutettavaksi tekstin värin, taustan värin, linkin värin sekä liukuvärit. Haluan että käyttäjä pystyy muuttamaan lohkon asettelua, joten lisään asettelun (align) tuen. Lisäksi mahdollistan välistysten (spacing) muokkaamisen, lisäämällä tuen täytteelle, marginaaleille sekä lohkojen väliselle välistykselle. Viimeiseksi lisään vielä tuen typografialle, asettamalla tekstikoon sekä rivikorkeuden muokattavaksi.

Lohko tukee nyt haluamiani lohkoasetuksia, mutta haluan vielä asettaa jonkinlaiset oletusarvot. Lohkon oletustyylin voi määrittää esimerkiksi erillisessä CSS-tiedostossa. WordPress ylikirjoittaa säännöt, joita ylläpitäjä on muokannut, sillä lohkotuet näemmä kirjoittavat tyylittelyt CSS-muuttujina suoraan elementteihin, jolloin ne ovat spesifisyydeltään tarkemmat (mutta huomioivat mahdolliset muutokset CSS-muuttujiin esimerkiksi teemassa). Poikkeuksena tähän olevan asettelu, joka lisätään luokkana. Dokumentaation mukaan lohkon asettelun align-parametrille voidaan asettaa default-arvo syöttämällä se attribuuttina. Perinteisemmin lehdissä kainalojutut on eroteltu selkeästi muusta sisällöstä esimerkiksi pohjavärin avulla, joten asetan oletukseksi ”full”, jolloin lohkon tausta ylettää näytön reunasta reunaan, erottuen tavallisemmasta tekstisisällöstä selkeämmin.

4.2.5 Lohkokategorian rekisteröiminen

Jotta itse tekemäni lohkot löytyisivät selkeämmin samasta paikasta, haluan lisätä ne omana kategorianaan lohkonlisäystyökaluun. Tätä varten WordPressistä löytyy lohkoeditorin kategorioita suodattava koukku ”block_categories_all”, joka mahdollistaa paitsi olemassa olevien kategorioiden uudelleenjärjestämisen, myös uusien kategorioiden lisäämisen. Koukulle välitetään parametrina lista nykyisistä kategorioista. Lisään sen perään uuden kategorian listana, jonka avaimina on määritetty kategorian nimike sekä otsikko. Lopuksi asetan muutetun listan palautettavaksi, jolloin sitä voidaan käyttää WordPressin suorituksessa aina kun kategorioita listataan, esimerkiksi lohkojenlisäystyökalussa. Muutin vielä lopuksi kainalojutun rekisteröivää funktiota siten, että kategoria osoittaa nyt uuteen luomaani kategoriaan. Kuvassa 15 on kuvattu uuden lohkokategorian lisääminen.

```
19  add_filter('block_categories_all', function ( $categories ) {
20
21      $categories[] = array(
22          'slug' => 'joonamellin_blocks',
23          'title' => 'Joona Mellin Blocks'
24      );
25
26      return $categories;
27  });
```

Kuva 15. Lohkokategorian rekisteröiminen

4.2.6 Viimeisimmät artikkelit -lohko (Server side render)

Verkkosivustoilla tyypillisiä komponentteja ovat erilaiset uusimpia artikkeleita poimivat komponentit. Tässä osiossa luon palvelimen puolella tulostettavan lohkon, joka hakee tietokkanasta viimeisimpiä artikkeleita käyttäjän antamien asetusten perusteella.

Lisään editorin tulosteeseen alustavasti näkyville lyhyen tulosteen kuvauksena lohkon toiminnasta. Tulen myöhemmin tässä luvussa korvaamaan sen esikatselulla palvelimen tekemästä tulosteesta.

Kuten kainalojuttulohkon kanssa, aloitan lohkon kehittämisen lisäämällä ”blocks” kansioon uuden JSX-tiedoston lohkolle, ja lisäämällä niin ikään lohkon rekisteröivän funktion globaalista muuttujasta. Lohkon rekisteröiminen editorille toimii pitkälti samaan tapaan kuin edellisessä tapauksessa. Lisään lohkolle nimen, ikonin, kuvauksen, avainsanat sekä nuolinotaationa toteutetun editorifunktion. Edellisestä asiakkaan puolella renderöitävästä lohkosta poiketen palvelimen puolella tulostettavalle funktiolle ei tarvitse luoda tallennefunktiota, sillä palvelin luo tulosteen palvelimen ajon aikana luettavan PHP-funktion avulla. Lisään editorin tulosteeseen alustavasti näkyville lyhyen tulosteen kuvauksena lohkon toiminnasta. Tulen myöhemmin tässä luvussa korvaamaan sen esikatselulla palvelimen tekemästä tulosteesta. Lisään myös lisäosan PHP-suorituksen aloittavaan tiedostoon lohkon JavaScript-tiedoston jonoon lisäävän wp_enqueue_script-funktion samaan tapaan kuten edellisenkin lohkon kanssa. Kuvassa 16 on esitetty artikkelinostolohkon riippuvuuksien lataaminen sekä kyseisen lohkon rekisteröiminen editorissa. Kuvassa 17 on kuvattu artikkelinoston lisääminen suorituksen latausjonoon riippuvuuksineen.

wp-content > plugins > joonamellin-blocks > block > ViimeisimmatArtikkelit.jsx > ...

```jsx
const { registerBlockType } = window.wp.blocks;
const { InnerBlocks, useBlockProps } = window.wp.blockEditor;

registerBlockType('joonamellin-clientside-blocks/viimeisimmatartikkelit', {
    apiVersion: 2,
    title: 'Viimeisimmät artikkelit',
    description: 'Näyttää viimeisimmät artikkelit',
    category: 'joonamellin_blocks',
    icon: 'format-aside',
    keywords: [
        'latest',
        'post',
        'posts',
        'viimeisimmät',
        'artikkelit'
    ],
    edit: ( props ) => {

        const blockProps = useBlockProps();

        return(
            <div { ...blockProps }>
                Viimeisimmät artikkelit
            </div>
        )
    }
});

```

Kuva 16. Viimeisimpien artikkelien lohkon rekisteröiminen editorin puolella

```php
add_action('enqueue_block_editor_assets', function () {

    // Kainalojuttu
    wp_enqueue_script('joonamellin-blocks-kainalojuttu', plugins_url('/dist/kainalojuttu.js', __FILE__),
    ['wp-blocks', 'wp-components', 'wp-editor']);

    // Viimeisimmät artikkelit
    wp_enqueue_script('joonamellin-blocks-viimeisimmatartikkelit', plugins_url('/dist/viimeisimmatartikkelit.js', __FILE__),
    ['wp-blocks', 'wp-components', 'wp-editor']);

});

