LAB University of Applied Sciences

Kyselytyökalun suunnittelu ja kehitys sisäl-lönhallintajärjestelmään

Case: Tammi Digital Oy

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
2022
Eetu Andronoff

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Andronoff, Eetu	Opinnäytetyö, AMK	2022
	Sivumäärä 49	

Työn nimi

Kyselytyökalun suunnittelu ja kehitys sisällönhallintajärjestelmään

Case: Tammi Digital Oy

Tutkinto

Insinööri, Tieto- ja viestintätekniikka (AMK)

Toimeksiantajan nimi, titteli ja organisaatio

Antti Immonen, Teknologiajohtaja, Tammi Digital Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja kehittää WordPress-lisäosa, jolla voitaisiin helposti lisätä monivalintakyselyitä WordPress-sivustoille sekä kerätä dataa niiden kautta. Työkalun suunnittelussa ja kehityksessä tuli ottaa huomioon sekä käyttäjän, ylläpitäjän että kehittäjän näkökulmat. Työn toimeksiantaja on lahtelainen ohjelmistoyritys Tammi Digital Oy.

WordPress on maailman suosituin WWW-sisällönhallintajärjestelmä. Se on kirjoitettu PHP-ohjelmointikielellä ja hyödyntää joko MySQL- tai MariaDB-tietokantaa. WordPress-lisäosat ovat koodipaketteja, joilla pystytään lisäämään toiminnallisuuksia WordPressiin.

Ensimmäisenä lisäosalle suunniteltiin pääpiirteet. Näille pääpiirteille annettiin prioriteetit, joiden pohjalta loput suunnitteluvaiheesta toteutettiin. Kehitysvaiheessa toteutettiin ensimmäiseksi kyselyiden hallintakäyttölittymä Advanced Custom Fields -lisäosaa hyödyntämällä. Kyselyn esittämiseksi sivulla luotiin funktio, jota voitaisiin kutsua sivun sisällöstä lyhytkoodin avulla. Kyselyt kehitettiin tallentamaan dataa käyttäjän valinnoista välittömästi hyödyntäen AJAXia. Luotiin myös tapa ladata kyselyiden keräämä data CSV-tiedostomuodossa.

Lisäosa saatiin suunniteltua ja kehitettyä toimeksiannon ja WordPress-lisäosan kehityksen parhaiden käytänteiden mukaisesti. Lisäosassa otettiin erityisesti huomioon sivuston loppukäyttäjän sekä ylläpitäjän näkökulmat.

Asiasanat

WordPress, WordPress-lisäosa, AJAX

Abstract

Author(s)	Type of Publication	Published
Andronoff, Eetu	Thesis, UAS	2022
	Number of Pages	
	49	

Title of Publication

Designing and developing a multiple choice quiz tool for a CMS

Case: Tammi Digital Oy

Name of Degree

Bachelor of Engineering, Information and Communications Technology

Name, title and organization of the client

Antti Immonen, Chief Technology Officer, Tammi Digital Oy

Abstract

The goal of the thesis was to design and develop a WordPress plugin for creating multiple choice quizzes for WordPress sites. The plugin's purpose was to make it possible to easily add quizzes on WordPress sites and collect data from them. The plugin was to be designed and developed with the user's, the administrator's, and the developer's points of view in mind. The plugin was made for Tammi Digital Oy, a software company from Lahti.

WordPress is the world's most popular content management system. WordPress is built with PHP and utilizes either a MySQL or a MariaDB database. WordPress plugins are packages of code that add additional functionality to WordPress.

The main features of the plugin were first designed and prioritized. The rest of the design process was done according to this prioritization. The administration interface for the quizzes was developed first utilizing the Advanced Custom Fields plugin. A function that can be called via a shortcode in the page content was created to show quizzes on pages. The quizzes were developed to save data from a user's decisions immediately by using AJAX. A way to download the collected data as a CSV-file was also added.

The plugin was designed and developed according to the client's requirements and WordPress plugin development's best practices. The user's and the administrator's points of view were especially paid attention to.

Keywords

WordPress, WordPress plugin, AJAX

Sisällys

1 Johdanto

WordPress on maailman suosituin WWW-sisällönhallintajärjestelmä. W3Techsin mukaan maaliskuussa 2022 sitä käyttää 43,3 % kaikista verkkosivuista ja 65,1 % verkkosivuista, joilla on käytössään jokin sisällönhallintajärjestelmä. Miljoonat käyttäjät käyttävät WordPressiä verkkosivujensa rakentamiseen ja ylläpitoon. (W3Techs.)

WordPress tarjoaa käyttäjilleen valmiita lohkoja, joilla käyttäjä voi rakentaa oman sivustonsa, sekä teemoja, joilla käyttäjä voi tyylitellä sivuston oman näköisekseen. WordPressiin löytyy myös tuhansia lisäosia, joilla käyttäjä pystyy lisäämään sivustolleen ominaisuuksia, joita WordPressin ydin ei tarjoa. (WordPress.com.)

Opinnäytetyön toimeksiantaja on lahtelainen ohjelmistoyritys Tammi Digital Oy. Yrityksen toimiala on verkkosivustojen kehitys- ja ylläpitopalvelut sekä ohjelmistokehitys.

Työn tavoitteena on suunnitella ja kehittää työkalu monivalintakyselyiden luomiseen WordPress-sivustoille. Työn tarkoituksena on mahdollistaa monivalintakyselyiden helppo lisääminen toimeksiantajan kehittämille verkkosivustoille. Monivalintakyselyiden tarkoitus on parantaa sivuston käyttäjien käyttökokemusta sekä antaa sivuston ylläpitäjälle samalla tietoa käyttäjistään.

Työkalulla tulee pystyä luomaan, muokkaamaan ja esittämään monivalintakyselyitä sivustolla. Työkalulla luotujen monivalintakyselyiden tulee myös kerätä käyttäjien tekemistä valinnoista tietoa, jota sivuston ylläpitäjä pystyy myöhemmin tarkastelemaan ja analysoimaan. Työkalu toteutetaan WordPress-lisäosana, jonka työnimeksi annetaan Tammi Textchat.

Työkalun suunnittelussa ja kehityksessä otetaan huomion sekä kehittäjän, ylläpitäjän että käyttäjän näkökulmat. Lisäksi työkalun kehityksessä otetaan huomioon WordPress-lisäosan kehityksen parhaat käytänteet sekä tietoturvallisuus.

2 WordPress-sisällönhallintajärjestelmä

2.1 Yleistä WordPressistä

WordPress on PHP-ohjelmointikielellä kirjoitettu ja joko MySQL- tai MariaDB-tietokantaa hyödyntävä avoimen lähdekoodin WWW-sisällönhallintajärjestelmäohjelmisto (WordPress.org p). WordPress käyttää General Public License -lisenssiä, jonka mukaan ohjelmistoa saa vapaasti ajaa, tutkia, muokata, jakaa uudelleen sekä jakaa muokattuna (WordPress.org a).

WordPressin perustajat Matt Mullenweg ja Mike Little julkaisivat WordPressin toukokuussa 2003 (Mullenweg 2003). WordPressiä alettiin alun perin kehittämään käyttäen pohjana avoimen lähdekoodin blogiohjelmistoa b2/cafelog (Warner 2009).

2.2 WordPress-lisäosat

WordPress-lisäosat ovat koodipaketteja, joilla lisätään toiminnallisuuksia WordPressiin (WordPress.org u). WordPressiin löytyy maaliskuussa 2022 jo yli 59 000 lisäosaa, jotka sivulleen voi ladata, mutta lisäosia on mahdollista myös kehittää itse (WordPress.org k).

Halutut lisäominaisuudet lisätään WordPressiin lisäosien kautta, ettei muokkauksia tehtäisi WordPressin ytimeen. WordPressin ydin ylikirjoitetaan jokaisen päivityksen yhteydessä, milloin kaikki siihen tehdyt muokkaukset myös ylikirjoittuvat, minkä takia kaikki tehdyt muutokset jouduttaisiin tekemään uudelleen. (WordPress.org i.)

2.2.1 Rakenne

Lisäosat rakennetaan PHP-ohjelmointikielellä, mutta niihin voidaan myös liittää muita tiedostoja kuten kuva-, CSS- tai JavaScript-tiedostoja. Yksinkertaisimmillaan lisäosa on vain yksittäinen PHP-tiedosto. (WordPress.org u.)

Lisäosan tiedostot tulee sijoittaa WordPressin plugins-kansion alle. Lisäosalle suositellaan luomaan myös oma alikansionsa, jotta kaikki lisäosan tiedostot löytyvät helposti yhdestä paikasta. (WordPress.org n.)

Lisäosan PHP-ohjelmointikielellä kirjoitetun päätiedoston tulee sisältää niin sanottu DocBlock-osio tiedoston ylätunnisteessa, joka sisältää tietoa lisäosasta (Kuva 1). DocBlock-osion tulee sisältää vähintään lisäosan nimi. (WordPress.org u.)

```
 6   /*
 7   Plugin Name: Hello Dolly
 8   Plugin URI: http://wordpress.org/plugins/hello-dolly/
 9   Description: This is not just a plugin, it symbolizes
10   Author: Matt Mullenweg
11   Version: 1.7.2
12   Author URI: http://ma.tt/
13   */
```

Kuva 1. WordPressin Hello Dolly -lisäosan DocBlock-osio

Lisäosat tarvitsevat toimiakseen myös PHP-funktioita (WordPress.org i). Funktioita luodessa tulee ottaa huomioon funktioiden mahdolliset nimikonfliktit. Koska kaikki WordPress-sivun lisäosat kommunikoivat WordPressin ytimen kanssa, mitkään lisäosat eivät saisi sisältää samannimisiä funktioita. Tästä syystä yksi parhaista käytänteistä WordPress-lisäosan kehityksessä on nimetä funktiot lisäosan omalla etuliitteellä, kuten lisäosan nimellä (Kuva 2). Samaa käytännettä tulisi käyttää myös muuttujille, ettei niidenkään välille tule nimikonflikteja. (WordPress.org d.)

```
149  function etuliite_funktio( $parametri ) {
150      return "Palautetaan " + $parametri;
151  }
```

Kuva 2. PHP-funktio etuliite_funktio etuliitteellä, parametrillä ja palautuksella

Lisäosat ajavat näitä funktioita WordPressin eri osissa vaikuttaakseen WordPressin toimintaan. Funktiot pystytään liittämään WordPressin ytimen eri osiin käyttämällä koukkuja. (WordPress.org n.)

2.2.2 Koukut

Koukut (Hook) ovat tapa, jonka avulla koodissa pystytään vuorovaikuttamaan tai tekemään muutoksia toiseen osaan koodia ennalta määritetyissä pisteissä. Koukkujen avulla WordPress-lisäosat pystyvät vaikuttamaan WordPressin ytimeen muokkaamatta ytimen koodia. Käyttääkseen WordPressin koukkuja, tulee kirjoittaa funktio ja rekisteröidä se

halumaansa WordPress-koukkuun. Koukut jaetaan kahteen ryhmään: toimintoihin ja suodattimiin. (WordPress.org h.)

Toiminnot ovat koukkuja, jotka vastaanottavat tietoa ja tekevät jonkin toiminnon, mutta eivät palauta mitään. Toiminnot siis pysäyttävät koodin etenemisen tehdäkseen oman toimintonsa, jonka jälkeen koodi jatkaa etenemistä normaaliin tapaan (Kuva 3). (WordPress.org b.)

```
149  function etuliite_toiminto() {
150      // funktion toiminto
151  }
152  add_action( 'init', 'etuliite_toiminto' );
```

Kuva 3. etuliite_toiminto-funktio ja sen liittäminen init-koukkuun

Suodattimet ovat koukkuja, jotka vastaanottavat tietoa ja muokkaavat sitä jollain tavalla, jonka jälkeen ne palauttavat sen takaisin koukulle. Suodattimet siis muokkaavat jotain tietoa, mitä tullaan vielä käsittelemään koodissa myöhemmin (Kuva 4). (WordPress.org g.)

```
149  function etuliite_otsikko_suodatin( $title ) {
150      return 'Otsikko ' . $title . ' suodatettiin';
151  }
152  add_filter( 'the_title', 'etuliite_suodatin_otsikko' );
```

Kuva 4. etuliite_otsikko_suodatin-funktio liitetty suodattimena the_title-koukkuun

2.2.3 Artikkelien käsittely

WordPress-sivuston sisältö koostuu artikkeleista. WordPress jakaa toisistaan eroavan sisällön erilaisiin artikkelityyppeihin. WordPress luo oletusarvoisesti asennuksen yhteydessä seitsemän erilaista artikkelityyppiä: artikkelit, sivut, liitteet, muutokset, valikot, mukautettu CSS sekä muutossarjat. WordPressiin on myös mahdollista lisätä mukautettuja artikkelityyppejä käyttäjän toimesta. (WordPress.org o.)

Mukautettuja artikkelityyppejä voidaan tarpeen mukaan luoda käyttämällä register_post_type-funktiota. Mukautetut artikkelityypit suositellaan aina luotavaksi varmuuden vuoksi WordPress-lisäosan avulla, ettei mukautettuja artikkelityyppejä menetetä esimerkiksi sivuston teeman vaihtuessa. (WordPress.org o.)

2.2.4 Lyhytkoodit

PHP-koodin ajaminen WordPressin sisällön seassa on estetty turvallisuussyistä. Tästä syystä WordPressin versiossa 2.5 esitettiin lyhytkoodit. Lyhytkoodit ovat makroja, joilla voidaan ajaa PHP-koodia sisällön seassa siististi ja turvallisesti. Lyhytkoodeja pystyy upottamaan sivun sisällön sekaan WordPressin käyttöliittymän kautta (Kuva 5) tai PHP-koodissa. WordPressillä on muutama sisäänrakennettu lyhytkoodi, mutta niitä voidaan myös lisätä itse hyödyntämällä WordPressin tarjoamaa add_shortcode-funktiota. Lyhytkoodeille pystytään lisäämään myös parametrejä. (WordPress.org t.)

Kuva 5. Lyhytkoodi-lohko WordPress-sivun muokkaustilassa

Lyhytkoodit ovat käytännössä suodattimia WordPressin sisällössä, joten niiden tulisi parhaiden käytänteiden mukaan aina palauttaa jotain. Kuten funktioille ja muuttujillekin, lyhytkoodeille tulisi myös asettaa etuliite välttääkseen nimeämiskonflikteja muiden lyhytkoodien kanssa. Lyhytkoodin syötteet tulisi myös aina turvallisuuden vuoksi puhdistaa. (WordPress.org t.)

2.2.5 Lisäosan asetukset

Lisäosille pystytään luomaan käyttöliittymän kautta muokattavia asetuksia. Näiden käyttöliittymien ja asetusten luomiseen WordPress tarjoaa kaksi eri rajapintaa: Settings APIn sekä Options APIn. (WordPress.org r.)

Settings API tarjoaa tavan luoda uusia ylläpitosivuja ja niille asetuslomakkeita, sekä käsitellä näihin lomakkeisiin syötettävää tietoa. Lomakkeille voidaan määrittää erillisiä osioita ja

osioille erillisiä kenttiä. Uusia osioita tai kenttiä voidaan myös lisätä olemassa oleville asetussivuille. WordPressiin pystyy kehittämään asetuksia myös ilman Settings APIa, mutta WordPress suosittelee sitä käytettäväksi useiden hyötyjen takia:

- Settings API varmistaa, että luodut käyttöliittymäelementit ovat ulkonäöltään yhteneväisiä muiden WordPressin asetusten kanssa.
- Settings APIlla kehitetyt asetussivut ovat kestäviä, koska API päivittyy WordPressin ytimen mukana.
- Settings API nopeuttaa kehittämistä tarjoamalla useita työkaluja asetusten luomiseen.