```

Kuva 17. Artikkelinostolohkon koodin lisääminen suorituksen latausjonoon

Palvelimen puolella tulostettavalle lohkolle rekisteröinti pitää tehdä editorin JavaScript-suorituksen lisäksi myös palvelimen puolella tehtävän suorituksen aikana. Tätä varten WordPressissä on myös palvelimen puolella funktio rekisteröintiä varten. Lisään suorituksen alkuvaiheilla ajettavaan ”init” action-koukkuun anonyymin funktion, jonka sisällä ajan lohkon rekisteröivän funktion. Sen ensimmäinen parametri on, kuten JavaScriptin vastaavassa funktiossa, lohkon nimike. Toisena parametrina välitetään lista, joka sisältää lohkon palvelinpuolen tulostetta koskevia tietoja, kuten

mahdollisia attribuutteja. Lisään alkuun avaimella ”render_callback” funktion nimen, joka tulee suorittamaan sisällön tulostuksen. Palvelimen puolella lohkon rekisteröivä funktio on kuvattu asetuksineen kuvassa 18.

```
add_action('init', function() {

    register_block_type(
        'joonamellin-clientside-blocks/viimeisimmatartikkelit',
        array(
            'render_callback' => 'joonamellin_block_viimeisimmat_artikkelit',
        )
    );

});
```

Kuva 18. Artikkelinostolohkon rekisteröiminen palvelimen puolella

Seuraavaksi luon äsken nimeämäni sisällön tulostuksen tekevän funktion. Se vastaanottaa argumentteina attribuutteja, joita en ole vielä esitellyt. Aion käyttää tulosteen koostamiseen HTML-syntaksia PHP-syntaksin ohessa, joten käytän tulosteen koostamiseksi tulostepuskurointia. Lisään funktion alkuun kutsun puskuroinnin aloittavaan ”ob_start”-funktion, ja palautan funktion lopuksi puskurin sisällön ”ob_get_clean”-funktiolla. Näiden välissä esitetty PHP- ja HTML-syntaksi koostetaan puskurissa yhtenäiseksi tulosteeksi, joka palautetaan tätä funktiota kutsuneeseen paikkaan. Tulosteen koostava funktio sekä ylivuotopuskurin käynnistäminen ja sulkeminen on esitetty kuvassa 19.

```
51  function joonamellin_block_viimeisimmat_artikkelit ( $attributes ) {
52      ob_start();
53
54      return ob_get_clean();
55  }
```

Kuva 19. Lohkon palvelinpuolen tulosteen tulostava funktio

Lohkon tavoitteena on tulostaa sivustolle lisättyjä artikkeleita, mikä onnistuu WordPressissä palvelimen puolella esimerkiksi WP_Query-luokan avulla. Esittelen uuden parametrin ”posts”, jonka arvoksi määritän uuden instanssin WP_Query luokasta. WP_Query-objekti vastaanottaa parametrina listan argumenteista, joten lisään niiden muokkaamista varten listan ”queryarguments”. Ensimmäisenä argumenttina rajaan, että näytettäväksi haetaan vain julkaistuja artikkeleita. Artikkelityypiksi

asetan ”post” eli artikkelit, ja artikkelien sivukohtaiseksi määräksi asetan 5. Artikkelin alustavat argumentit on kuvattu tarkemmin kuvassa 20.

```
51  function joonamellin_block_viimeisimmat_artikkelit ( $attributes ) {
52      ob_start();
53
54      $queryarguments = array(
55          'post_status' => 'publish',
56          'post_type' => 'post',
57          'posts_per_page' => 5,
58      );
59
60      $posts = new WP_Query( $queryarguments );
61
62      return ob_get_clean();
63  }
64
```

Kuva 20. Artikkelihaun alustavat argumentit

Puran seuraavaksi haun tulokset käyttämällä while-luuppia, sekä WP_Query-objektiin sisältyvää ”the_post” -metodia. Metodi hakee vuorossa olevan artikkelin tarkasteltavaksi, jonka jälkeen WordPressin omat artikkelin tietoihin osoittavat funktiot hakevat oletuksena tätä luupissa olevaa artikkelia. Poimin WordPressin perusfunktioilla muuttujiin artikkelin otsikon, päivämäärän, katkelman sekä linkin. Puskuroinnin ollessa päällä voin päättää PHP koodin hetkeksi, ja lisätä väliin HTML-merkintöjä. Kirjoitin kullekin artikkelille yksinkertaisen HTML-tulosteen, jossa otsikko toimii linkkinä artikkeliin, ja muu tieto näytetään artikkelinoston yhteydessä otsikon alapuolella. Kiedoin vielä koko luupin ympärille div-elementin, jotta voin helpommin kohdistaa muotoiluja artikkelinostoihin esimeriksi antamalla div-elementille flex- tai grid-näyttömoodin. Palautan lopuksi artikkelidatan osoittamaan artikkeliin, jonka sisällä lohko on, käyttämällä wp_reset_postdata-funktiota, sillä muuten osoitus jäisi viimeiseen luupin sisällä olleeseen artikkeliin. Kuva 21 esittää artikkelien argumenttien määrittämistä, artikkelien hakemista sekä tulosten perkaamista lohkon tulosteeseen.

```php
function joonamellin_block_viimeisimmat_artikkelit ( $attributes ) {
    ob_start();

    $queryarguments = array(
        'post_status' => 'publish',
        'post_type' => 'post',
        'posts_per_page' => 5,
    );

    $posts = new WP_Query( $queryarguments );

    ?><div class="viimeisinartikkeli"><h2>Viimeisimmät artikkelimme</h2><?php

    while ( $posts -> have_posts()) {
        $posts->the_post();
        $title = get_the_title();
        $date = get_the_date();
        $excerpt = get_the_excerpt();
        $link = get_the_permalink();
        $author = get_the_author();

    ?>
    <article>
        <h3><a href="<?php echo $link; ?>"><?php echo $title; ?></a></h3>
        <p><?php echo $excerpt; ?></p>
        <small><?php echo $date . ' by: ' . $author; ?></small>
    </article>
    <?php
    } // End while loop
    wp_reset_postdata();
    ?>
    </div>
    <?php
    return ob_get_clean();
}

```

Kuva 21. Artikkelien haku ja tulosten käsittely

Asettamalla lohkon editorissa voin todeta lohkon toimivan, sekä löytyvän oikean kategorian alta. Se tulostaa lisäksi tallennuksen jälkeen artikkeleita PHP-funktion tulosteen mukaisesti.

Haluan kuitenkin parannella lohkon käytettävyyttä mahdollistamalla artikkelikategorian rajaamisen, sekä tarjoamalla asetuksen näytettävien artikkeleiden määrän muokkaamiseksi. Molemmat asetukset lisätään editoriin lohkon asetuksiin, ja ne muuttavat loppukäyttäjälle frontendissä näkyvää tulostetta.

Aloitan asetusten luomisen lisäämällä ne editorin puolen rekisteröintiin uusina attribuutteina, eli lohkon lisätietoina. Nämä annetaan objektilistana, jossa kullekin attribuutille määritetään ainakin attribuutin tyyppi, sekä mahdollisesti alkuarvo. Lisään lohkon attribuuteiksi listan kategorioista, joita käytetään valinnan tekemiseen. Seuraavaksi lisään valitun kategorian, johon tallennetaan käyttäjän valinta. Viimeiseksi lisään kokonaisluku -muotoisen attribuutin kerrallaan näytettävien artikkeleiden määrälle. Kuvassa 22 on esitetty artikkelinostolohkon attribuuttien määrittely JSX-tiedostossa.

```
20      attributes: {
21          categories: {
22              type: 'array'
23          },
24          selectedCat: {
25              type: 'array'
26          },
27          filterByCat: {
28              type: 'boolean'
29          },
30          postsPerPage: {
31              type: 'int',
32              default: 5
33          }
34      },
```

Kuva 22. Artikkelinostolohkon attribuutit

Haen seuraavaksi kategoriat muuttujaan, hyödyntäen WordPressin sisäänrakennettua REST-rajapintaa. Globaali wp-muuttuja sisältää apiFetch-metodin, jonka avulla rajapinnasta voidaan hakea tietoja. Funktiolle annetaan parametreista muodostettu objekti, johon sisällytän tässä tapauksessa vain URL-osoitteen, josta haluan sisältöä haettavan. Kategorioita hakiessa oikea rajapinnan osoite on ’../wp-json/wp/v2/categories”. Käytän sen jälkeen samaan funktioon kuuluvaa then-metodia, joka käsittelee asynkronisesti haetun datan. Tiedon tallentamiseksi kantaan käytän lohkojen rekisteröintifunktion propsille lisättyä metodia setAttributes, jolla propseihin asetettuja tietoja voidaan päivittää. Asetan kategoria-attribuutin arvoksi rajapinnasta palautetun kategorialistan. Käärin hakufunktion tarkistukseen siitä, onko attribuutteja vielä haettu – data haetaan rajapinnasta vain, jos se ei ole ennestään olemassa. Varaudun myös virheiden välttämiseksi tilanteeseen, jossa kategorioita ei ole tallennettuna tai niitä ei rajapinnan avulla löytynyt. Varaudun lisäksi vielä tilanteeseen, jossa haku on vielä käynnissä. Molemmissa tilanteissa ylläpitäjän nähtäville tulostetaan editoriin kuvaava lauseke, joka korvataan oikealla editorin tulosteella reaktiivisesti, kunhan tiedot on onnistuneesti haettu. Kategorioiden hakemisen apufunktio sekä hakuun ja tyhjiin tuloksiin varautuminen on esitetty kuvassa 23.

```
39      if ( !props.attributes.categories ){
40          wp.apiFetch({
41              url: '../wp-json/wp/v2/categories'
42          }).then( categories => {
43              console.log('categories');
44              console.log(categories);
45              props.setAttributes( {
46                  categories: categories
47              })
48          });
49      }
50
51      if ( !props.attributes.categories ) {
52          return 'Fetching categories...';
53      }
54
55      if ( !props.attributes.categories && props.attributes.categories.length === 0) {
56          return "This site doesn't seem to have categories"
57      }
58
```

Kuva 23. Artikkelinostolohkon kategorioiden hakeminen ja virhetilanteiden käsittely

Kategoriat haetaan nyt attribuutteihin, mutta tarvitsen ylläpitäjälle jonkinlaisen asetuskentän, josta tämä voi valita haluamansa kategorian. Asetuskentät voisi ohjelmoida suoraan lohkon edit -osioon lomakkeena, mutta haluan noudattaa editorin mukana tulevien lohkojen noudattamaa WYSIWYG-ajattelua myös omien lohkojeni osalta. Käytännössä editorin esikatselun tulisi näyttää mahdollisimman tarkalle esikatselulle sivuston lopullisesta ulkoasusta, ja asetukset yms. ylläpitäjälle kontekstoivaa tietoa tarjoavat asiat tulisi mieluummin sijoittaa lohkon työkaluriville tai sivupalkin lohkoasetuksiin. Työkalupalkin asetukset keskittyvät pääasiassa inline-tyylittelyihin, joten koen sivupalkki-asetukset luontevammaksi sijoituspaikaksi tämän kaltaisille asetuksille. Sijoittamalla asetukset sivupalkkiin ylläpitäjälle muodostuu parempi käsitys muokkaamansa sivun lopullisesta ulkoasusta jo editorissa, ja lohkon asetukset löytyvät intuitiivisesti sivupalkin lohkoasetuksista. Sivupalkin asetusten jatkamiseen on tehty globaalin wp-muuttujan alta löytyvä InspectorControls-komponentti, jonka sisään voidaan lisätä muuta sisältöä. Seuratakseni editorin tyylejä käytän globaalin muuttujan components-kirjaston alla olevia paneelikomponentteja tietojen jaottelemiseen lohkoasetuksissa.

Asetan InspectorControls-komponentin editorin tulosteen alkuun, ja sijoitan sen sisälle yhden paneeliosion PanelBody-komponentin avulla. Asetan sille otsikoksi ”Attributes”, sillä aion kerätä kaikki attribuuttien asetukset tämän osion sisään. Käytän tämän sisällä vielä PanelRow-komponenttia yksittäisten asetusten jakamiseksi omille riveilleen. Itse asetusten muokkaamista varten components-kirjasto sisältää erilaisia hallintaelementtejä. Mikään ei estä käyttämästä HTML input-vakioelementtejä, mutta sisäänrakennetut komponentit ovat muotoilulta ja saavutettavuudeltaan yhtenäiset muun editorin kanssa, joten päädyn käyttämään niitä tämän lohkon kanssa.

Koska kategorioita on muutamia, muttei kuitenkaan kymmenittäin, koen valintalistan olevan tässä yhteydessä järkevin elementti. Tällainen on kirjastossa oleva SelectControl-komponentti, jonka lisään paneelin ensimmäisen rivin sisälle. Asetan sen kuvaukseksi ”Category” sisältötyypin mukaan, ja arvoksi propsin alla olevien attribuuttien valittu kategoria -tiedon. Tämän tulee olla tyhjä tai täsmätä arvoltaan johonkin listalla olevaan valintaan. Itse valinnat olen hakenut attribuutteihin kategorialistaan, joten puran listan natiivin JavaScriptin map-metodilla. Asetan kunkin valinnan arvoksi kategorian yksilöivän tunnuksen, ja kuvaukseksi kategorian nimen. Päivittyneiden tietojen käsittelemiseksi lisään muutoksen yhteydessä suoritettavaksi updateCategory-funktion, jonka luon seuraavaksi. Funktio vastaanottaa käyttäjän valitseman arvon, ja asettaa sen lohkon rekisteröinnin propsin alla olevalla setAttributes-metodilla valitun kategorian arvoksi. SetAttributes-metodia voisi kutsua myös suoraan, mutta käytän omaa funktiota tiedon välittämiseen, jotta voin tarvittaessa esimerkiksi validoida arvoja oman funktion sisällä ennen tallentamista.

Näytettävien artikkeleiden määrän valitsemiseksi tarvitsen komponentin johon käyttäjä voi syöttää haluamansa numeroarvon. Löysin komponenttikirjaston lähdekoodia selaamalla kokeellisen NumberControl-komponentin, joka vaikutti toimivan hyvin. Kyseessä on kuitenkin kokeellinen toiminto, joka saattaa muuttua ja rikkoutua versiopäivitysten myötä, joten päädyin vaihtamaan sen paremmin tuettuun TextControl-komponenttiin samasta kirjastosta. Tälle komponentille voi asettaa tyypiksi numeron, jolloin kenttä antaa käyttäjän syöttää ainoastaan kokonaislukuja. Lisään kentän arvoksi postsPerPage-attribuutin arvon, ja lisään tietojen muuttumisen yhteydessä suoritettavaksi updatePostPerPage-metodin, joka edellisen kentän tavoin lisää arvon attribuutille SetAttributes-metodin avulla.