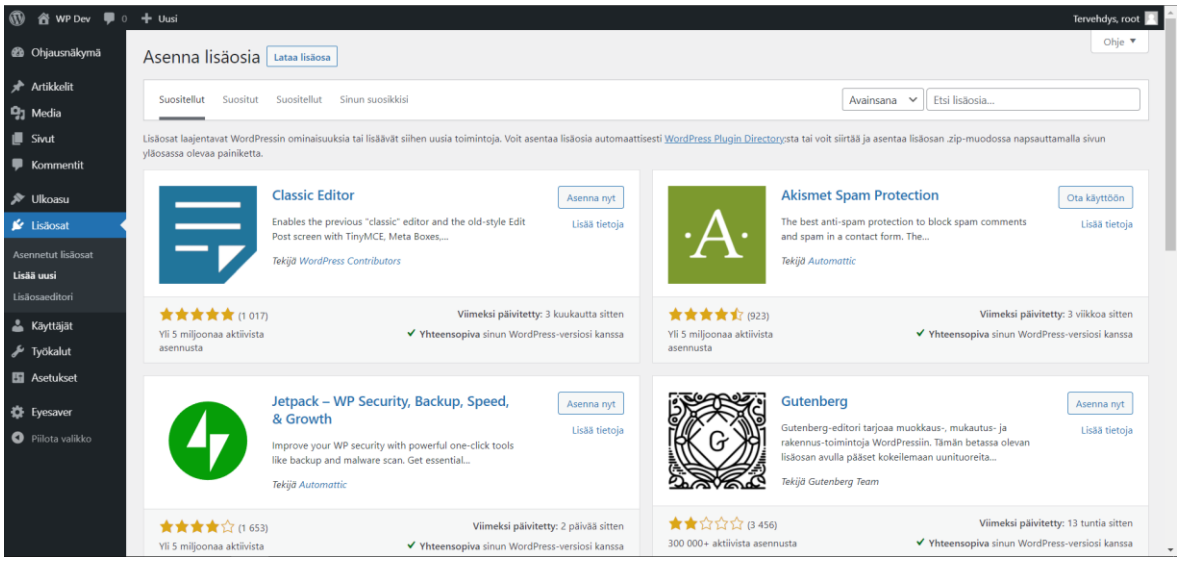
(WordPress.org s.)

Options API tarjoaa tavan luoda, lukea, päivittää ja poistaa WordPressin asetuksia. Yhdessä Settings APIn kanssa, sillä voidaan hallita Settings APIn avulla luoduille asetussivuille määritettyjä asetuksia. Asetukset tallennetaan WordPressin tietokantaan options-tauluun. (WordPress.org m.)

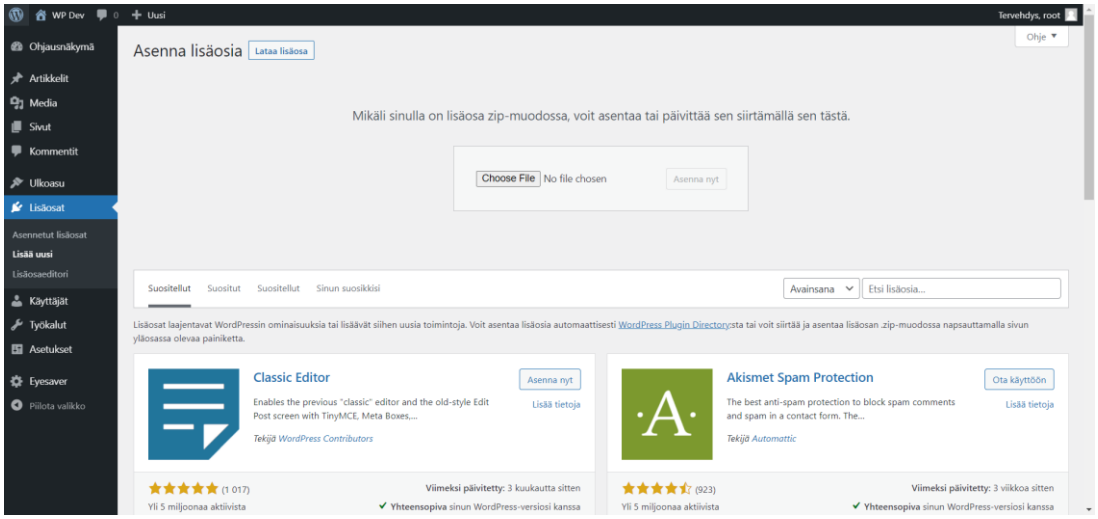
2.2.6 Lisäosan asentaminen

Lisäosia pystyy asentamaan WordPress-sivustolleen kahdella tapaa. Valmiita lisäosia pystyy etsimään ja lataamaan suoraan WordPressin lisäosakirjastosta, mikäli lisäosa on laitettu lisäosakirjastoon saataville. Lisäosia pystyy asentamaan myös lataamalla lisäosan zip-tiedoston sivustolle suoraan tietokoneeltaan. (WordPress.org l.)

Lisäosakirjastoon pääsee käsiksi WordPress-sivuston ylläpitopuolella menemällä osioon Lisäosat ja sen alta osioon Lisää uusi. Tältä sivulta pystytään selaamaan ja asentamaan lisäosia (Kuva 6). Samalta sivulta pystytään myös lataamaan lisäosa sivustolle zip-tiedostosta (Kuva 7).

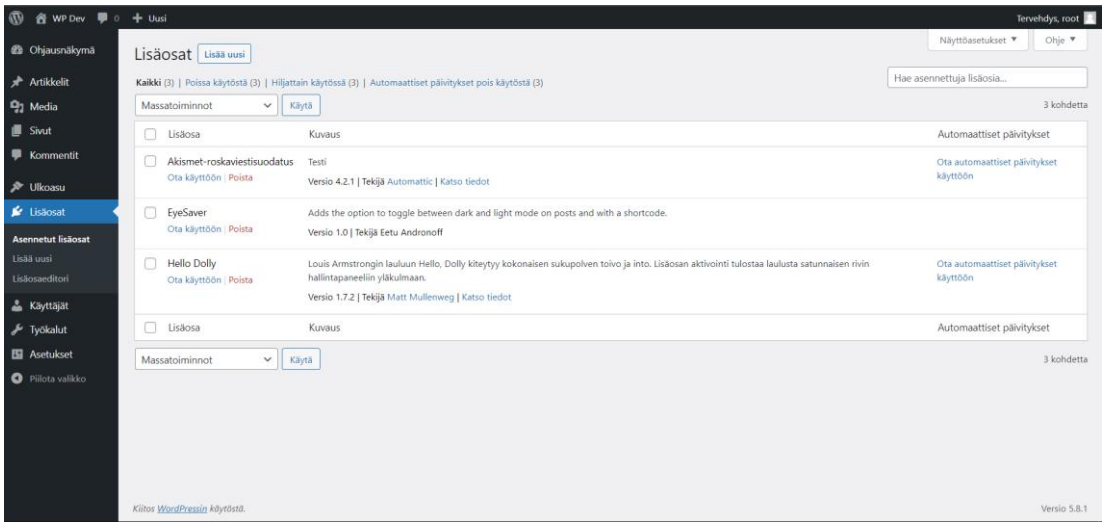


Kuva 6. WordPressin lisäosakirjasto



Kuva 7. Lisäosan zip-tiedoston latausnäkymä

Sivustolle asennetut lisäosat löytyvät Asennetut lisäosat -sivulta (Kuva 8). Asennettuja lisäosia voidaan halutessaan ottaa käyttöön tai poistaa käytöstä poistamatta niitä sivustolta.



Kuva 8. Asennetut lisäosat -sivu

2.3 Advanced Custom Fields -lisäosa

Advanced Custom Fields tai ACF on kehittäjille suunnattu WordPress-lisäosa, joka antaa käyttäjän lisätä sisältökenttiä WordPressin muokkausnäkymiin. Lisäkenttiä pystyy lisäämään nopeasti ja helposti suoraan lisäosan käyttöliittymästä. Kenttiä pystyy lisäämään moniin eri WordPressin osiin: muun muassa artikkeleihin, käyttäjiin, taksonomioihin, mediaan ja kommentteihin. ACF sisältää yli 30 eri kenttätyyppiä ja mahdollistaa käyttäjille myös omien räätälöityjen kenttien luomisen ja lisäämisen. (Delicious Brains.)

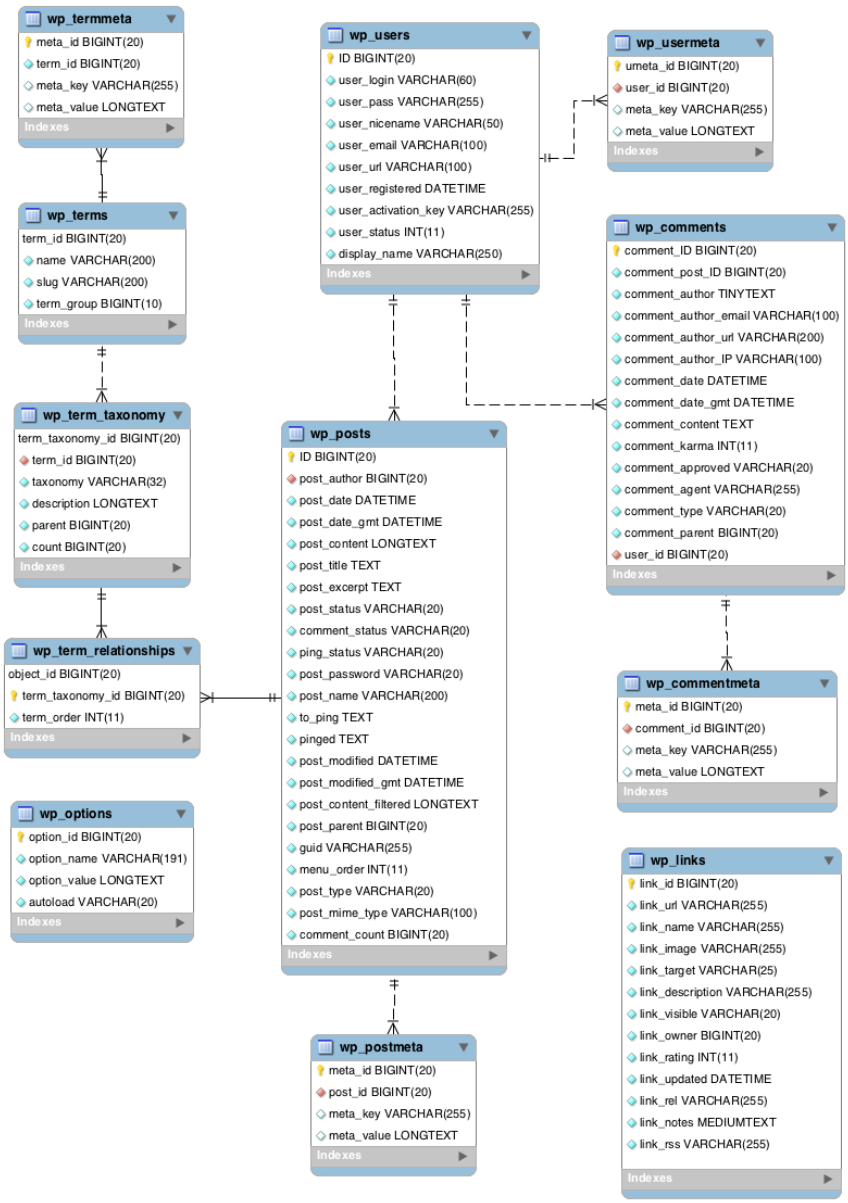
ACF tarjoaa rajapinnan, jonka kautta kehittäjät pystyvät esimerkiksi kutsumaan lisäkenttiin syötettyjä tietoja sivupohjissaan. ACF tarjoaa myös kattavan dokumentaation ja aktiivisen tukifoorumin kehityksen tukemiseen. (Delicious Brains.)

2.4 Tietokanta

WordPress.orgin (WordPress.org j) mukaan WordPressin versio 5.9.1 vaatii toimiakseen vähintään joko MySQL-tietokannasta version 5.7 tai MariaDB-tietokannasta version 10.2. WordPress luo käyttämänsä tietokantataulut asennuksen yhteydessä (WordPress.org f).

Kuviossa 1 on kuvattu WordPressin version 4.4 tietokantatauluja ja niiden yhteyksiä. WordPress luo tietokantataulut artikkeleille, käyttäjille, kommenteille, kategorioille ja tunnisteille, taksonomioille, asetuksille sekä linkeille. Tämän lisäksi WordPress luo taulut metatietoja varten artikkeleille, käyttäjille, kommenteille sekä kategorioille ja tunnisteille.

WordPress luo myös taulun wp_term_relationships tallentamaan yhteyden artikkelien ja taksonomioiden välillä. (WordPress.org j.)



Kuvio 1. WordPressin version 4.4 tietokantadiagrammi (WordPress.org j)

WordPressin loppukäyttäjän ei yleisesti tarvitse välittää WordPressin tietokantarakenteesta, mutta lisäosan kehittäjälle tieto voi olla oleellista. WordPress-lisäosien tarvitsee usein tallentaa tietoa WordPressin tietokantaan. Lisäosien tallentama tieto voidaan jakaa kahteen ryhmään:

- asetustietoihin, jotka käyttäjä usein asettaa lisäosan asennuksen yhteydessä

- dataan, jota lisäosa kerää jatkuvasti käyttäjän käyttäessä lisäosaa.

(WordPress.org e.)

Lisäosa voi luoda uuden tietokantataulun WordPressin tietokantaan automaattisesti, kun lisäosa asennetaan. On kuitenkin suositeltavaa harkita tietojen tallentamista WordPressin ytimen luomiin metatietojen tietokantatauluihin uuden tietokantataulun sijaan, jos se on mahdollista. (WordPress.org e.)

2.5 AJAX

AJAX eli ”Asynchronous Javascript + XML” on yhdistelmä teknologioita, minkä tarkoituksena on tehdä verkkosivusta responsiivisempi käyttäjälle. AJAX mahdollistaa käyttäjälle vuorovaikutuksen sivun kanssa asynkronisesti, erillään kommunikaatiosta palvelimen kanssa. Tämän avulla käyttäjän selain pystyy lähettämään palvelimelle pyyntöjä ja esittämään palvelimelta saatuja vastauksia, ilman että sivua täytyy ladata uudelleen. Tämä uudelleenlataamisen välttäminen tekee sivun käyttökokemusta sulavampaa. (Garrett 2005.)

AJAXin nimessä esiintyvä XML on tiedonvälitysformaatti. Vaikka XML onkin AJAXin perinteinen tiedonvälitysformaatti, tiedonvälitys voidaan kuitenkin suorittaa millä tahansa formaatilla. PHP-koodin kanssa työskennellessä kehittäjät usein suosivat JSON-formaattia, koska sen sisäistä tietorakennetta on helpompi käyttää. (WordPress.org c.)

AJAX ei myöskään vaadi käyttäjältä toimintoja suorittaakseen tehtäviä. Esimerkiksi Google Docs -tekstinkäsittelyohjelma tallentaa muokattavan dokumentin muutaman minuutin välein AJAXin avulla automaattisesti, ilman että käyttäjän tarvitsee aloittaa tallennusprosessia. (WordPress.org c.)

Käyttäjäkokemuksen parantamisen lisäksi AJAX pystyy myös vähentämään huomattavasti siirrettävän datan määrää. Kun sivun sisältöä muutetaan AJAXilla, ainoastaan oleellinen data voidaan siirtää selaimen ja palvelimen välillä sen sijaan, että koko sivun sisältö täytyisi ladata uudelleen. (WordPress.org c.)

2.5.1 AJAX WordPressissä

AJAX on valmiina implementoitu WordPressin ytimeen, sillä WordPress hyödyntää AJAXia toiminnoissaan. Esimerkiksi lisätessä WordPressin ylläpitopuolella sivulle tai artikkelille kategorian tai tunnisteen, sivu päivittyy latautumatta uudelleen AJAXin avulla. (WordPress.org c.)