Lisään vielä erillisen asetuksen sille huomioidaanko kategoriafiltteri haussa. SelectControl-komponentti edellyttää, että jokin kategoria on valittuna, mutta haluan tukea myös tilannetta, jossa käyttäjä haluaa hakea kaikkien kategorioiden artikkelit. Tämän voisi toteuttaa esimerkiksi luomalla tyhjän valinnan, mutta mielestäni on selkeämpää luoda boolean-asetus, jonka avulla käyttäjä voi valita käytetäänkö suodatinta vai ei. Käytän tarkoitukseen components-kirjastosta löytyvää ToggleControl-komponenttia, ja asetan sille kuvauksen ohella arvoksi uuden boolean-attribuutin filterByCat, jonka lisään myös lohkon rekisteröinnin yhteydessä esiteltäviin attribuutteihin.

Kuvassa 24 on esitetty asetuspaneelin ohjelmakoodi kokonaisuudessaan, sisältäen sekä paneelirivien että eri asetuskenttien rekisteröinnit. Asetuskentät on liitetty omiin päivitysfunktioihinsa, jotka on kuvattu kokonaisuudessaan kuvassa 25.

```
<InspectorControls>
    <PanelBody title="Attributes">
        <PanelRow>
            <ToggleControl
                label="Filter by category"
                checked = { props?.attributes?.filterByCat }
                onChange= { updateCategoryFilter }
                />
        </PanelRow>
        <PanelRow>
            <SelectControl
                label="Category"
                onChange={ updateCategory }
                value={ props.attributes.selectedCat }
                options = {
                        props.attributes.categories.map((c) =>
                        ({ value: c.id, label: c.name })
                        )
                }
                />
        </PanelRow>
        <PanelRow>
            <TextControl
            label="Posts per page"
            type="number"
            value={ props.attributes.postsPerPage }
            onChange={ updatePostsPerPage }
            />
        </PanelRow>
    </PanelBody>
</InspectorControls>
```

Kuva 24. Artikkelinostolohkon lohkoasetukset

```
59      function updateCategory( value ) {
60          props.setAttributes({
61              selectedCat: value
62          });
63      }
64
65      function updatePostsPerPage( value ) {
66          props.setAttributes({
67              postsPerPage: value
68          });
69      }
70
71      function updateCategoryFilter( value ) {
72          props.setAttributes({
73              filterByCat: value
74          });
75      }
76
```

Kuva 25. Artikkelinostolohkon asetuksia päivittävät apufunktiot

Nyt kun attribuutit on rekisteröity editorin skripteissä, ne tulee vielä rekisteröidä lohkon asetuksiin PHP-koodin puolella. Lisäksi minun tulee vielä muuttaa lohkon sisällön tulostavaa funktiota siten, että se huomioi artikkeleiden hakuasetuksissa editorissa tallennetut attribuutit. Lisään ensimmäiseksi attribuutit lohkon palvelimenpuolella rekisteröivään funktioon lisäosan aloittavassa tiedostossa. Attribuutit lisätään lohkon rekisteröintiin listamuodossa. Listan avaimeen laitetaan attribuutin nimi ja arvoksi attribuutin tyyppi listana. Lohkojen rekisteröinti palvelimen puolella on kuvattu kokonaisuudessaan kuvassa 26.

```
add_action('init', function () {
    register_block_type(
        'joonamellin-clientside-blocks/viimeisimmatartikkelit',
        array(
            'render_callback' => 'joonamellin_block_viimeisimmat_artikkelit',
            'attributes' => [
                'selectedCat' => [
                    'type' => 'integer'
                ],
                'postsPerPage' => [
                    'type' => 'integer'
                ],
                'filterByCat' => [
                    'type' => 'boolean'
                ]
            ]
        )
    );
});

```

Kuva 26. Artikkelinostolohkon asetusten rekisteröinti palvelimen puolella

Kun attribuutit on rekisteröity myös taustajärjestelmässä, voidaan ne ottaa vastaan lohkon sisällön tulostavassa palvelinpuolen tulostefunktiossa. Lisään attribuutit muuttujana funktion parametriksi. Irrotan niistä kategorian, suodatuksen sekä sivulla näytettävien artikkelien määrän omiksi muuttujikseen, ja lisään ne argumentteihin, joiden perusteella artikkelit haetaan taustajärjestelmästä. Lisään myös oletusarvot, joita käytetään, jos attribuuttia ei jostain syystä löytyisi. Attribuuttien liittäminen parametreihin ja liittäminen hakuparametreihin on kuvattu kokonaisuudessaan kuvassa 27.

```
function joonamellin_block_viimeisimmat_artikkelit ( $attributes ) {
    ob_start();

    $postsperPage = $attributes['postsPerPage'] ? $attributes['postsPerPage'] : 5;
    $selectedCategory = $attributes['selectedCat'] ? $attributes['selectedCat'] : 0;
    $filterbycategory = $attributes['filterByCat'] ? $attributes['filterByCat'] : false;

    $queryarguments = array(
        'post_status' => 'publish',
        'post_type' => 'post',
        'posts_per_page' => $postsperPage,
    );

    if ( $filterbycategory ) {
        $queryarguments['cat'] = $selectedCategory;
    }

    $posts = new WP_Query( $queryarguments );

```

Kuva 27. Artikkelinostolohkon attribuuttien käsitteleminen

Attribuutit huomioidaan nyt loppukäyttäjille näytettävässä sisällössä, eli ylläpitäjä voi muuttaa näytettävää sisältöä asetusten avulla. Olisi kuitenkin ylläpitäjäkäyttäjän kannalta helpompaa, mikäli tämä lisäksi näkisi editorissa esikatselun näytettävästä sisällöstä. WYSIWYG-ajattelun mukaan editorin tulosteen tulisi muistuttaa lopullista tulostetta mahdollisimman tarkasti, jotta sisältöä lisäävälle ylläpitäjälle muodostuisi mahdollisimman tarkka kuva lopullisen sivun sisällöstä. Editoriin on rakennettu komponentti serverSideRender, jolla palvelimen puolella olevia lohkon tulosteita voidaan esikatsella editorissa. Komponentin saa käyttöön globaalin wp-muuttujan alta, ja se tulostaa attribuuttien perusteella nimensä mukaisesti palvelimen puolella renderöitäviä lohkoja.