Jokainen AJAX-toiminto WordPressissä koostuu kahdesta pääkomponentista: selaimessa suoritettavasta jQuery-koodista sekä palvelimella suoritettavasta PHP-koodista. (WordPress.org c.) Palvelimen PHP-koodi tarvitsee kaksi eri osiota AJAXin implementointiin:

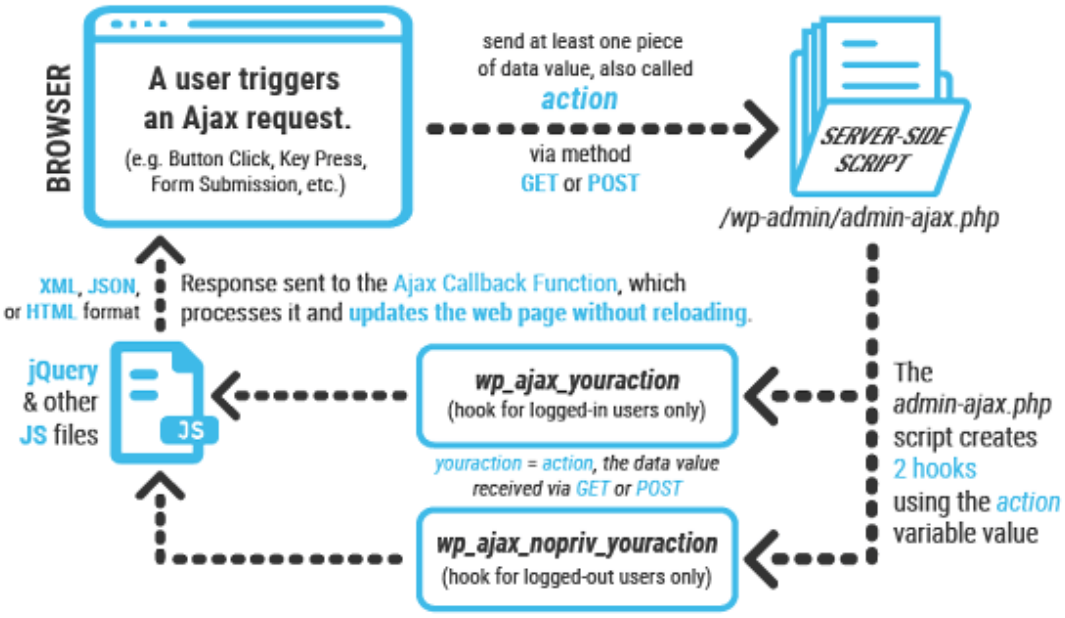
- PHP-koodin tulee ottaa jQuery-koodi käyttöön halutulla sivulla ja välittää sille kaikki arvot, joita jQuery-koodi tarvitsee.
- PHP-koodin tulee pystyä käsittelemään sille välitetyt AJAX-pyynnöt.

(WordPress.org q.)

Kuvassa 9 on kuvattu AJAX-toiminnon elinkaarta WordPressissä. Kaikki WordPressin AJAX-toiminnot noudattavat samaa toimintosarjaa:

- Käyttäjän selain kerää tarvittavat tiedot sivulta ja lähettää ne http-pyynnöllä palvelimelle admin-ajax.php-tiedostoon.
- Palvelimen admin-ajax.php-tiedosto luo pyynnön perusteelta kaksi koukkua pyynnön käsittelyyn.
- Koukkuihin liitetyt funktiot suoritetaan ja niiden palauttama data lähetetään takaisin selaimelle.
- Selain vastaanottaa palvelimen vastauksen ja käsittelee sen halutulla tavalla.

(Ravoof 2022.)



Kuva 9. AJAX-toiminnon elinkaari WordPressissä (Ravoof 2022)

2.5.2 AJAX-toiminnon käsittely

Kaikki WordPressin AJAX-pyynnöt tulee lähettää wp-admin-kansion admin-ajax.php-tiedostoon. Pyynnön kautta palvelimelle lähetettävät tiedot tulee sisällyttää data-taulukkoon. Lisäosan tarvitseman datan lisäksi taulukkoon tulee sisällyttää action-parametri, jolla pyyntö ohjataan palvelimella ajettavaan koodiin. Pyyntöjen, jotka saattavat tehdä muutoksia WordPressin tietokantaan, tulee myös sisältää nonce-tunniste, jotta palvelin tietää, että pyyntö lähetettiin sallitusta lähteestä. (WordPress.org c.)

Nonce eli ”Number used once” on uniikki sarjanumero, joka generoidaan tietylle käyttäjälle tiettyyn kontekstiin, jotta voidaan todentaa tapahtumasta saadun tiedon tulleen oikeasta lähteestä. Nonce-tunnisteita voidaan käyttää muun muassa suojaamaan lomakkeita väärinkäytöltä. (WordPress.org v.)

AJAX-pyynnön jQuery-koodille voidaan tarvittaessa välittää PHP-koodin kautta parametrejä, kuten esimerkiksi admin-ajax.php-tiedoston osoite (WordPress.org c). Kuvassa 10 esitetty AJAX-pyyntö asettaa pyynnön url-parametriksi admin-ajax.php-tiedoston osoitteen sille välitetyn tammi_textchat_js_object.ajaxurl-parametrin kautta sekä security-parametriksi tarvitun nonce-tunnisteen sille välitetyn tammi_textchat_js_object.nonce-parametrin kautta.

```
25          jQuery.ajax({
26              type: "post",
27              dataType: "json",
28              url: tammi_textchat_js_object.ajaxurl,
29              data: {
30                  action:`tammi_textchat_savedata`,
31                  security: tammi_textchat_js_object.nonce,
32                  quiz: quizId[2],
33                  question: questionText,
34                  answer: answerText
35              },
36              success: function(msg){
37                  console.log(msg);
38              }
39          });
```

Kuva 10. jQuery-koodiin määritetty AJAX-pyyntö WordPress-ympäristössä

WordPress tarjoaa funktion wp_enqueue_script skriptitiedostojen liittämiseen sivulle (Kuva 11). Funktio ottaa vastaan kolme parametriä. Ensimmäiseksi parametriksi tulee syöttää tiedostolle tunniste, jolla skriptiin voidaan viitata muissa funktioissa. Toiseksi parametriksi tulee syöttää täysi URL-osoite liitettävään tiedostoon. Täyden URL-osoitteen syöttämiseen suositellaan käytettävän WordPressin valmiita funktioita osoitteen hakemiseen, kuten plugins_url, joka palauttaa funktion sisältävän lisäosan kansion URL-osoitteen. Kolmantena parametrinä on taulukko skriptitunnisteista, johon tulee AJAX-pyyntöä varten syöttää vähintään ”jquery”. (WordPress.org q.)

```
128     wp_enqueue_script(
129         'tammi-textchat-js',
130         plugins_url( 'tammi-textchat.js', __FILE__ ),
131         array('jquery'),
132         null,
133         true
134     );
```

Kuva 11. wp_enqueue_script-funktio

wp_enqueue_script-funktioita ei voi ajaa välittömästi, kun PHP-tiedosto ladataan, vaan se tulee sisällyttää funktioon, joka on liitetty johonkin muutamasta tiedostojen liittämiseen tarkoitetuista koukuista. Käytetyn koukun tulisi olla ylläpitosivuilla admin_enqueue_scripts, tai julkisilla sivuilla wp_enqueue_scripts, paitsi kirjautumissivulla, jolla sen tulee olla login_enqueue_scripts. (WordPress.org q.)

AJAX-pyyntöä varten tarvittava nonce-tunniste voidaan luoda käyttäen WordPressin funktiota wp_create_nonce. Funktio ottaa vastaan parametriksi minkä tahansa tekstipätkän, mutta on suositeltua syöttää tunnisteen käyttötarkoitusta kuvaava tekstipätkä. (WordPress.org q.)

Kuten aikaisemmin mainittiin, PHP-koodin kautta pystytään välittämään jQuery-koodille tarvittavia arvoja. Arvot pystytään välittämään käyttämällä WordPressin wp_localize_script-funktiota. (WordPress.org q.)

Kuvan 12 mukaisesti, wp_localize-script-funktio tarvitsee kolme parametriä. Ensimmäiseksi parametriksi tulee syöttää lokalisoitavan skriptitiedoston tunniste, joka määritettiin wp_enqueue_script-funktion kautta. Toiseksi parametriksi tulee syöttää nimi JavaScript-objektille, jonka kautta jQuery-koodi voi kutsua välitettyjä arvoja. Viimeiseksi parametriksi tulee määrittää taulukko, joka sisältää jQuery-koodille välitettävät arvot. (WordPress.org w.)

```
129     wp_localize_script( 'tammi-textchat-js', 'tammi_textchat_js_object',
130         array(
131             'ajaxurl' => admin_url( 'admin-ajax.php' ),
132             'nonce' => $ajax_nonce,
133         )
134     );
```

Kuva 12. wp_localize_script-funktio

AJAX-pyynnön käsittelyä varten tulee luoda funktio, jonka tulee olla liitetty toimintona WordPressin AJAX-koukkuun. AJAX-koukku koostuu kahdesta osuudesta, joista ensimmäinen riippuu siitä, tuleeko käyttäjän olla kirjautunut sisään, ja toinen siitä, mitä jQuery-koodi välittää palvelimelle action-parametrin arvona. (WordPress.org q.)

Mikäli käyttäjän tulee olla kirjautunut sisään, koukun alkuosan tulee olla wp_ajax_, kun taas jos käyttäjän ei tule olla kirjautunut sisään, koukun alkuosan tulee olla wp_ajax_nopriv_. Koukun loppuosan tulee olla jQuery-koodin välittämä action-parametrin arvo. Jos AJAX-pyyntö tulee käsitellä useissa erilaisissa tapauksissa, esimerkiksi sekä kirjautuneilla että kirjautumattomilla käyttäjillä, voidaan funktio liittää useisiin koukkuihin (Kuva 13). (WordPress.org q.)

```
275 add_action( 'wp_ajax_nopriv_tammi_textchat_savedata', 'tammi_textchat_savedata' );
276 add_action( 'wp_ajax_tammi_textchat_savedata', 'tammi_textchat_savedata' );
```

Kuva 13. tammi_textchat_savedata funktio liitetty toimintona kahteen eri AJAX-koukkuun

Funktion, joka käsittelee AJAX-pyynnön, tulisi aina ensimmäisenä tarkistaa jQuery-koodin lähettämä nonce-tunniste, varmistaakseen että pyyntö on lähetetty luotetusta lähteestä. Tunniste voidaan tarkistaa käyttämällä WordPressin funktiota check_ajax_referer. Funktiolle tulee välittää parametriksi tismalleen sama tekstipätkä, kuin aikaisemmin wp_create_nonce-funktiolle. Jos tunniste vastaa aikaisemmin luotua, funktion suorittaminen jatkuu normaalisti. Mikäli tunniste ei vastaa aikaisemmin luotua, funktio keskeytetään välittömästi. (WordPress.org q.)

Kun tunniste on tarkistettu, funktio voi käsitellä jQuery-koodin lähettämän pyynnön mukana lähetettyä dataa. Funktio voi päästä käsiksi pyynnön välittämään dataan $_POST-

muuttujan kautta. $_POST-muuttuja on taulukko, joka sisältää jQuery-koodin AJAX-pyynnölle määrittämän data-taulukon sisällön. (WordPress.org q.)

Kun data on käsitelty, voidaan lähettää vastaus jQuery-koodille. Vastaus voidaan lähettää useassa eri formaatissa, mutta on suositeltavaa käyttää JSON-formaattia, sillä se on tehokas, helppokäyttöinen ja WordPress tarjoaa siihen valmiin wp_send_json-funktion, joka suorittaa kaikki tarpeelliset välivaiheet vastauksen lähettämiseen. (WordPress.org q.)

Kun funktio on suoritettu, pyynnön suorittaminen täytyy lopettaa. Mikäli funktion sisällä ajetaan wp_ajax_response. tai wp_send_json-funktio, lopetus tehdään automaattisesti funktioiden yhteydessä. Muissa tapauksissa funktion tulee lopuksi kutsua WordPressin wp_die-funktiota lopettamista varten. (WordPress.org q.)

2.5.3 AJAX WordPress-lisäosissa

WordPress-lisäosissa AJAX on ylivoimaisesti paras tapa käynnistää prosessi erillään WordPressin sisällöstä. WordPress tarjoaa globaalin JavaScript-muuttujan ajaxurl, joka sisältää täyden osoitteen AJAX-pyyntöjen lähettämistä varten. Osoitteen avulla pystytään lähettämään pyynnöt admin-ajax.php-tiedostoon, riippumatta niitä lähettävän tiedoston sijainnista WordPressin kansiorakenteessa.

ajaxurl-muuttuja on kuitenkin saatavilla vain WordPressin ylläpitopuolella, joten julkisilla sivuilla tulee hyödyntää aikaisemmin mainittua WordPressin wp_localize_script-funktiota. (WordPress.org c.)

3 Tammi Textchat -lisäosan suunnittelu ja kehitys

3.1 Toimeksianto

Työn toimeksiantona oli suunnitella ja kehittää työkalu monivalintakyselyiden luomiseen WordPress-ympäristöön. Työkalulla tuli olla mahdollista luoda, muokata ja esittää monivalintakyselyitä WordPress-sivustoilla. Työkalulla luotujen monivalintakyselyiden tuli myös kerätä käyttäjien tekemistä valinnoista dataa, jota toimeksiantaja ja toimeksiantajan asiakkaat voisivat jälkikäteen tarkastella ja analysoida. Työkalu päätettiin toteuttaa WordPress-lisäosana työnimikkeellä Tammi Textchat.

Monivalintakyselyiden tarkoitus oli palvella sivuston loppukäyttäjää. Kyselyiden tuli esimerkiksi tarjota loppukäyttäjälle hänen etsimäänsä tietoa tai ohjata hänet hänen etsimäänsä sisältöön. Tästä syystä kysymyksiin ei tarvittu mahdollisuutta määrittää oikeita tai vääriä vastauksia. Myöskään useampaa vastausta ei tarvinnut pystyä valitsemaan kerralla.

Työkalun suunnittelussa tuli ottaa huomioon sekä sivuston kehittäjän, ylläpitäjän että loppukäyttäjän näkökulmat. Työkalun tuli olla helppo asentaa ja käyttöönottaa sivuston kehittäjälle, olla monipuolinen mutta selkeä käyttää sivuston ylläpitäjälle, sekä olla houkutteleva ja käytännöllinen sivuston loppukäyttäjälle.

Työkalun kehityksessä tuli noudattaa WordPress-lisäosan kehityksen parhaita käytänteitä ja ottaa huomioon mahdolliset tietoturvariskit. Kehityksessä tuli myös erityisesti panostaa lisäosan skaalautuvuuteen, että työkalua voitaisiin käyttää ja hyödyntää mahdollisimman monilla erilaisilla WordPress-sivustoilla.

3.2 Lisäosan suunnittelu

Suunnitteluvaiheen aluksi suunniteltiin ja priorisoitiin työkalun pääpiirteet. Työkalun tärkeimmäksi ominaisuudeksi todettiin, että sen tuli palvella hyvin sivuston loppukäyttäjää. Toiseksi tärkeäksi ominaisuudeksi nostettiin esiin, että työkalun tuli hyödyttää sivuston ylläpitäjää kerätyn datan kautta ja olla ainakin suhteellisen helppokäyttöinen hallita. Vähiten tärkeäksi pääpiirteeksi todettiin lisäosan helppo asennus ja käyttöönotto, koska se tehtäisiin aina toimeksiantajan eikä heidän asiakkaidensa toimesta.

Tämän prioriteettijärjestyksen pohjalta luotiin järjestys työkalun suunnittelulle. Ensin suunniteltaisiin, miten monivalintakyselyt palvelisivat loppukäyttäjää ja millainen niiden käyttöliittymän tulisi olla. Sitten suunniteltaisiin, millainen kyselyiden hallintakäyttöliittymän tulisi olla ja miten kyselyistä kerätty data annettaisiin sivuston ylläpitäjälle. Lopuksi suunniteltaisiin lisäosan asennusprosessi.

Loppukäyttäjän käyttöliittymän suunnittelun aluksi mietittiin, millä tavoilla monivalintakyselyt voisivat palvella loppukäyttäjää. Esiin nostettiin kolme ominaisuutta, jotka voisivat olla hyödyllisiä loppukäyttäjälle:

- loppukäyttäjän etsimän tiedon esittäminen
- loppukäyttäjän ohjaaminen hänen etsimälleen sivulle
- kohdistettujen tuotteiden ja palveluiden tarjoaminen loppukäyttäjälle.

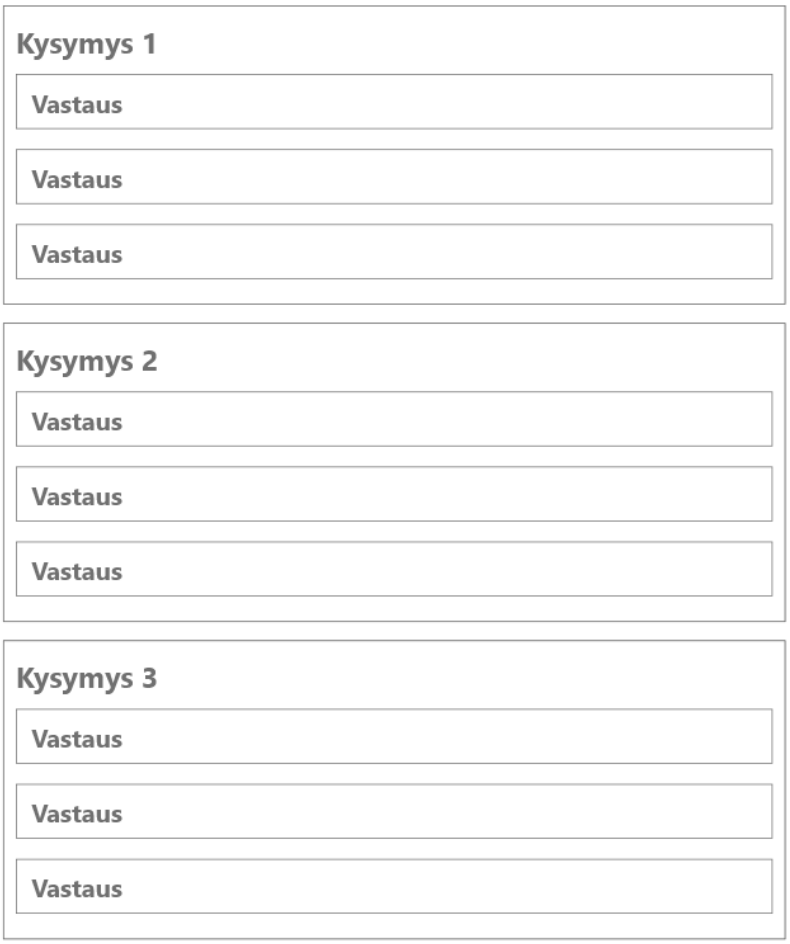
Näiden ominaisuuksien pohjalta päätettiin, mitä erilaisia toimintoja monivalintakyselyn vastauksen valitsemisella tulisi olla. Vastauksen valitsemiselle todettiin tarpeellisiksi toiminnoiksi toiseen kysymykseen ohjaus, sivulle ohjaus sekä sisällön esittäminen. Toiseen kysymykseen ohjaus mahdollistaisi jatkokysymysten esittämisen. Sivulle ohjaus pystyisi ohjaamaan käyttäjän sivulle, jolta hänen etsimänsä tieto tai palvelu löytyy. Sisällön esittämisellä voitaisiin antaa etsitty tieto käyttäjälle välittömästi.

Samalla suunniteltiin monivalintakyselyiden tiedonkeruutapa. Tieto käyttäjän tekemistä valinnoista päätettiin tallentaa erikseen jokaisesta kysymyksestä aina vastauksen valitsemisen yhteydessä. Tällä tavalla olisi helppoa verrata, kuinka paljon mitäkin vastausta oli suhteellisesti valittu missäkin kysymyksessä. Lisäksi, kun tallentaminen tehtäisiin jokaisen klikkauksen jälkeen, tieto käyttäjän valitsemista vastauksista ei olisi riippuvainen siitä, tekeekö käyttäjä monivalintakyselyä loppuun.

Tallennettaviksi tiedoiksi päätettiin: käytössä oleva kysely, käyttäjälle esitetty kysymys, käyttäjän valitsema vastaus sekä aikaleima. Näillä tiedoilla koettiin saatavan lähtökohtaisesti tarpeeksi tietoa datan hyödyntämiseen. Tiedot päätettiin samalla antaa lisäosan ensimmäisessä versiossa sivuston ylläpitäjälle CSV-tiedostona, jonka ylläpitäjä voisi itse käsitellä haluamallaan tavalla.

Seuraavaksi suunniteltiin työkalun hallintakäyttöliittymä. Hallintakäyttöliittymässä tuli ottaa huomioon sekä yksittäisten kysymysten ja niiden vastausten hallinnointi sekä koko kyselyn rakenteen ja kysymysten välisten yhteyksien hallinnointi. Monivalintakyselyiden hallintaan suunniteltiin kolme erilaista hallintakäyttöliittymää, jotka painottivat toiminnaltaan hallinnan eri osa-alueita. Hallintakäyttöliittymien ehdotukset nimettiin lista-, haitari- sekä valkokangaskäyttöliittymiksi.

Listakäyttöliittymässä monivalintakyselyn kysymykset olisivat listattuina allekkain (Kuva 14). Kysymysten alle määriteltäisiin myös kysymysten vastausvaihtoehdot. Itse vastauksen lisäksi vastausvaihtoehdoille määritettäisiin, minkä toiminnon vastauksen valitseminen suorittaisi. Mikäli vastauksen tulisi ohjata käyttäjä toiseen kysymykseen, tulisi vastaukselle määrittää toiseen kysymykseen viittaava uniikki tunniste.

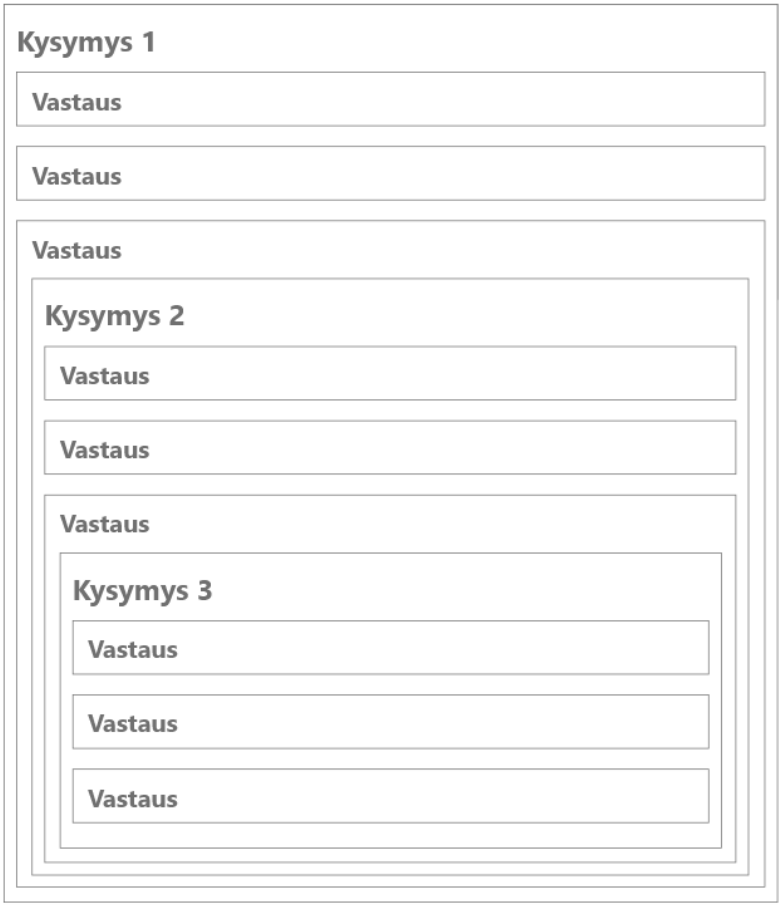


Kuva 14. Listakäyttöliittymän rautalankamalli

Listakäyttöliittymän vahvuuksiksi todettiin joustavuus ja skaalautuvuus. Käyttöliittymän joustavuudella viitattiin siihen, ettei käyttöliittymä rajoita millään tavalla kyselyn rakennetta. Jokin vastaus voitaisiin ylläpitäjän niin halutessaan esimerkiksi asettaa ohjaamaan käyttäjä takaisin kyselyn alkuun. Skaalautuvuudella viitattiin siihen, että kaikki kyselyn kysymykset löytyisivät ja olisivat muokattavissa suhteellisen helposti, vaikka kysymysten määrä olisi melko suuri. Listakäyttöliittymän heikkouksiksi todettiin kysymysten välisten yhteyksien määrittämisen hankaluus sekä kyselyn kokonaisrakenteen heikko hahmotettavuus.

Haitarikäyttöliittymä (Kuva 15) suunniteltiin toimimaan hyvin samantapaisesti kuin listakäyttöliittymä, mutta yrittäen mahdollistaa helpompi jatkokysymysten yhdistäminen sekä kyselyn kokonaisrakenteen hahmottaminen. Haitarikäyttöliittymässä toiseen kysymykseen ohjaamiseen ei tarvittaisi kysymyksille uniikkeja tunnisteita, vaan jatkokysymys tulisi syöttää suoraan siihen ohjaavan vastauksen alapuolelle. Tämä myös auttaisi ylläpitäjää

hahmottamaan kyselyn rakenteen, kun hallintakäyttöliittymässä kysymykset olisivat aina kyselyssä edettävän polun mukaisessa järjestyksessä.

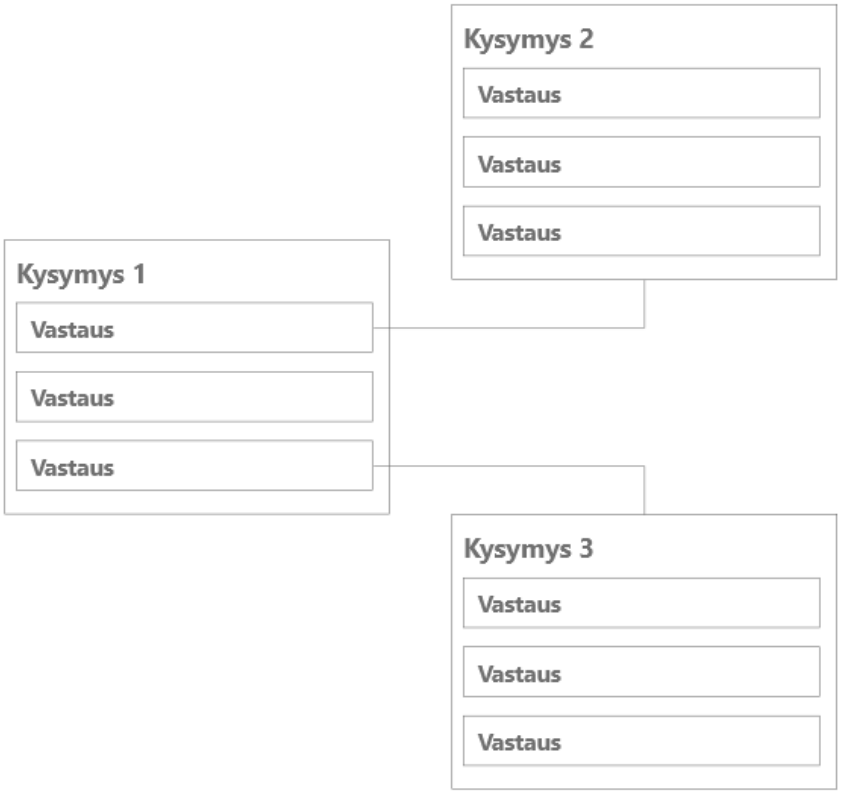


Kuva 15. Haitarikäyttöliittymän rautalankamalli

Haitarikäyttöliittymän vahvuudeksi todettiinkin kyselyn kokonaisrakenteen selkeys sekä kysymysten välisten yhteyden luomisen helppous. Haitarikäyttöliittymän heikkouksiksi todettiin sen sijaan listakäyttöliittymään verrattuna heikentyneet joustavuus sekä skaalautuvuus. Joustavuus heikentyisi, koska jatkokysymysten olisi pakko olla aina aiemman kysymyksen vastauksen alla, eikä loppukäyttäjää pystyttäisi ohjamaan aikaisempaan kysymykseen tai samaan kysymykseen useasta paikasta. Skaalautuvuus taas heikentyisi, koska mitä pidemmäksi kysely tehtäisiin, sitä vaikeammaksi kyselyn myöhäisempien kysymysten muokkaaminen tulisi.

Valkokangaskäyttöliittymä (Kuva 16) eroaa kahdesta muusta käyttöliittymäehdotuksesta huomattavasti. Valkokangaskäyttöliittymässä kysymyksiä ei esitettäisi listassa allekkain vaan elementteinä valkokankaalla. Näitä elementtejä pystyttäisiin lisätä valkokankaalle ja

sitten muokata ja siirrellä halutulla tavalla. Yhteyksiä kysymysten välille pystyttäisiin tehdä vetämällä näkyviä kytköksiä vastauksen ja jatkokysymyksen välille. Kysymyksien lisäksi valkokankaalle voitaisiin luoda elementeiksi myös kyselyn päätepisteitä: ohjauksia toisille sivuille sekä tulostettavia sisältöjä.



Kuva 16. Valkokangaskäyttöliittymän rautalankamalli

Valkokangaskäyttöliittymän vahvuuksiksi todettiin helppokäyttöisyys ja kyselyn rakenteen selkeys. Valkokankaan käyttäminen koettiin listaa käyttäjäystävällisemmäksi ja kysymysten välisten visuaalisten kytkösten nähtiin esittävän kyselyn rakenteen paljon haitarikäyttöliittymääkin selkeämmin. Valkokangaskäyttöliittymän heikkouksiksi todettiin heikko skaalautuvuus sekä muihin käyttöliittymäehdotuksiin verrattuna vaativampi kehitysprosessi. Käyttöliittymä skaalautuu heikosti suurempiin kyselyihin, sillä elementtien määrän kasvaessa, valkokankaan nähtiin ruuhkautuvan heikentäen sekä käyttöliittymän helppokäyttöisyyttä että rakenteen selkeyttä.

Käyttöliittymäehdotusten vahvuuksien ja heikkouksien arvioimisen jälkeen, käyttöliittymäehdotuksia vertailtiin keskenään (Taulukko 1). Käytettäväksi käyttöliittymäksi päätettiin valita listakäyttöliittymä. Haitarikäyttöliittymä hylättiin ensimmäisenä, sillä se koettiin

samankaltaiseen listakäyttöliittymään verrattuna palvelevan ylläpitäjää huonommin heikomman joustavuuden ja skaalautuvuuden takia. Listakäyttöliittymä valittiin valkokangaskäyttöliittymän yli myös paremman skaalautuvuuden sekä nopeamman kehitysprosessin takia. Lisäksi listakäyttöliittymän kyselyrakenteen hahmotettavuuden heikkous nähtiin mahdollisena korjata kehittämällä jälkikäteen kysymyslistasta erillinen kyselyrakennenäkymä.

Käyttöliittymä	Lista	Haitari	Valkokangas	Kerroin (Prioriteetti)
Joustavuus	5	1	3	1,5
Skaalautuvuus	4	2	2	1,5
Helppokäyttöisyys	3	3	5	1,5
Rakenteen määritys	2	4	4	1
Rakenteen selkeys	2	4	4	1
Arvioitu kehitysaika	5	5	2	1
Yhteensä	**27**	22	25	

Taulukko 1. Käyttöliittymäehdotusten vertailu

Hallintakäyttöliittymä päätettiin toteuttaa pääosin käyttäen Advanced Custom Fields -lisäosan lisäkenttiä, koska sen oletettiin nopeuttavan hallintakäyttöliittymän kehitysprosessia huomattavasti. ACF-lisäkenttien avulla hallintakäyttöliittymästä saataisiin myös suhteellisen käyttäjäystävällinen ilman lisätyötä, koska ACF:n valmiit kentät ovat jo oletusarvoisesti suhteellisen helppokäyttöisiä.