Jostain syystä komponentti on nimetty pienellä alkukirjaimella, jolloin JSX ei heti ymmärrä sitä komponentiksi. Tämä on kuitenkin kierrettävissä lisäämällä komponentti aliasta eli itse määritettyä viittausnimeä käyttäen. JSX ei siten ymmärrä suoraan kyseessä olevan komponentti, ellemme käytä ohjelmassa suoraan ”wp.serverSideRender” tai anna muuttujalle aliasta. Lisäsin komponentin mukaan isolla kirjoitettuna. ServerSideRender komponentin liittäminen riippuvuutena itse määritettyä aliasta hyödyntäen on kuvattu kuvassa 28.

```
const { registerBlockType } = window.wp.blocks;
const { serverSideRender : ServerSideRender } = wp;
```

Kuva 28. ServerSideRender-komponentin lisääminen editorin riippuvuudeksi

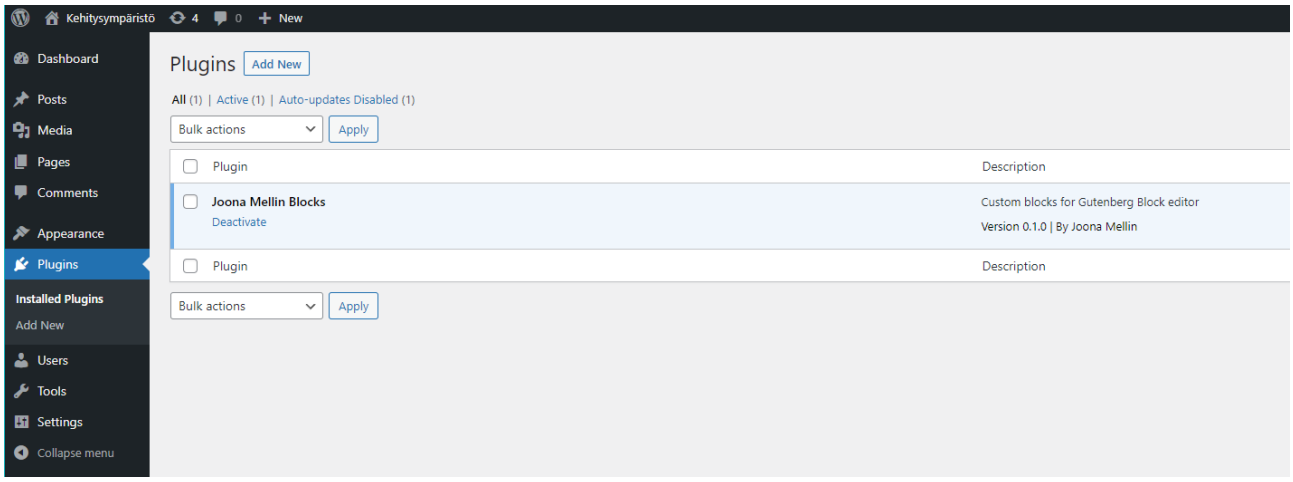
Lisään komponentin editorin tulosteeseen sivupalkkiasetusten jälkeen. Komponentille annetaan parametreina lohko, joka halutaan tulostaa, sekä attribuutit, joiden pohjalta tuloste tehdään. Asetan parametreiksi attribuuttien tämänhetkiset arvot, jolloin tuloste päivittyy reaktiivisesti aina kun jokin arvoista muuttuu. Käyttäjä näkee siis valitsemiensa asetusten vaikutuksen lyhyellä vasteajalla suoraan editorin tulostealueella. Attribuuttien esittäminen komponentille on kuvattu kuvassa 29.

```
110                <ServerSideRender
111                    block="joonamellin-clientside-blocks/viimeisimmatartikkelit"
112                    attributes= {{
113                        selectedCat: props.attributes.selectedCat,
114                        postsPerPage: props.attributes.postsPerPage,
115                        filterByCat: props.attributes.filterByCat
116                    }}
117                />
```

Kuva 29. ServerSideRender-komponentin attribuuttien rekisteröiminen

4.3 Tuotos

Projektin tuotoksena syntyi WordPress-lisäosa, jonka aktivoimalla ylläpitäjä saa käyttöönsä kaksi uutta sisältölohkoa. Sisältölohkot on rekisteröity käytettäväksi lohkoeditorissa, ja ylläpitäjä pystyy niiden avulla lisäämään mukautettua sisältöä verkkosivun tulosteeseen. Uudet lisäosat on aseteltu omaan lohkokategoriaansa, jotta käyttäjän olisi helppo löytää ne sekä toisaalta tunnistaa mitkä lohkot on lisätty luomani lisäosan avulla. Kuvassa 30 on esitetty lisäosan näkyminen hallintapaneelin lisäosat -sivun listauksessa.

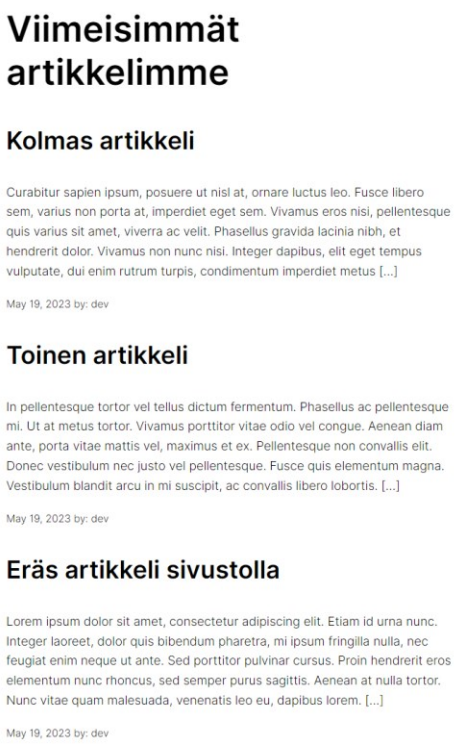


Kuva 30. Luomani lisäosa hallintapaneelissa

Kainalojuttulohkon avulla käyttäjä pystyy lisäämään artikkeliin kainalojuttuosion, eli aiheeseen liittyvän artikkeliosion, joka on irrallinen artikkelin päätekstin virrasta, mutta tuo aiheeseen kontekstia. Lohko mahdollistaa rajatun kaltaisten muiden sisältölohkojen sisällyttämisen sisäänsä, jolloin ne omaksuvat kainalojuttuartikkelin muotoilun. Muotoilun asetukset on asetettu lohkon sivupalkkiasetuksiin sekä työkalupalkkiin, jolloin ne noudattavat WordPressin mukana tulevien lohkojen tapaan. Sisältöä muokkaava ylläpitäjä näkee reaaliaikaisesti editorissa asetusten vaikutuksen tulosteeseen. Ylläpitäjä voi muuttaa lohkon tekstin, taustan sekä linkkien värejä. Tämä voi myös muuttaa lohkon asettelua olemaan sivupohjan levyinen, leveä tai täysileveä. Myös leipätekstin kokoa sekä rivikokoa voi muuttaa kainalojutun asetusten avulla. Esimerkkituloste valmiista kainalojuttulohkosta on esitetty kuvassa 31.

Kuva 31. Kainalojuttulohkon tuloste

Viimeisimmät artikkelit -lohkon avulla käyttäjä pystyy lisäämään nostoelementin, joka näyttää valittuja tietoja sivustolle lisätyistä artikkeleista. Ajatuksena on, että nostolohko ehdottaa sivuston lukijalle artikkeleita luettavaksi, ja tämä pystyy hyperlinkin avulla siirtymään yksittäisen artikkelin sivulle lukemaan koko sisällön. Ylläpitäjälle tarjotaan lohkon sivupalkkiasetusten kautta mahdollisuus muuttaa noston yhteydessä näytettävien artikkeleiden määrää, sekä rajata näytettäviä artikkeleita artikkelin kategorian perusteella. Ylläpitäjän on mahdollista myös valita, ettei kategoriarajausta tehdä, jolloin nostolohko näyttää artikkeleita kaikista kategorioista. Valmiin artikkelinostolohkon tuloste on esitetty tarkemmin kuvassa 32.