Koska työkalu päätettiin toteuttaa WordPress-lisäosana, sen asentaminen eri WordPress-sivustoille on vaivatonta. Työkalu on kuitenkin riippuvainen Advanced Custom Fields -lisäosasta hallintakäyttöliittymässä käytettävien lisäkenttien vuoksi. Advanced Custom Fields -lisäosan Pro-version tulee siksi olla asennettuna sivustolle ennen Tammi Textchat -lisäosan asennusta.

3.3 Lisäosan alustaminen

Kehityksen aluksi lisäosalle luotiin päätiedostoksi tammi-textchat.php. Tähän tiedostoon sisällytettäisiin lisäosan tiedot sekä lisäosan kaikki PHP-ohjelmointikielellä toteutettavat pääominaisuudet. Lisäosan JavaScript-koodia varten luotiin erillinen tammi-textchat.js-tiedosto. Lisäosan tyylimäärittelyjä varten luotiin taas erillinen tammi-textchat-style.css-tiedosto.

Päätiedoston alkuun kirjoitettiin lisäosan tiedot DocBlock-osioon. Lisäosan tietoihin sisällytettiin lisäosan nimi, lisäosan kuvaus, lisäosan versio, lisäosan tekijä sekä tekijän verkko-osoite (Kuva 17).

```
/**
 * Plugin Name: Tammi Textchat
 * Description: Lisäosa monivalintakyselyiden luomiseen.
 * Version: 1.0
 * Author: Tammi Digital
 * Author URI: https://tammidigital.fi/
 */
```

Kuva 17. Tammi Textchat -lisäosan version 1.0 DocBlock-osio

3.4 Monivalintakyselyiden hallinta

Monivalintakyselyiden hallinnan kehitys jaettiin kahteen osaan. Monimutkaisen ja muusta sisällöstä eroavan sisältönsä takia monivalintakyselyille päätettiin luoda erillinen mukautettu artikkelityyppi. Tälle artikkelityypille tuli sen jälkeen kehittää itse hallintakäyttöliittymä.

Artikkelityypin luominen tuli tehdä PHP-koodin kautta lisäosan päätiedostoon. Hallintakäyttöliittymä taas tultaisiin toteuttamaan Advanced Custom Fields -lisäosan käyttöliittymän kautta.

3.4.1 Artikkelityypin luominen

Artikkelityypin luomista varten tehtiin funktio tammi_textchat_custom_post_type. Funktio ajaa artikkelityyppien luomista varten tarkoitetun register_post_type-funktion. tammi_textchat_custom_post_type-funktio kiinnitettiin toimintona init-koukkuun, joka laukaistaan heti kun WordPress on latautunut.

register_post_type-funktiolle välitetään parametreina artikkelityypin avain sekä taulukko luotavalle artikkelityypille tarpeellisista tiedoista. Kuvan 18 mukaisesti monivalintakyselyiden artikkelityypin tiedoiksi asetettiin

- labels-arvolla artikkelityypin nimeksi ”Monivalintakyselyt” ja yksittäisen artikkelin nimeksi ”Kysely”
- supports-arvolla artikkelityypille tuki otsikko- ja lyhytkoodiominaisuuksille
- public-arvolla artikkelityyppi julkiseksi
- has_archive-arvolla artikkelityypin arkistosivu pois käytöstä
- publicly_queryable-arvolla ettei artikkelityyppiä pysty hakea sivuston julkisella puolella.

```
153  function tammi_textchat_custom_post_type() {
154      register_post_type('tammi_textchat',
155          array(
156              'labels'      => array(
157                  'name'          => __('Monivalintakyselyt', 'textdomain'),
158                  'singular_name' => __('Kysely', 'textdomain'),
159              ),
160              'supports' => array( 'title','shortcode' ),
161              'public'      => true,
162              'has_archive' => false,
163              'publicly_queryable'  => false,
164          )
165      );
166  }
167  add_action('init', 'tammi_textchat_custom_post_type');
```

Kuva 18. tammi_textchat_custom_post_type-funktio ja sen liittäminen init-koukkuun

3.4.2 Hallintakäyttöliittymän kehitys

Monivalintakyselyiden hallintakäyttöliittymän luomista varten luotiin Advanced Custom Fields -lisäosan kenttäryhmä. Kenttäryhmän nimeksi asetettiin Monivalintakysely. Kenttäryhmä kohdistettiin käytettäväksi ainoastaan monivalintakyselyille luodulla mukautetulla artikkelityypillä.

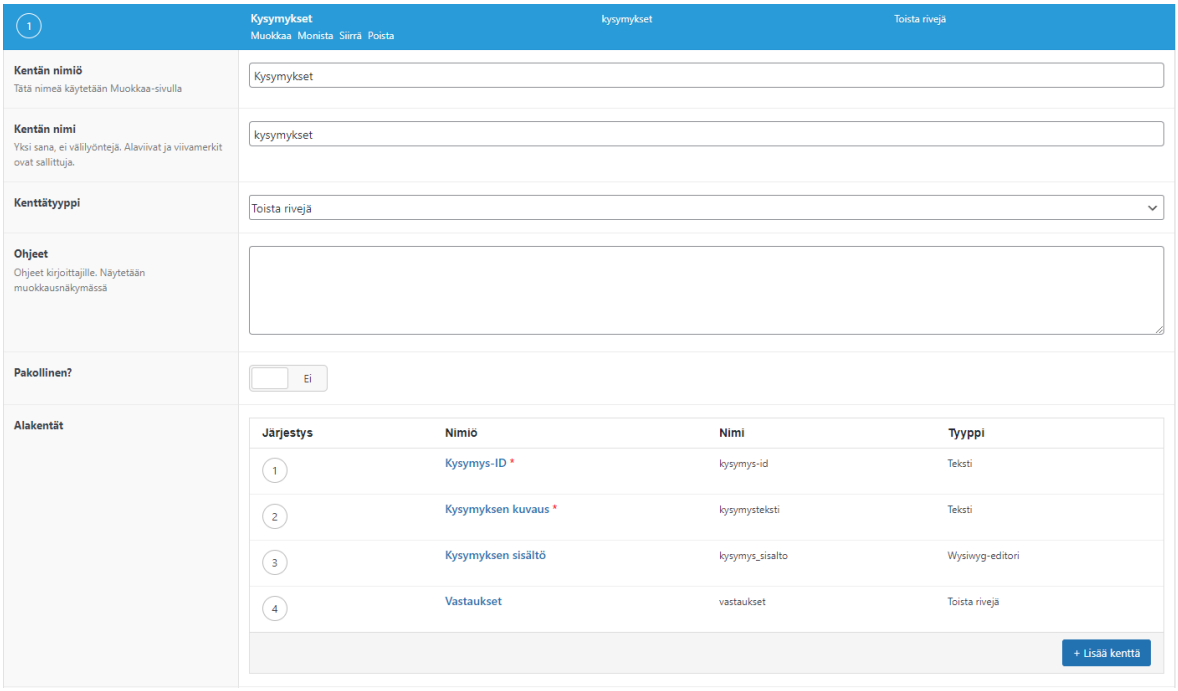
Monivalintakyselyiden tietorakenne koostuu useasta kerroksesta, jonka takia hallintakäyttöliittymän kenttäryhmänkin kentät koostuvat useista tasoista. Ylimmäksi tasoksi luotiin Toista rivejä -kenttä Kysymykset. Tähän kenttään lisättäisiin kaikki monivalintakyselyyn kuuluvat kysymykset.

Kysymyksien alle luotiin alakentiksi (Kuva 19):

- tekstikenttä Kysymys-ID
- tekstikenttä Kysymyksen kuvaus
- wysiwyg-kenttä Kysymyksen sisältö
- Toista rivejä -kenttä Vastaukset.

Kysymys-ID:tä käytetään kysymyksen uniikkina tunnisteena siihen viitatessa. Kysymyksen kuvausta käytetään datan tallentamiseen tietokantaan. Kysymyksen sisältöön syötetään kysymyksessä esitettävä sisältö, joka voi wysiwyg-tyyppisessä kentässä olla muotoillun tekstin lisäksi esimerkiksi mediaa kuten kuvia. Tietokantaan tallennetaan kysymyksen ID:n ja

sisällön sijaan kysymyksen kuvaus, koska ID:n ei ajateltu olevan tietojen tulkitsemista varten tarpeeksi kuvaava ja koska Kysymyksen sisältö -kentän sisällön ajateltiin olevan liian vaikealukuista tekstimuodossa. Vastauksiin lisättäisiin kaikki kysymyksen vastaukset.



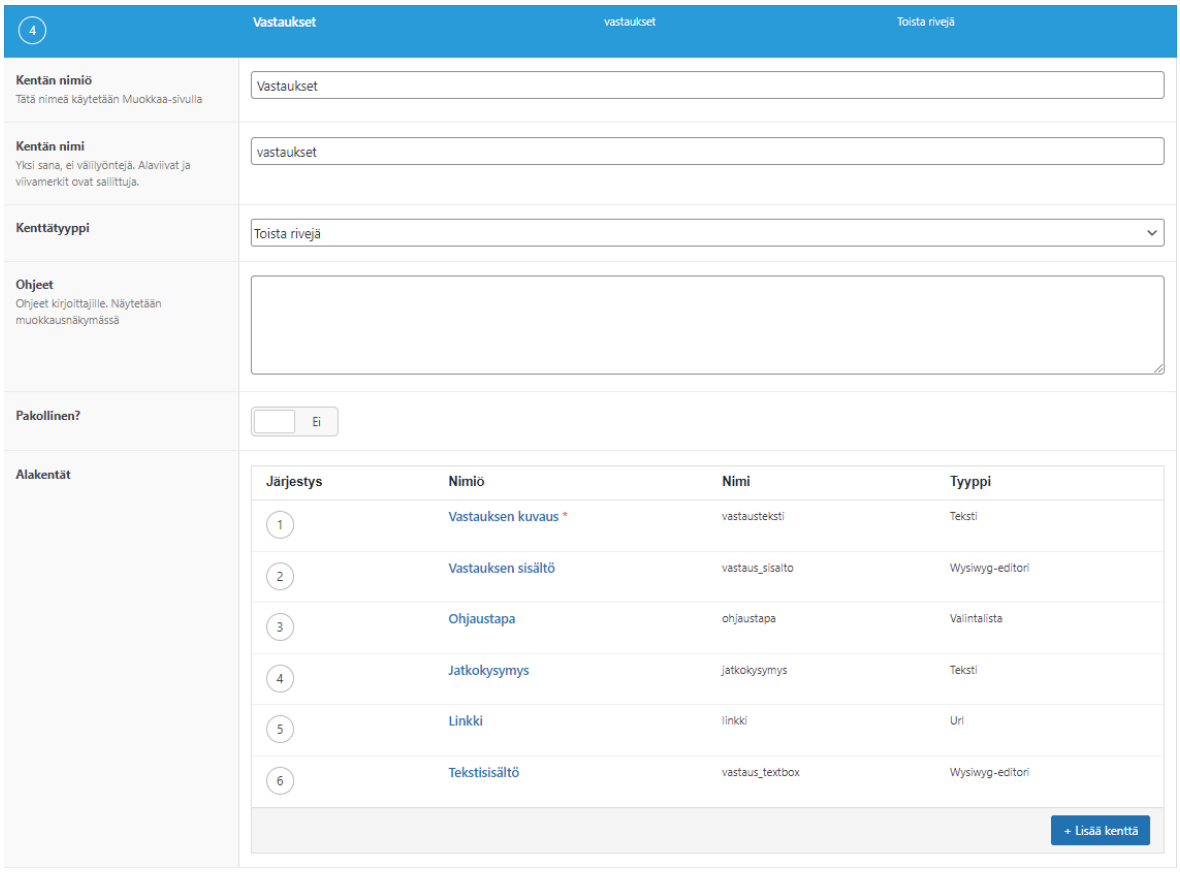
Kuva 19. Kysymykset-kenttä ja sen alakentät

Vastauksien alle luotiin alakentät (Kuva 20):

- tekstikenttä Vastauksen kuvaus
- wysiwyg-editori Vastauksen sisältö
- valintalista Ohjaustapa.

Lisäksi vastauksien alle luotiin myös ehdolliset kentät, jotka esitettäisiin riippuen valitusta ohjaustavasta:

- tekstikenttä Jatkokysymys
- url-kenttä Linkki
- wysiwyg-editori Tekstisisältö.



Kuva 20. Vastaukset-kenttä ja sen alakentät

Vastauksen kuvausta, kuten kysymyksenkin kuvausta, käytetään datan tietokantaan tallentamiseen. Vastauksen sisältö esitetään vastauksen sisällä. Ohjaustavalla määritetään toiminto, joka halutaan suoritettavan vastausta klikkaamalla. Ohjaustavan vaihtoehdoiksi asetettiin Jatkokysymys, Linkki ja Teksti.

Koska ohjaustavat eroavat toiminnaltaan toisistaan, kaikille kolmelle ohjaustavalle tarvittiin omat lisäkenttänsä, jotka esitettäisiin ehdollisesti riippuen valitusta ohjaustavasta. Jatkokysymys asetettiin näkymään ehdollisesti vain, jos ohjaustavaksi on valittu jatkokysymys. Jatkokysymykseen tulee syöttää sen kysymyksen kysymys-ID, johon halutaan vastauksen valitsemisen ohjaavan. Linkki asetettiin näkymään ehdollisesti vain, jos ohjaustavaksi on valittu linkki. Linkkiin tulee syöttää se URL-osoite, johon käyttäjä ohjataan vastauksen valittuaan. Tekstisisältö asetettiin näkymään ehdollisesti vain, jos ohjaustavaksi on asetettu teksti. Tekstisisältöön tulee syöttää sisältö, joka halutaan esittää, kun vastaus valitaan.

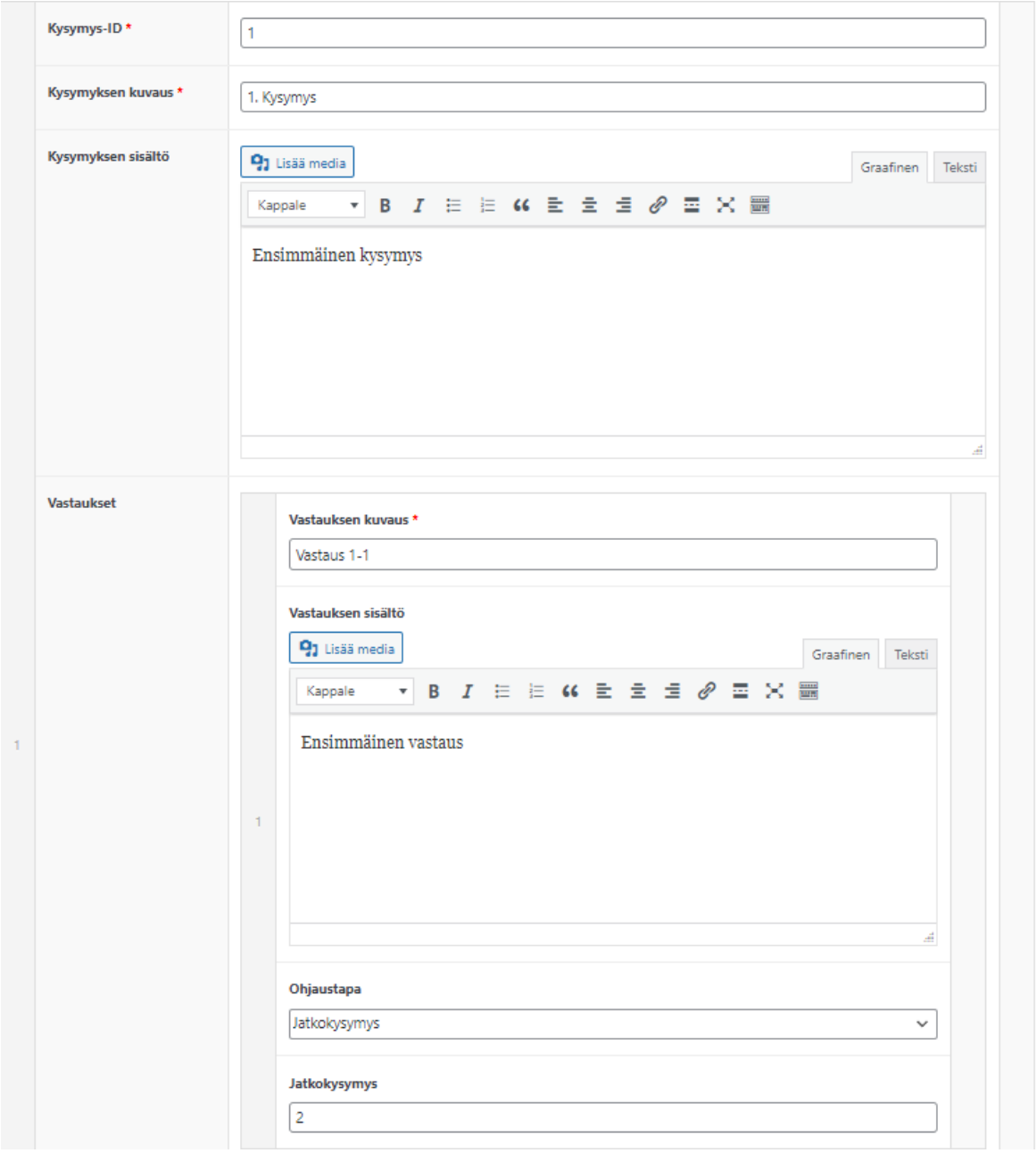
Kyselyn rakenteen lisäksi hallintakäyttöliittymään luotiin kenttiä, jolla pystyttäisiin muokkaamaan helposti monivalintakyselyn ulkoasua (Kuva 21). Ulkoasua muokkaavilla kentillä mahdollistettiin pääosin vain värien, tekstikoon ja asettelun muuttamista, sillä kyselyille ajateltiin

useimmiten kehitettävän erilliset CSS-määritykset kohdesivuston teemaan, jolla monivalintakysely tyyliteltäisiin sivuston teeman mukaiseksi.

2	Tyylimääritykset		Accordion
3	Tausta		Välilehti
4	Tausta Muokkaa Monista Siirrä Poista	tausta	"Tosi / Epätosi" -valinta
5	Reunaviiva	reunaviiva	"Tosi / Epätosi" -valinta
6	Taustaväri	taustavari	Värinvalitsin
7	Reunaviivan väri	reunaviivan_vari	Värinvalitsin
8	Reunojen pyöristys (border-radius)	container_br	Teksti
9	Margin	container_margin	Teksti
10	Padding	container_padding	Teksti
11	Kysymys		Välilehti
12	Kysymyksen tekstiväri	kysymysvari	Värinvalitsin
13	Fonttikoko	kysymys_fontsize	Teksti
14	Keskitys	kysymys_keskitys	Valintalista
15	Vastaukset		Välilehti
16	Vastauksen tekstiväri	vastausvari	Värinvalitsin
17	Fonttikoko	vastaus_fontsize	Teksti
18	Vastauksen reunaväri	vastausreuna	Värinvalitsin
19	Vastauksen leveys (%)	vastausleveys	Numero
20	Reunojen pyöristys (border-radius)	vastaus_br	Teksti
21	Padding	vastaus_padding	Teksti

Kuva 21. Kentät monivalintakyselyn ulkoasun muokkaamiseen

Kenttäryhmän avulla lisäkentät olivat nyt käytettävissä monivalintakyselyiden mukautetun artikkelityypin artikkeleilla. Monivalintakyselyn kenttien sisältöä pystyttiin muokkaamaan artikkelin ylläpitotilassa (Kuva 22) ja kutsua koodissa artikkelia esittäessä.



Kuva 22. Kyselyn hallintakäyttöliittymä artikkelin ylläpitotilassa

Luotu kenttäryhmä tuli pystyä siirtämään lisäosan mukana. Tätä varten kehitettiin lisäosan päätiedoston PHP-koodiin tammi_textchat_acf_json_save_point-funktio kenttäryhmän tallentamista varten sekä tammi_textchat_acf_json_load_point-funktio kenttäryhmän lataamista varten (Kuva 23). tammi_textchat_acf_json_save_point-funktio kiinnitetiin suodattimena acf/settings/save_json-koukkuun, jonka avulla tehty kenttäryhmä tallennettaan lisäosan tiedostoihin. tammi_textchat_acf_json_load_point-funktio kiinnitettiin taas suodattimena acf/settings/load_json-koukkuun, jonka avulla kenttäryhmä ladataan lisäosan tiedostoista sivustolle.

```
206  // Saving ACF fields to plugin dir
207  function tammi_textchat_acf_json_save_point( $path ) {
208      // update path
209      $path = plugin_dir_path(__FILE__) . 'acf-json';
210
211      // return
212      return $path;
213  }
214  add_filter('acf/settings/save_json', 'tammi_textchat_acf_json_save_point');
215
216  // Loading ACF fields from plugin dir
217  function tammi_textchat_acf_json_load_point( $paths ) {
218      unset($paths[0]);
219      $paths[] = plugin_dir_path(__FILE__) .'acf-json';
220      return $paths;
221  }
222  add_filter('acf/settings/load_json', 'tammi_textchat_acf_json_load_point');
```

Kuva 23. ACF-kenttäryhmän tallentaminen ja lataaminen

3.5 Skripti- ja tyylitiedostojen liittäminen lisäosaan

Lisäosaan päätettiin seuraavaksi kehittää monivalintakyselyn esittäminen loppukäyttäjälle. Monivalintakyselyn esittämistä varten oli tuli ensin liittää mukaan aiemmin luodut tammi-textchat.js- sekä tammi-textchat-style.css-tiedostot.

Lisäosan JavaScript-tiedoston liittämistä varten luotiin erillinen funktio tammi_textchat_init_js (Kuva 24). Funktio luo ensin nonce-tunnisteen AJAX-toimintoa varten WordPressin wp_create_nonce-funktiolla ja tallentaa sen muuttujaan. Tämän jälkeen funktio hakee ja liittää tammi-textchat.js-tiedoston lisäosaan wp_enqueue_script-funktiolla, asettaen tiedoston tunnisteeksi ”tammi-textchat-js”, osoitteeksi polun tiedostoon ja skriptitunnisteiksi pelkän jqueryn. Tiedoston liittämisen jälkeen sille tuli vielä välittää AJAX-

toiminnolle tarpeelliset arvot wp_localize_script-funktiolla. wp_localize_script-funktiolle annettiin tunnisteeksi aiemmin määritetty tammi-textchat-js, objektin nimeksi tammi_textchat_js_object ja välitettäviksi arvoiksi admin-ajax.php-tiedoston osoite sekä funktion alussa generoitu nonce-tunniste. tammi_textchat_init_js-funktio liitettiin lopuksi toimintona wp_enqueue_scripts-koukkuun, koska tiedostoa tarvittaisiin julkisilla sivuilla.

```
125  function tammi_textchat_init_js() { // Get JS
126      $ajax_nonce = wp_create_nonce( "tammi-textchat-nonce" ); // Create Nonce
127
128      wp_enqueue_script(
129          'tammi-textchat-js',
130          plugins_url( 'tammi-textchat.js', __FILE__ ),
131          array('jquery'),
132          null,
133          true
134      );
135      wp_localize_script( 'tammi-textchat-js', 'tammi_textchat_js_object',
136          array(
137              'ajaxurl' => admin_url( 'admin-ajax.php' ),
138              'nonce' => $ajax_nonce,
139          )
140      );
141  }
142  add_action('wp_enqueue_scripts','tammi_textchat_init_js');
```

Kuva 24. tammi_textchat_init_js-funktio ja sen liittäminen wp_enqueue_scripts-koukkuun

Lisäosan tyylitiedosto päätettiin ensin vain rekisteröidä, jotta se voitaisiin liittää erikseen aina tarpeen mukaan. Tyylitiedosto rekisteröitiin wp_register_style-funktiolla, jolla sen tunnisteeksi asetettiin tammi-textchat-style. Tällä tunnisteella tyylitiedostoon voitaisiin myöhemmin viitata wp_enqueue_style-funktiolla.

3.6 Monivalintakyselyn esittäminen

Monivalintakyselyn liittämistä sivustolle varten luotiin funktio tammi_textchat_create_monivalinta. Funktion tuli ottaa vastaan monivalintakyselyartikkeli, jonka pohjalta kysely tuli luoda, ja palauttaa kyselyn HTML-rakenne. Tätä funktiota voitaisiin kutsua aina, kun kysely tuli esittää sivulla. Funktiolle määritettiin parametriksi id, johon tuli välittää tulostettavan monivalintakyselyartikkelin uniikki artikkeli-id.

Funktio hakee ensimmäisenä kyselylle tarpeelliset tyylimäärittelyt sekä kyselyn artikkelin (Kuva 25). Tyylimäärittelyt haetaan liittämällä mukaan tammi-textchat-style-tyylitiedosto

wp_enqueue_style-funktiolla hyödyntäen aiemmin wp_register_style-funktiolla määritettyä tunnistettta. Tämän jälkeen funktio hakee halutun monivalintakyselyn artikkelin käyttämällä get_post-funktiota hyödyntäen funktiolle parametrina välitettyä artikkeli-id:tä. get_post-funktion palauttama artikkeliolio tallennetaan muuttujaan, jotta artikkeliin voidaan viitata aina tarvittaessa.

```
17      wp_enqueue_style('tammi-textchat-style'); // Get plugin style
18      $kysely = get_post($id['id']); // Save the post id to a variable
```

Kuva 25. Tyylien ja artikkelin hakeminen tammi_textchat_create_monivalinta-funktiossa

Tämän jälkeen funktiossa käsitellään artikkelille määritettyjä lisäkenttiä. Ensimmäisenä tarkistetaan have_rows-funktion avulla, onko kyselylle luotu ollenkaan kysymyksiä. Jos kysymyksiä ei ole, funktio ei tee muita toimintoja. Jos kyselylle on luotu kysymyksiä, haetaan ensimmäisenä kaikki kyselylle lisäkenttien kautta määritetyt kysymysten ja vastausten tyylimäärittelyt get_field-funktiolla ja tallennetaan ne muuttujiin (Kuva 26).

```
19      if(have_rows('kysymykset', $kysely->ID)) {
20          // Get question and answer styles
21          $kysymysvari = get_field('kysymysvari', $kysely->ID);
22          $kysymysfontti = get_field('kysymys_fontsize', $kysely->ID);
23          $kysymyskeskitys = get_field('kysymys_keskitys', $kysely->ID);
24          $vastausvari = get_field('vastausvari', $kysely->ID);
25          $vastausfontti = get_field('vastaus_fontsize', $kysely->ID);
26          $vastausradius = get_field('vastaus_br', $kysely->ID);
27          $vastausreuna = get_field('vastausreuna', $kysely->ID);
28          $vastauspadding = get_field('vastaus_padding', $kysely->ID);
```

Kuva 26. Kyselyn kysymyksien tarkistus ja lisäkenttien tyylimääritysten hakeminen

Seuraavaksi funktiossa luodaan monivalintakyselyn HTML-rakenne. Ennen rakenteen luomista aloitetaan ulostulon puskurointi ob_start-funktiolla, minkä avulla kaikki esitettävä sisältö kerätään yhteen pakettiin, joka pystytään palauttamaan helposti kerralla. Ensimmäisenä HTML-rakenteelle luodaan kyselyä ympäröivä div-elementti (Kuva 27). Elementille asetetaan luokaksi tammi-textchat-container ja id:ksi monivalintakyselyn artikkeli-id:hen perustuva uniikki tunniste. Elementille tuli asettaa uniikki tunniste, että sivuilla, joilla on

useampi monivalintakysely, pystyttäisiin erottamaan kyselyt toisistaan. Elementin tyyleiksi asetetaan myös lisäkenttien kautta syötettyjen tyylimääritysten arvot, jos niitä on syötetty.

```
29      // begin output buffering to return the full quiz at once
30      ob_start();
31      echo '<div class="tammi-textchat-container" id="tammi-textchat-'.$id['id'].'" style="';
32      if (get_field('tausta', $kysely->ID)) { // Background Color
33          echo 'background-color: '.get_field('taustavari', $kysely->ID).';';
34      }
35      if (get_field('reunaviiva', $kysely->ID)) { // Border
36          echo 'border: 1px solid '.get_field('reunaviivan_vari', $kysely->ID).';';
37      }
38      echo 'border-radius: '.get_field('container_br', $kysely->ID).';
39      margin: '.get_field('container_margin', $kysely->ID).';
40      padding: '.get_field('container_padding', $kysely->ID).';';
41      echo '">';
```

Kuva 27. Ulostulon puskuroinnin aloitus sekä kyselyä ympäröivän elementin luominen

Seuraavaksi HTML-rakenteeseen tuli luoda kaikki monivalintakyselyn kysymykset. Aluksi kysymysten esittämistä varten luotiin toistorakenne, joka kävisi kaikki kyselyn kysymykset läpi (Kuva 28). Ennen toistorakennetta alustetaan muuttujalle first_row arvoksi true, jonka avulla ensimmäinen kysymys pystyttäisiin erottamaan toistorakenteen sisällä. Kaikki kysymysten kenttään syötetyt rivit käydään läpi while-toistorakenteella. Toiston aluksi haetaan rivin tiedot the_row-funktiolla. Mahdollisia vastauksen valitsemisesta esitettäviä sisältöjä varten alustetaan text_boxes-muuttuja tyhjäksi taulukoksi. Kysymyksen kysymys-id-kenttään syötetty tunniste haetaan ja tallennetaan question_id-muuttujaan, jotta siihen voidaan viitata myöhemmin.

```
42      $first_row = true; // For making the first question visible
43      while(have_rows('kysymykset', $kysely->ID)) {
44          the_row(); // Get the question row
45          $text_boxes = array(); // To create text-answers after the question
46          $question_id = get_sub_field('kysymys-id', $kysely->ID);
```

Kuva 28. Kysymysten toistorakenteen avaus sekä tarpeelliset muuttujat

Muuttujien asettamisen jälkeen toistorakenteessa luodaan kysymyksen sisältö (Kuva 29). Jos kyseessä on ensimmäinen rivi, luodaan kysymystä ympäröivä div-elementti, jonka luokaksi asetetaan tammi-textchat-kysymys-container ja id:ksi monivalintakyselyn artikkeli-id:hen sekä kysymyksen kysymys-id:hen perustuva tunniste, ja lopuksi asetetaan first_row-

muuttujan arvoksi false. Jos kyseessä ei ole ensimmäinen rivi, luodaan muuten samanlainen div-elementti, mutta elementille asetetaan lisäksi luokka tammi-textchat-kysymys-hidden ja elementti asetetaan tyyleiltään piilotetuksi. Ympäröivän elementin luomisen jälkeen luodaan div-elementti kysymyksen sisällölle, jonka luokaksi asetetaan tammi-textchat-kysymys, data-question-arvoksi kysymyksen kuvaus ja tyyleiksi lisäkenttien kautta määritetyt tyylimääritykset. Elementin sisään tulostetaan kysymykselle syötetty sisältö, jonka jälkeen elementti suljetaan.

```
47          if ($first_row) {
48              echo '<div class="tammi-textchat-kysymys-container"
49              id="tammi-textchat-'.$id['id'].'-kysymys-'.$question_id.'">';
50              $first_row = false;
51          } else {
52              echo '<div class="tammi-textchat-kysymys-container tammi-textchat-kysymys-hidden"
53              id="tammi-textchat-'.$id['id'].'-kysymys-'.$question_id.'" style="display:none;">';
54          }
55          echo '<div class="tammi-textchat-kysymys"
56          data-question="'.get_sub_field('kysymysteksti', $kysely->ID).'"
57          style="color:'.$kysymysvari.';
58          font-size:'.$kysymysfontti.';
59          text-align:'.$kysymyskeskitys.';
60          ">';
61          the_sub_field('kysymys_sisalto', $kysely->ID); // Question text
62          echo '</div>
```

Kuva 29. Kysymyksen sisällön esittäminen

Kysymyksen sisällön jälkeen tuli luoda kysymyksen vastaukset. Vastauksille luodaan oma toistorakenteensa, joka käy kaikki kysymyksen vastaukset läpi (Kuva 30). Ennen toistorakenteen avaamista luodaan vastauksia ympäröivä div-elementti, jolle asetetaan luokka tammi-textchat-vastaukset, ja alustetaan vastaus_count-muuttuja kierroslaskuriksi arvolla 1. Tämän jälkeen avataan while-toistorakenne, jonka ehdoksi asetetaan have_rows-funktio, jolle välitetään Vastaukset-kenttä. Toiston aluksi haetaan jälleen rivin tiedot the_row-funktiolla. Tämän jälkeen haetaan ja tallennetaan muuttujiin vielä vastauksen kuvaus sekä sisältö.

```
63          <div class="tammi-textchat-vastaukset">';
64          $vastaus_count = 1;
65          while(have_rows('vastaukset')) { // Answers
66              the_row();
67              $vastausteksti = get_sub_field('vastausteksti', $kysely->ID);
68              $vastaussisalto = get_sub_field('vastaus_sisalto', $kysely->ID);
```

Kuva 30. Vastausten toistorakenteen alustus ja avaus sekä tarpeelliset muuttujat

Muuttujien asettamisen jälkeen tulostetaan vastauksen sisältö (Kuva 31). Vastauksen sisällön ympärille luodaan div-elementti, jonka luokaksi asetetaan tammi-textchat-vastaus-container ja tyylimäärityksiksi lisäkenttien kautta määritetyt tyylimääritykset. Tämän jälkeen tarkistetaan, mikä ohjaustapa vastaukselle on valittu.