Kuva 32. Artikkelinostolohkon tuloste

5 Pohdinta

Tämän opinnäytetyön tutkimuskysymykset ovat olleet: Miten sisältöä kehitetään lohkoeditorille? Miten luon yksinkertaisimman mahdollisen toimivan lohkon (MVP)? Miten luon tyypillisiä sisältölohkoja hyödyntäen moderneja verkkokehityskirjastoja? Miten hyödynnän lohkoille tarkoitettuja rajapintoja ja komponentteja?

Sisältöä kehitetään editorille monipuolisesti eri kielien avulla, mutta yhtenäistä on, että lohko tulee rekisteröidä vähintään JavaScriptin puolella, ja palvelimen renderöintiä tehdessä myös palvelimen puolella. Kun rekisteröinti on toteutettu, kehittäjä voi vapaasti käyttää käyttöliittymän tekemiseen esimerkiksi pelkkää HTML-syntaksia, tai laajentaa sitä sopivaksi kokemallaan JavaScript-kirjastolla. Päädyin itse käyttämään Reactia ja JSX-syntaksia niiden ollessa itselleni tutuin yhdistelmä. Koska editorin komponentit on kehitetty React:illa, tämä on myös intuitiivisin tapa käyttää editorin mukana ladattavia käyttöliittymäkomponentteja kuten asetuskomponentteja.

Yksinkertaisin mahdollinen lohko sisältäisi vain JavaScript-tiedoston, joka rekisteröitäisiin ladattavaksi editoriin. Lohko hyödyntäisi asiakkaan puolella renderöintiä, ja käytännössä lopullisella sivulla näytettävä tuloste tulisi tietokannasta lohkon rekisteröivän JavaScript-tiedoston muodostaman tallenteen perusteella. Yksinkertaisimmillaan lohko tarvitsee vain nimen, editorin tulosteen, joka voi olla vaikkapa vain HTML-merkintää, sekä tallenteen tulosteen, joka niin ikään voi olla yksinkertaisimmillaan HTML-merkintää.

Moderneja verkkokehityskirjastoja on helppo ottaa mukaan kehittäessä sisältöä editorille. Editorissa lohkoa käsitellään JavaScript-ohjelmana, joten sitä voi laajentaa lataamalla JavaScript-kirjastoja ja komponentteja esimerkiksi Node-paketinhallinnan avulla, kunhan kaikki riippuvuudet ladataan editoriin tai pakataan samaan JavaScript-tiedostoon.

Lohkokehitykseen tarkoitettuja komponentteja voidaan hyödyntää joko lataamalla tarvittavat komponentit Node-paketinhallinnan avulla, tai käyttämällä editorin mukana selaimeen ladattavaa globaalia wp-muuttujaa, joka sisältää käytännössä editorin käyttämät komponentit. Nämä komponentit hyödyntävät valmiiksi lohkoille tarkoitettuja rajapintoja ja ominaisuuksia. Toisaalta lohkot voivat käyttää WordPressin eri rajapintoja kuten WP_Query-hakuja kutsumalla AJAX- tai REST API-rajapintoja itse.

Osaa ideoistani lohkoille en päässyt tämän opinnäytetyön puitteissa kokeilemaan, jotta sen laajuus ei pääsisi liiaksi paisumaan. Esimerkiksi ulkoisten rajapintojen, oman rajapinnan sekä WordPress Transient -välimuistituksen olisi voinut koostaa yhteen vaikkapa sääennustelohkon muodossa. Yksi mielenkiintoinen aihe olisi ollut JavaScript-kirjastojen hyödyntäminen sivun tulosteen puolella.

Tällainen voisi olla esimerkiksi edistynyt artikkelinostolohko, jossa kirjautumaton lukija voi selata eri sisältöjä muuttamalla suodattimia tai asetuksia. Tällaisten monimutkaisempien lohkojen kehittäminen olisi ollut mielenkiintoista, mutta ei alustuksineen olisi mahtunut saman opinnäytetyön piiriin.

Ammattimaiseen ympäristöön tehdessä tulisi todennäköisesti huomioida sisällön kääntämisen tuki esimeriksi WordPressin I18n -kirjaston avulla. Käännöstuki mahdollistaisi lisäosan käyttämisen useammilla eri kielillä, ja kääntämistuki on yksi WordPressin kilpailukykyisimpiä ominaisuuksia. Lisäksi käyttäjien tekemät syötteet kannattaa varmuuden vuoksi tuotannossa validoida tarkemmin. Se on erityisen tärkeää, jos syötteitä voi tehdä kirjautumattoman loppukäyttäjän toimesta, mutta on hyvä validoida myös erilaisten ylläpitäjäkäyttäjien syötteet varmuuden vuoksi. Muuten sisältölohkoja voisi kehittää lisäämällä erilaisia parametreja nostolohkon dynaamisesti haettavan tiedon suodattamiseen, sekä molempien lohkojen osalta erilaisten asetteluiden ja ulkoasuvalintojen mahdollistamiseksi.

Opinnäytetyön ulkopuolelle jäi laajuuden kohtuullistamiseksi tarkastelu siitä missä tilanteissa tulisi suosia serverin-, ja missä asiakkaan puolella tapahtuvaa renderöintiä. Metodien performanssieroihin tai tietoturvaeroihin ei otettu myöskään kantaa. Molempien tarkastelussa olisi kuitenkin ollut hyvää pohjaa tarkemmin rajattuina erillisinä tutkimuksina.

Kokonaisuutena voin katsoa opinnäytetyön saavuttaneen sille asetetut tavoitteet onnistuttuani luomaan valitsemillani tekniikoilla toimivat ja mihin tahansa moderniin WordPress-asennukseen helposti lisättävät sisältölohkot. Sisältölohkot ovat ylläpitäjän näkökulmasta selkeitä kokonaisuuksia, jotka toimivat yhtenäisesti WordPressin asennuksen mukana tulevien lohkojen tapaan. Julkaistulle sivustolle tulostuva sisältö toimii suorituskykyisesti ja luotettavasti ylläpitäjän asettamien asetusten mukaan.

Opinnäytetyön tekeminen sujui hyvin, joskin sen suorittaminen kokopäivätyön ohella kesti kauemmin kuin olin alun perin ajatellut. Työ on kuitenkin edennyt johdonmukaisesti ja pysynyt koko tutkimusprosessin ajan rakenteeltaan pääosin samanlaisena, toki alaotsikoiden ja rakenteen hieman eläessä tutkimuksen edetessä ja aiheeseen liittyvän ymmärryksen kasvaessa.

Opin opinnäytetyötä tehdessäni lohkoeditorin sekä erityisesti erilaisten sisältölohkojen toiminnan syvemmin kuin olisin ehtinyt päivätyössäni verkkokehittäjänä perehtymään. Olen jo nyt hyödyntänyt tutkimuksen kautta tulleita oppeja kehittäessäni aiempaa monimutkaisempia ja kehittyneempiä sisältölohkoja asiakasprojekteihin. Tutkimuksen myötä kertyneen ymmärryksen avulla olen myös pystynyt ottamaan perustellusti kantaa lohkokehityksen menetelmiin sekä teknologiavalintoihin. Konkreettisesti olen tutkimuksen aikana oppinut itselleni aiemmin tuntemattomia rajapintoja, kuten lohkotyylit, ja toisaalta löytänyt testaamalla ja lähdekoodia selaamalla toimintoja, joita en ollut

löytänyt dokumentaatiosta. Tällainen oivallus oli esimerkiksi lohkotukien välittäminen palvelimen puolella tulostavalle lohkolle lisäämällä tukien mukaan nimetyt attribuutit lohkon rekisteröintiin. Lohkotukien muodostamat käyttöliittymät päivittävät saman nimisiä attribuutteja automaattisesti, jolloin ne voi välittää tulosteen koostavaan funktioon. Asiakkaan puolella tulostavissa lohkoissa näin ei tapahdu, sillä lohkotuet muuttavat tuolloin tulostetta suoraan ilman että attribuutteja tarvitsee erikseen rekisteröidä.

Lähteet

A white pixel 2020. How to add custom block styles to WordPress Gutenberg blocks. Luettavissa: https://awhitepixel.com/blog/wordpress-gutenberg-custom-block-styles/. Luettu: 20.2.2022.

Corvalan, D., Sengupta, D. & Singhal, M. 2016. Getting started with React. Packt Publishing. Birmingham.

Cosper, J. 20.9.2022. How to Create a WordPress Child Theme (Tutorial). DreamHost. Luettavissa: https://www.dreamhost.com/blog/install-wordpress-child-theme/. Luettu: 24.11.2022.

Dessign 2022. Gutenberg (block editor) vs classic editor review, which one is better option? Luettavissa: https://dessign.net/gutenberg-vs-classic-editor/. Luettu: 20.2.2022.

Fullhost 2022. Apache vs Nginx: Which type of server should you use for WordPress? Luettavissa: https://www.fullhost.com/blog/apache-vs-nginx-which-type-of-server-should-you-use-for-wordpress/. Luettu: 23.11.2022.

Gordon, Z. 2017. How to use registerBlockType() to create blocks in WordPress. Luettavissa: https://wp.zacgordon.com/2017/12/28/how-to-use-to-registerblocktype-to-create-blocks-in-wordpress/. Luettu: 13.3.2022.

Gutenberg Hub s.a. a. How to apply custom CSS to any Gutenberg Block. Luettavissa: Luettavissa: https://gutenberghub.com/how-to-apply-custom-css-to-any-gutenberg-block/. Luettu: 20.2.2022.

Gutenberg Hub s.a. b. How to Create a Custom Blocks Category for WordPress (Gutenberg). Luettavissa: Luettavissa: https://gutenberghub.com/how-to-create-custom-block-category/. Luettu: 24.11.2022.

Hostinger 2022. What Is npm? A Basic Introduction to Node Package Manager for Beginners. Luettavissa: https://www.hostinger.com/tutorials/what-is-npm. Luettu: 23.11.2022.

JavaScript for WP 2019. InspectorControls – how to add settings to custom Gutenberg blocks. Luettavissa: https://javascriptforwp.com/how-to-use-inspectorcontrols/. Luettu: 22.11.2022.

Kinsta 2021. The ultimate guide to WordPress shortcodes (with examples to create your own). Luettavissa: https://kinsta.com/blog/wordpress-shortcodes/. Luettu: 22.2.2022.

Kinsta 2022a. Diving deep into the latest Gutenberg WordPress editor (2022). Luettavissa: https://kinsta.com/blog/gutenberg-wordpress-editor/. Luettu: 9.11.2022.

Kinsta 2022b. A Beginner’s Guide to WordPress Database: What It Is and How to Access It. Luettavissa: https://kinsta.com/knowledgebase/wordpress-database/. Luettu: 23.11.2022.

Lefebre, Y. 2022. WordPress plugin development cookbook. 3. painos. Packt Publishing. Birmingham.

Noble Desktop. 29.4.2022. What is the React Framework and why is it so popular? Luettavissa: https://www.nobledesktop.com/classes-near-me/blog/what-is-react. Luettu: 28.3.2023.

Nymark, C. 2022. Custom block styles. Full Site Editing. Luettavissa: https://fullsiteediting.com/lessons/custom-block-styles/. Luettu: 22.11.2022.

Sears, D. 2021. WordPress extending classic editor support until 2022. Bluehost Blog. Luettavissa: https://www.bluehost.com/blog/wordpress-extending-classic-editor-support-until-2022/. Luettu: 20.2.2022.

Stoyan, S. 2015. React up & running. O’Reilly Media Inc. Sebastopol.

Tadlock, J. 11.1.2022. What Are Block Themes? What you need to know before WordPress 5.9. WP Tavern. Luettavissa: https://wptavern.com/what-are-block-themes-what-you-need-to-know-before-wordpress-5-9. Luettu: 23.11.2022.

Tuca, A. 13.1.2023. WordPress Block Editor: The Ultimate Gutenberg Guide (2023). SmartBlogger. Luettavissa: Luettavissa: https://smartblogger.com/wordpress-block-editor/. Luettu: 23.3.2023.

Ultimate Blocks. How to add anchor links in WordPress (Gutenberg). Luettavissa: https://ultimateblocks.com/anchor-links-wordpress/. Luettu: 20.2.2022.

Watson, M. 2021. Using a template with innerBlocks in the WordPress Block Editor (Gutenberg). Wholesome Code. Luettavissa: https://wholesomecode.ltd/using-a-template-with-innerblocks-in-the-wordpress-block-editor-gutenberg. Luettu: 22.11.2022.

WordPress VIP. How to Enhance WordPress Blocks With Block.json and Server-side Registration. Luettavissa: https://wpvip.com/2023/02/08/block-json-server-side-registration/. Luettu: 28.3.2023.

WordPress.org s.a. a. Block Editor Handbook. Luettavissa: https://developer.wordpress.org/block-editor/. Luettu: 20.2.2022.

WordPress.org s.a. b. Say hello to the new editor. Luettavissa: https://wordpress.org/gutenberg/. Luettu: 20.2.2022.

WordPress.org s.a. c. WordPress Editor. Luettavissa: https://wordpress.org/support/article/wordpress-editor/. Luettu: 20.2.2022.

WordPress.org s.a. d. Key Concepts. Luettavissa: https://developer.wordpress.org/block-editor/explanations/architecture/key-concepts/. Luettu: 20.2.2022.

WordPress.org s.a. e. Registration. Luettavissa: https://developer.wordpress.org/block-editor/reference-guides/block-api/block-registration/. Luettu: 13.3.2022.

WordPress.org s.a. f. Settings sidebar. Luettavissa: https://wordpress.org/support/article/settings-sidebar/. Luettu: 22.11.2022.

WordPress.org s.a. g. WordPress Block Editor. Luettavissa: https://wordpress.org/support/article/wordpress-editor/. Luettu: 22.11.2022.