Jos vastauksen ohjaustapa on jatkokysymys, vastausta ympäröivä elementti asetetaan suorittamaan klikkauksen yhteydessä nextQuestion-funktio, joka ohjaa käyttäjän jatkokysymykseen. Funktiolle välitetään kyselyn id, kysymyksen id sekä vastaukselle määritetty jatkokysymyksen id, jotta käyttäjä pystytään ohjaamaan oikeaan kysymykseen. Tämän jälkeen luodaan vastauksen sisällölle div-elementti, jolle asetetaan luokiksi tammi-textchat-vastaus sekä kyselyn id:stä ja toistorakenteen kierrosnumerosta koottava uniikki luokka. Elementille asetetaan myös data-answer-arvoksi vastauksen kuvaus tietokantaan tallentamista varten, sekä tyylimäärittelyiksi lisäkenttien kautta asetetut tyylimääritykset. Lopuksi elementin sisään tulostetaan vastauksen sisältö ja elementti suljetaan.

Jos vastauksen ohjaustapa on teksti, vastausta ympäröivä elementti asetetaan sen sijaan suorittamaan klikkauksesta showText-funktio, joka tulee esittämään käyttäjälle halutun sisällön. Funktiolle välitetään kyselyn id, kysymyksen id sekä text_boxes-taulukon alkioiden määrä, jotta funktio pystyy ohjaamaan käyttäjän oikeaan sisältöön. Tämän jälkeen elementti luodaan samalla tavalla, kuin silloin kun ohjaustapana on jatkokysymys. Lopuksi kuitenkin myös lisätään vastauksen Tekstisisältö-kentän sisältö alkiona text_boxes-taulukkoon käyttäen array_push-funktiota.

Jos vastauksen ohjaustapa on linkki, vastausta ympäröivää elementtiä ei aseteta suorittamaan funktiota klikkauksen yhteydessä. Vastauksen sisällölle luodaan myös div-elementin sijaan linkkielementti, jolle asetetaan muuten samat luokat ja tyylimääritykset kuin div-elementille, mutta niiden lisäksi asetetaan myös elementin href-arvoksi vastauksen Linkki-kenttään syötetty URL-osoite. Tällä tavalla, kun vastausta klikataan, käyttäjä ohjataan kenttään syötettyyn URL-osoitteeseen.

Vastauksen sisällön luomisen jälkeen, vastausta ympäröivä div-elementti suljetaan. Vastausten määrää laskevaa vastaus_count-muuttujaa myös kasvatetaan yhdellä.

```
echo '<div class="tammi-textchat-vastaus-container"
style="flex: 0 0 '.get_field('vastausleveys', $kysely->ID).'%"';
if(get_sub_field('ohjaustapa') == 'Linkki') { // Link
    echo '>
    <a class="
        tammi-textchat-link tammi-textchat-vastaus
        tammi-textchat-'.$id['id'].'-vastaus-'.$vastaus_count.'
    "
    href="'.get_sub_field('linkki').'"
    data-answer="'.$vastausteksti.'"
    style="color:'.$vastausvari.';
    font-size:'.$vastausfontti.';
    border-radius:'.$vastausradius.';
    border: 1px solid '.$vastausreuna.';
    padding:'.$vastauspadding.';">'.
    $vastaussisalto
    .'</a>';
} else if(get_sub_field('ohjaustapa') == 'Jatkokysymys') { // Another question
    echo ' onclick="nextQuestion('.$id['id'].','.$question_id.','.get_sub_field('jatkokysymys').')">
    <div class="tammi-textchat-vastaus tammi-textchat-'.$id['id'].'-vastaus-'.$vastaus_count.'"
    data-answer="'.$vastausteksti.'"
    style="color:'.$vastausvari.';
    font-size:'.$vastausfontti.';
    border-radius:'.$vastausradius.';
    border: 1px solid '.$vastausreuna.';
    padding:'.$vastauspadding.';">'.
    $vastaussisalto
    .'</div>';
}
else if(get_sub_field('ohjaustapa') == 'Teksti') { // Text
    echo ' onclick="showText('.$id['id'].','.$question_id.','.count($text_boxes).')">
    <div class="tammi-textchat-vastaus tammi-textchat-'.$id['id'].'-vastaus-'.$vastaus_count.'"
    data-answer="'.$vastausteksti.'"
    style="color:'.$vastausvari.';
    font-size:'.$vastausfontti.';
    border-radius:'.$vastausradius.';
    border: 1px solid '.$vastausreuna.';
    padding:'.$vastauspadding.';">'.
    $vastaussisalto
    .'</div>';
    array_push($text_boxes,get_sub_field('vastaus_textbox'));
}
echo '</div>';
$vastaus_count++;
```

Kuva 31. Vastauksen sisällön esittäminen

Kun kaikki kysymyksen vastaukset on käyty läpi, auki jääneet div-elementit suljetaan ja kerätyistä tekstisisällöistä luodaan erilliset div-elementit, joihin käyttäjä voidaan tarvittaessa ohjata myöhemmin (Kuva 32). Ensin vastauksia ympäröivä div-elementti sekä kysymystä ympäröivä div-elementti molemmat suljetaan. Tämän jälkeen kaikki text_boxes-taulukkoon tallennetut alkiot käydään läpi, luoden jokaiselle oman div-elementin. Elementit luodaan vastaamaan piilotettua kysymystä, mutta lisäksi niiden id:ksi asetetaan kyselyn id:tä, kysymyksen id:tä ja alkion järjestysnumeroa hyödyntävä uniikki tunniste, ja sisällöksi asetetaan alkion sisältö. Lopuksi kyselyä ympäröivä div-elementti suljetaan. HTML-rakenteen ollessa valmis, funktio lopettaa ulostulon puskuroinnin ja palauttaa koko kyselyn ob_get_clean-funktion avulla.

```
114             echo '</div></div>';
115             for ($i = 0; $i < count($text_boxes); $i++) { // Create text boxes
116                 echo '<div
117                 class="tammi-textchat-kysymys-container tammi-textchat-kysymys-hidden"
118                 id="tammi-textchat-'.$id['id'].'-kysymys-'.$question_id.'-teksti-'.$i.'"
119                 style="display:none;
120                 color:'.$vastausvari.';">
121                 <h2>'.$text_boxes[$i].'</h2>
122                 </div>';
123             }
124         }
125         echo '</div>';
126     }
127     // end output buffering, grab the buffer contents, and empty the buffer
128     return ob_get_clean();
```

Kuva 32. Elementtien sulku ja funktion palautus

3.7 Monivalintakyselyn toiminta

Kysely pystyttäisiin nyt esittämään loppukäyttäjälle, mutta sille tuli vielä kehittää toiminnallisuus kysymysten ja sisältöjen välillä siirtymiseen. Tämä toiminnallisuus tuli kehittää JavaScriptillä, joten se sijoitettiin lisäosan tammi-textchat.js-tiedostoon.

Kyselyn HTML-rakenne toteutettiin tavalla, missä kaikki kysymyselementit, paitsi ensimmäinen kysymys, piilotetaan oletusarvoisesti näkyvistä. Toiminnallisuuden tuli siis piilottaa ja tuoda esiin elementtejä sen mukaan, mitä vastauksia käyttäjä valitsee.

Ensimmäisenä luotiin hideQuestion-funktio (Kuva 33), jolla pystyttäisiin piilottamaan aiempi kysymys, kun siirrytään jatkokysymykseen tai tekstisisältöön. Funktiolle määritettiin parametreiksi kyselyn id sekä kysymyksen id. Kysely etsii ensin sivulta kysymyksen div-elementin sen uniikin id:n avulla käyttäen JavaScriptin document-olion getElementById-funktiota. Tämän jälkeen funktio lisää elementille tammi-textchat-kysymys-hidden-luokan elementin classList-ominaisuuden toggle-funktiolla. Lopuksi funktio asettaa elementin tyyleiltään näkymättömäksi.

```
14  function hideQuestion(quizId, id) {
15      var question = document.getElementById("tammi-textchat-"+quizId+"-kysymys-"+id);
16      question.classList.toggle("tammi-textchat-kysymys-hidden");
17      question.style.display = "none";
18  }
```

Kuva 33. hideQuestion-funktio

Seuraavaksi tehtiin jo aiemmin jatkokysymyksiin ohjaavissa vastauksissa viitattu nextQuestion-funktio (Kuva 34), jolle määritettiin parametreiksi kyselyn, kysymyksen sekä jatkokysymyksen id:t. Funktio kutsuu ensin hideQuestion-funktiota, välittäen parametreiksi kyselyn ja nykyisen kysymyksen id:t, jotta nykyinen kysymys piilotetaan klikkauksen jälkeen. Tämän jälkeen sivulta etsitään jatkokysymyksen div-elementti sen uniikin id:n avulla käyttäen jälleen getElementById-funktiota. Seuraavaksi jatkokysymyksen elementin tammi-textchat-kysymys-hidden-luokka poistetaan elementin classList-ominaisuuden toggle-funktiolla. Lopuksi jatkokysymys asetetaan tyyleiltään näkyviin. Tällä tavalla aiempi kysymys saadaan piiloon ja se korvataan jatkokysymyksellä.

```
1  function nextQuestion(quizId, thisQ, nextQ) {
2      hideQuestion(quizId, thisQ);
3      var newQ = document.getElementById("tammi-textchat-"+quizId+"-kysymys-"+nextQ);
4      newQ.classList.toggle("tammi-textchat-kysymys-hidden");
5      newQ.style.display = "block";
6  }
```

Kuva 34. nextQuestion-funktio

Viimeisenä funktiona monivalintakyselyn toiminnallisuudelle luotiin aiemmin tekstisisältöön ohjaavissa vastauksissa viitattu showText-funktio (Kuva 35), jolle määritettiin parametreiksi kyselyn ja kysymyksen id sekä tekstisisällön järjestysnumero. nextQuestion-funktion tavoin showText-funktio piilottaa ensin nykyisen kysymyksen hyödyntäen hideQuestion-funktiota. Tämän jälkeen tekstisisällön div-elementti haetaan sivulta getElementById-funktiolla hyödyntäen kyselyn ja kysymyksen id:itä sekä sisällön järjestysnumeroa. Sisällön elementti asetetaan näkyviin korvaamaan piilotettu kysymys.

```
8  function showText(quizId, thisQ, thisA) {
9      hideQuestion(quizId, thisQ);
10     var textBox = document.getElementById("tammi-textchat-"+quizId+"-kysymys-"+thisQ+"-teksti-"+thisA);
11     textBox.style.display = "block";
12 }
```

Kuva 35. showText-funktio

3.8 Lyhytkoodin lisääminen

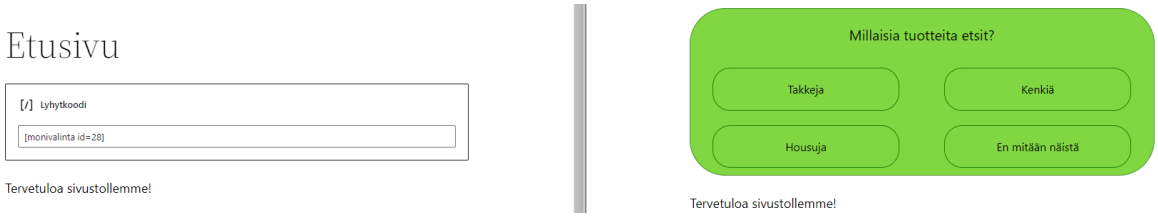
Monivalintakyselyiden rakenne ja toiminnallisuus olivat molemmat nyt valmiita, ja monivalintakyselyitä pystyttäisiin luomaan käyttövalmiina sivustolle. Lisäosaan piti kuitenkin vielä kehittää tapa, jolla sivuston ylläpitäjä pystyisi määrittämään, minne monivalintakysely asetettaisiin. Tätä varten päätettiin luoda lyhytkoodi, jolla monivalintakysely voitaisiin liittää haluttuun paikkaan.

Lyhytkoodin luomista varten tehtiin funktio tammi_textchat_init_shortcodes, joka liitettiin toimintona init-koukkuun (Kuva 36). Funktio luo lyhytkoodin WordPressin add_shortcode-funktiolla, asettaen lyhytkoodin tunnisteeksi monivalinta ja lyhytkoodin kutsumaksi funktioksi kyselyn HTML-rakenteen palauttavan tammi_textchat_create_monivalinta-funktion.

```
151  function tammi_textchat_init_shortcodes() {
152      add_shortcode('monivalinta', 'tammi_textchat_create_monivalinta');
153  }
154  add_action( 'init', 'tammi_textchat_init_shortcodes' );
```

Kuva 36. tammi_textchat_init_shortcodes-funktio ja sen liittäminen init-koukkuun

Tämän lyhytkoodin avulla monivalintakysely pystytään liittämään sivulle joko lisäämällä lyhytkoodi suoraan WordPressin sisältöön (Kuva 37) tai kutsumalla WordPressin do_shortcode-funktiota PHP-koodin sisällä. Lyhytkoodille tuli määrittää id-parametriksi sen monivalintakyselyn artikkeli-id, jolle haluttu kysely on rakennettu.



Kuva 37. Monivalintakyselyn lyhytkoodi ylläpitosivulla sekä julkisella sivulla

Lyhytkoodin ajateltiin olevan tarpeellista löytyä helposti ylläpitäjälle, ettei sitä tarvitsisi muistaa ulkoa. Tästä syystä päätettiin esittää monivalintakyselykohtainen lyhytkoodi sivuston

ylläpitopuolella sekä monivalintakyselyartikkeleiden listauksessa että monivalintakyselyartikkeleilla.

Ensin lisättiin lyhytkoodi monivalintakyselyartikkeleiden listausnäkymään. Lyhytkoodille tuli erikseen luoda sarake listausnäkymän taulukkoon sekä määrittää sarakkeen sisältö.