WordPress.org s.a. h. Hooks. Luettavissa: https://developer.wordpress.org/plugins/hooks/. Luettu: 23.3.2023.

WordPress.org s.a. i. Page/Post Settings sidebar. Luettavissa: https://wordpress.org/documentation/article/page-post-settings-sidebar/. Luettu: 28.3.2023.

WordPress.org s.a. j. Page templates. Luettavissa: https://developer.wordpress.org/themes/template-files-section/page-template-files/. Luettu: 28.3.2023.

WordPress.org s.a. k. Block templates. Luettavissa: https://developer.wordpress.org/block-editor/reference-guides/block-api/block-templates/. Luettu: 28.3.2023.

WordPress.org s.a. l. Block supports. Luettavissa: https://developer.wordpress.org/block-editor/reference-guides/block-api/block-supports/. Luettu: 28.3.2023.

WordPress.org s.a. m. Package Reference. Luettavissa: https://developer.wordpress.org/block-editor/reference-guides/packages/. Luettu: 28.3.2023.

WordPress.org s.a. n. Roadmap. Luettavissa: https://wordpress.org/about/roadmap/. Luettu: 28.3.2023.

WP Beginner 2022. How to disable Gutenberg and keep the classic editor in WordPress. Luettavissa: https://www.wpbeginner.com/plugins/how-to-disable-gutenberg-and-keep-the-classic-editor-in-wordpress/. Luettu: 20.2.2022.

WP Beginner 2023. How to use the new WordPress block editor. Luettavissa: https://www.wpbeginner.com/beginners-guide/how-to-use-the-new-wordpress-block-editor/. Luettu: 23.3.2023.

Liitteet

Liite 1. Käsitteet

CMS, Content management system. Sisällönhallintajärjestelmät, kuten WordPress, tarjoavat ylläpitäjille sivuston sisällön muokkaamiseen tarkoitettuja työkaluja. Tämä mahdollistaa sivustojen ylläpitämisen ilman koodiosaamista, yksinkertaisten käyttöliittymien avulla.

CSR, Client-side rendering – Asiakkaan puolella renderöinnillä tarkoitetaan sitä, että sivuston avaavan lukijan tietokone laskee tietokannasta haetun sisältödatan esitystavan JavaScript-koodin avulla.

CSS, Cascading stylesheets – Verkkosisältöjen ulkoasun muokkaamisen tarkoitettu merkintäkieli, joka kertoo selaimelle, miten sivuston eri elementit tulee esittää.

DOM, Document object model – Dokumenttiobjektimalli on verkkosivun elementeistä muodostettu rajapinta, joka kuvaa verkkosivun sisältöä. Mahdollistaa sivuston sisällön muokkaamisen JavaScriptin avulla.

Gutenberg-editori – Sisältölohkoihin perustuva editori tunnetaan projektinimellä Gutenberg. Lohkoeditori mahdollistaa monipuolisten sisältöjen lisäämisen sivustolle erilaisten sisältölohkojen avulla.

HTML, Hypertext markup language – Hypertekstin merkintäkieli, jolla verkkosivut esitetään selaimen tulkittavaksi. Yksi verkkokehittämisen perusteknologioista.

JavaScript – Ohjelmointikieli, joka mahdollistaa monipuolisten käyttöliittymien ja toiminnallisuuksien luomisen verkkoselaimilla käytettäviin verkkosisältöihin.

JSX – Syntaksilaajennus JavaScriptiin, joka mahdollistaa HTML-tyyppisen merkintätavan käyttämisen JavaScript ohjelmakoodin joukossa.

Lukija – Sivuston sisältöjä selaava ihminen, joka ei välttämättä omaa käyttäjätunnuksia sivustolle. Sivuston sisällöt luodaan lukijoiden, eli sivuston loppuasiakkaiden, tarkasteltavaksi.

React – JavaScript ohjelmakirjasto, joka tarjoaa monipuolisia työkaluja reaktiivisten tilapohjaisten komponenttien ja käyttöliittymien luomiseen.

SSR, Server-Side Rendering – Palvelimen puolella tapahtuvalla renderöinnillä tarkoitetaan sitä, että palvelin välittää valmiiksi HTML-syntaksiin muutettua merkintää lukijan laitteen näytettäväksi.

Sisältölohko – Sisältölohkot ovat editorissa asetettavia ohjelmakomponentteja, jotka mahdollistavat erilaisten sisältöjen näyttämisen sivustolla.

Sivuston ylläpitäjä – Sivuston hallintajärjestelmän käyttöoikeudet omaava kirjautunut käyttäjä, joka ylläpitää sivuston sisältöjä sekä asetuksia hallinnollisten käyttöliittymien avulla.

Verkkokehittäjä – Ohjelmakoodia muuttava henkilö, jolla on tyypillisesti oikeudet sekä sivuston hallintanäkymiin että sen lähdekoodiin. Verkkokehittäjä laajentaa ja muuttaa sivuston toimintaa ohjelmakoodia muuttamalla.

Virtual DOM – Reactin erityisominaisuus, jossa dokumenttiin tehdyt muutokset käsitellään virtualisoidun DOM-objektin kautta. Vertaamalla virtuaalista ja todellista DOM-näkymää React ohjelmakoodi optimoi elementtien päivityksen suorituskykyä.

WordPress on avoimella lähdekoodilla toteutettu sisällönhallintajärjestelmä, jonka toimintojen laajentamiseen tämä opinnäytetyö keskittyy.

WordPress Core – WordPressin kehittäjäyhteisön julkaisemaan ohjelmakoodiin viitataan yleisesti corena, eli ytimenä. Core sisältää ylläpitonäkymien ohella WordPressin toiminnallisen ohjelmakoodin, jota voidaan jatkaa teemojen ja lisäosien avulla.

WordPress Customizer – WordPressin teemojen asetuksia muutetaan perinteisemmissä teemoissa customizer valikon avulla. Uudemmissa lohkoteemoissa Customizeria ei enää ole, vaan se on korvattu sivustoeditorilla.

WordPress Hook API – WordPress perustuu koukkujärjestelmään, jossa ohjelmakoodin eri suoritusvaiheisiin voidaan välittää joko funktioita tai filttereitä. Tarkoituksena on muuttaa ohjelman toimintaa, tai muokata erilaisia asetuksia ohjelman suorituksen aikana.

WordPress Plugin – Lisäosilla sivustolle voidaan tuoda erilaisia ominaisuuksia. Toisin kuin teemoja, lisäosia voi olla käytössä useita samanaikaisesti.

WordPress Theme – Teemat ovat ohjelmakokonaisuuksia, joiden perustavoite on luoda sivustolle sivupohjat sekä tyylit, joiden avulla sivuston sisältö esitetään. Monesti teemat kuitenkin sisältävät lisäksi monipuolista toiminnallista koodia.

WYSIWYG, “What you see is what you get” – Lyhenteellä tarkoitetaan sitä, että esikatselu tai muu ensivaikutelma vastaa lopullista tuotosta. Verkkokehityksessä WYSIWYG-termiä käytetään kuvaamaan sisältöeditorien muokkausalueiden vastaavan lopullista sivustolla näytettävää tulostetta.