Sarakkeen luomista varten tehtiin funktio tammi_textchat_add_shortcode_column, joka liitettiin suodattimena WordPressin dynaamiseen manage_{$post_type}_posts_columns-koukkuun, jonka avulla mukautetun artikkelityypin listausnäkymän sarakkeita pystytään muokkaamaan (Kuva 38). Funktio ottaa koukulta vastaan parametrinä listausnäkymän sarakkeet, lisää sarakkeen Shortcode, ja palauttaa ne.

```
175  function tammi_textchat_add_shortcode_column( $columns ) {
176      $columns['shortcode'] = 'Shortcode';
177      return $columns;
178  }
179  add_filter( 'manage_tammi_textchat_posts_columns', 'tammi_textchat_add_shortcode_column', 5 );
```

Kuva 38. Sarakkeen luominen listausnäkymän taulukkoon

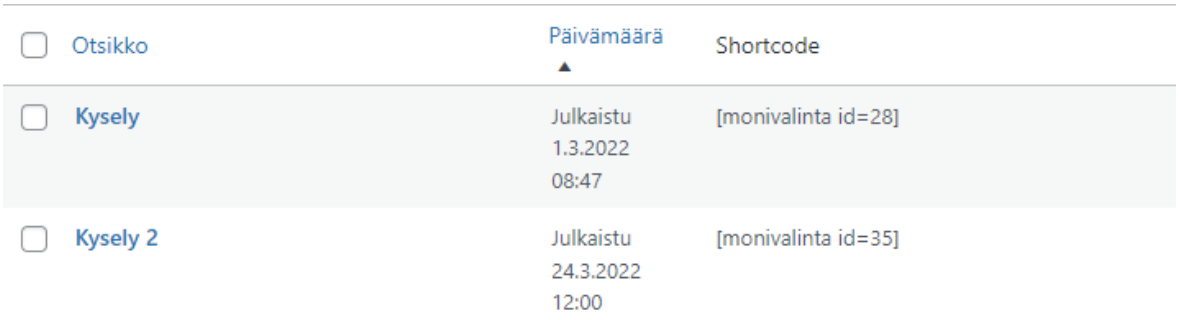
Sisällön lisäämiseksi sarakkeeseen tehtiin funktio tammi_textchat_shortcode_column_content, joka liitettiin toimintona WordPressin dynaamiseen manage_{$post->post_type}_posts_custom_column-koukkuun, jonka avulla sisältö pystytään lisäämään mukautetun artikkelityypin listausnäkymän sarakkeeseen (Kuva 39). Funktio ottaa koukulta vastaan sarakkeen sekä sen artikkelin id:n, jonka kohdalla ollaan. Jos sarake on lyhytkoodille luotu sarake, funktio asettaa sarakkeen sisällöksi lyhytkoodin sisällön kyseessä olevan artikkelin id:llä.

```
181  function tammi_textchat_shortcode_column_content( $column, $id ) {
182    if( 'shortcode' == $column ) {
183      echo '[monivalinta id='.$id.']';
184    }
185  }
186  add_action( 'manage_tammi_textchat_posts_custom_column', 'tammi_textchat_shortcode_column_content', 5, 2 );
```

Kuva 39. Sisällön lisääminen listausnäkymän sarakkeeseen

Monivalintakyselyiden listausnäkymään saatiin näin luotua erillinen sarake, jonka kautta sivuston ylläpitäjä pystyy helposti kopioimaan haluamansa monivalintakyselyn lyhytkoodin (Kuva 40). Koska funktiot kiinnitettiin monivalintakyselyiden mukautetulle artikkelityypille

dynaamisesti luotuihin koukkuihin, sarake ei myöskään näkyisi turhaan muilla artikkelityypeillä.



Kuva 40. Shortcode-sarake monivalintakyselyjen listausnäkymässä

Lopuksi lyhytkoodi tuli vielä lisätä näkyviin monivalintakyselyartikkelin ylläpitosivulla. Tätä varten luotiin funktio tammi_textchat_edit_page_shortcode, joka kiinnitettiin toimintona ylläpitosivujen ilmoitusten lisäämiseen tarkoitettuun admin_notices-koukkuun (Kuva 41). Jos käyttäjä on monivalintakyselyartikkelin ylläpitosivulla, luodaan tammi_textchat_shortcode_banner-funktio, joka liitetään toimintona edit_form_after_title-koukkuun, jonka avulla voidaan lisätä sisältö ylläpitosivulle heti otsikon alapuolelle. tammi_textchat_shortcode_banner funktio lisää otsikon alle elementin, joka sisältää kyseisen monivalintakyselyn lyhytkoodin (Kuva 42).

```
189  function tammi_textchat_edit_page_shortcode() {
190      $screen = get_current_screen();
191
192      if ( is_admin() && ($screen->id == 'tammi_textchat') ) {
193
194          function tammi_textchat_shortcode_banner() {
195              echo '<div class="postbox" style="background:#ff3363;color:#fff;"><div class="inside">';
196              echo 'Shortcode: [monivalinta id='.get_the_ID().']';
197              echo '</div></div>';
198          }
199          add_action( 'edit_form_after_title', 'tammi_textchat_shortcode_banner' );
200      }
201  }
202  add_action( 'admin_notices', 'tammi_textchat_edit_page_shortcode' );
```

Kuva 41. Lyhytkoodin esittäminen ylläpitosivulla

Kuva 42. Lyhytkoodi esitettynä artikkelin otsikon alla monivalintakyselyn ylläpitosivulla

3.9 Datan kerääminen

Lisäosaan tuli seuraavaksi kehittää datan kerääminen monivalintakyselyn kautta. Lisäosan suunnitteluvaiheessa päätettiin tallennettaviksi tiedoiksi: käytössä oleva kysely, käyttäjälle esitetty kysymys, käyttäjän valitsema vastaus sekä aikaleima. Tieto tuli tallentaa välittömästi vastauksen klikkaamisen jälkeen ja tiedot tuli olla mahdollista ladata sivustolta CSV-tiedostona.

Datan keräämistä varten tuli ensin luoda WordPressiin uusi tietokantataulu. Tämän jälkeen kehitettäisiin tietojen tallentaminen tietokantaan AJAXin avulla. Lopuksi tehtäisiin keino ladata kerätty data sivustolta.

3.9.1 Tietokantataulun luominen

Tietokantataulun luomista varten tehtiin funktio tammi_textchat_db_install, joka asetettiin suoritettavaksi, kun lisäosa aktivoidaan register_activation_hook-funktiolla (Kuva 43). Tällä tavalla tietokantataulu luotaisiin vain kerran lisäosan asennuksen yhteydessä.

Funktio alustaa ensimmäisenä globaalin wpdb-muuttujan, jonka avulla voidaan kommunikoida WordPressin tietokannan kanssa. Sitten funktio alustaa tietokantataulun nimen lisäten sen alkuun tietokannan etuliitteen sekä tietokannan käyttämän merkistön. Viimeisenä alustetaan vielä ajettava SQL-komento, joka asetetaan luomaan tietokantataulu

suunnitelluilla tallennettavilla tiedoilla sekä aiemmin määritetyillä taulun nimellä ja merkistöllä. Tämän jälkeen haetaan WordPressin upgrade.php-tiedosto ja ajetaan sen dbDelta-funktion avulla aiemmin luotu SQL-komento tietokantaan.

```
229  function tammi_textchat_db_install() {
230      global $wpdb;
231
232      $table_name = $wpdb->prefix . 'quizdata';
233
234      $charset_collate = $wpdb->get_charset_collate();
235
236      $sql = "CREATE TABLE $table_name (
237          answerid int NOT NULL AUTO_INCREMENT,
238          quiz int,
239          question varchar(255),
240          answer varchar(255),
241          td_timestamp varchar(255),
242          PRIMARY KEY  (answerid)
243      ) $charset_collate;";
244
245      require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
246      dbDelta( $sql );
247  }
248  register_activation_hook( __FILE__, 'tammi_textchat_db_install' );
```

Kuva 43. tammi_textchat_db_install-funktio ja sen liittäminen lisäosan aktivointiin

3.9.2 AJAX-toiminto

Tietojen tallentaminen tietokantaan vastauksen klikkaamisen yhteydessä välittömästi tuli kehittää AJAXin avulla. AJAX-toimintoa varten tuli kehittää sekä selaimessa suoritettava jQuery-koodi että palvelimella suoritettava PHP-koodi. Osa PHP-koodista AJAXia varten kuitenkin tehtiin jo aikaisemmin, kun lisäosan JavaScript-tiedostolle välitettiin AJAXille tarpeelliset arvot jo tiedoston liittämisen yhteydessä. Itse pyynnön käsittely PHP-koodin puolella tuli kuitenkin vielä kehittää jQuery-koodin lisäksi.

Ensimmäisenä kehitettiin AJAX-pyynnön lähettäminen jQueryllä (Kuva 44). Pyynnön lähettävä funktio kiinnitettiin sivuston latauduttua kyselyiden vastauksen klikkaamiseen. Funktio hakee ensimmäisenä vastauksen ympärillä olevan kyselyn id:n, kysymyksen kuvauksen sekä vastauksen kuvauksen. Tämän jälkeen funktio lähettää AJAX-pyynnön jQuery.ajax-funktiolla. Pyyntö kohdistetaan WordPressin admin-ajax.php-tiedostoon JavaScript-

tiedostolle välitetyn arvon kautta. Pyynnön sisältö lähetetään JSON-muodossa sisältäen kutsuttavan funktion, tiedostolle välitetyn nonce-tunnisteen, sekä tietokantaan tallennettavat tiedot.

```
jQuery(document).ready(function() {
    jQuery( ".tammi-textchat-vastaus" ).click(function() {
        var quizId = jQuery(this).parent().parent().parent().parent().attr("id").split("-");
        var questionText = jQuery(this).parent().parent().parent().children().data("question");
        var answerText = jQuery(this).data("answer");
        jQuery.ajax({
            type: "post",
            dataType: "json",
            url: tammi_textchat_js_object.ajaxurl,
            data: {
                action:`tammi_textchat_savedata`,
                security: tammi_textchat_js_object.nonce,
                quiz: quizId[2],
                question: questionText,
                answer: answerText
            },
            success: function(msg){
                console.log(msg);
            }
        });
    });
});
```

Kuva 44. Datan tallentamisen AJAX-toiminnon jQuery-koodi

Tämän jälkeen tuli vielä kehittää PHP-koodi AJAX-pyyntöjen käsittelyyn. Pyyntöjen käsittelyä varten luotiin funktio tammi_textchat_savedata (Kuva 45), joka liitettiin kahteen eri AJAX-koukkuun, jotta AJAX-toiminnot toimisivat sekä kirjautuneilla että kirjautumattomilla käyttäjillä. Funktion aluksi varmistetaan, että pyynnön lähettäjällä on asianmukainen nonce-tunniste. Tämän jälkeen alustetaan jälleen globaali wpdb-muuttuja ja asetetaan pyynnön välittämät tietokantaan tallennettavat tiedot muuttujiin. Tämän jälkeen aiemmin luotuun tietokantatauluun syötetään uutena rivinä pyynnön lähettämät tiedot ja lopuksi funktio suljetaan wp_die-funktiolla.

```php
// Save data to DB
function tammi_textchat_savedata() {
    check_ajax_referer( 'tammi-textchat-nonce', 'security' ); // Check Nonce
    do_action( 'tammi_textchat_click_answer' ); // Hook when clicking on answer
    global $wpdb;

    $quiz = ($_POST['quiz']);
    $question = ($_POST['question']);
    $answer = ($_POST['answer']);


    $table_name = $wpdb->prefix . 'quizdata';

    $wpdb->insert(
        $table_name,
        array(
            'quiz' => $quiz,
            'question' => $question,
            'answer' => $answer,
            'td_timestamp' => date("Y-m-d H:i:s")
        )
    );
    echo 'Saved to '.$table_name.': '.$quiz.'-'.$question.'-'.$answer.'-'.date("Y-m-d H:i:s");
    wp_die();
}
// Register savedata-function to admin-ajax
add_action( 'wp_ajax_nopriv_tammi_textchat_savedata', 'tammi_textchat_savedata' );
add_action( 'wp_ajax_tammi_textchat_savedata', 'tammi_textchat_savedata' );
```

Kuva 45. tammi_textchat_savedata ja sen liittäminen AJAX-koukkuihin

3.9.3 Tietojen hakeminen ja tulostus

Ylläpitäjälle tuli vielä luoda tapa ladata monivalintakyselyn keräämät tiedot tietokannasta. Tätä varten päätettiin luoda lisäosan asetussivu. Tälle asetussivulle pystyttäisiin lisäämään linkki tietojen lataamiseen sekä lisäosan asetukset, joita lisäosaan saatettaisiin jatkokehityksessä lisätä.

Asetussivun lisäämistä varteen tehtiin funktio tammi_textchat_create_menu, joka kiinnitettiin toimintona admin_menu-koukkuun (Kuva 46). Koukku suoritetaan juuri ennen ylläpitovalikon lataamista. Funktio luo asetussivun käyttämällä add_menu_page-funktiota. Asetussivun nimeksi asetetaan Kyselytulokset, käyttövaatimukseksi asetetaan ylläpitäjätason oikeudet ja sen sisältö määritetään tulostettavaksi tammi_textchat_settings_page-funktiolla.

```php
function tammi_textchat_create_menu() {
    add_menu_page(
        'Kyselytulokset',
        'Kyselytulokset',
        'administrator',
        __FILE__,
        'tammi_textchat_settings_page',
        plugins_url('/images/icon.jpg', __FILE__)
    );
}
add_action('admin_menu', 'tammi_textchat_create_menu');
```

Kuva 46. tammi_textchat_create_menu-funktio ja sen liittäminen admin_menu-koukkuun

Sen jälkeen tammi_textchat_settings_page-funktio luotiin asetussivun sisällön esittämistä varten. Funktio luo yksinkertaisen HTML-rakenteen, jossa tulostetaan otsikoksi Kyselytulokset ja sen alle linkki, joka lataa sivun uudelleen action-parametrillä download_csv_file.

Jos action-parametriksi on määritetty download_csv_file, suoritetaan CSV-tiedoston luomisen ja lataamisen suorittava toimintosarja (Kuva 47). Tietojen lataamista varten alustetaan aluksi jälleen globaali wpdb-muuttuja. Tietojen tallentamista varten luodaan ja avataan CSV-tiedosto fopen-funktiolla, minkä jälkeen WordPressin tietokantaan luodusta quizdata-taulusta haetaan kaikki rivit results-muuttujaan. Rivit käydään läpi yksi kerrallaan ja tallennetaan äskettäin luotuun tiedostoon fputcsv-funktiolla. Kun kaikki rivit on käyty läpi, tiedosto suljetaan fclose-funktiolla. Selaimelle välitetään tarpeelliset tiedot latausta varten kuten tiedoston tyyppi, ja sen käyttämä merkistö. Lopuksi tiedosto ladataan käyttäjälle readfile-funktiolla ja toimintosarja suljetaan.

```php
if ( isset($_GET['action'] ) && $_GET['action'] == 'download_csv_file' ) {

    global $wpdb;

    $wp_filename = "kyselydata_".date("d-m-y").".csv";
    $wp_file = fopen($wp_filename,"w");

    $table_name = $wpdb->prefix . 'quizdata';
    $results = $wpdb->get_results(

        "SELECT * FROM $table_name"

    );

    foreach($results as $row)
    {
        $wp_array = array(
            "Kysely"=>$row->quiz,
            "Kysymys"=>$row->question,
            "Vastaus"=>$row->answer,
            "Aikaleima"=>$row->td_timestamp
        );
        fputcsv($wp_file,$wp_array);
    }

    fclose($wp_file);
    header('Content-Encoding: UTF-8');
    header("Content-Description: File Transfer");
    header("Content-Disposition: attachment; filename=".$wp_filename);
    header("Content-Type: application/csv; charset=UTF-8");
    // UTF-8 BOM
    echo "\xEF\xBB\xBF";
    readfile($wp_filename);
    exit;
} ?>
```

Kuva 47. Toimintosarja CSV-tiedoston luomiseen ja lataamiseen

4 Yhteenveto

Työn tavoitteena oli suunnitella ja kehittää WordPress-lisäosa, jolla voidaan helposti lisätä monivalintakyselyitä toimeksiantajan kehittämille WordPress-sivustoille. Monivalintakyselyiden tarkoitus oli palvella sivuston loppukäyttäjiä sekä antaa sivuston ylläpitäjälle samalla tietoa käyttäjistään.

Lisäosa saatiin suunniteltua ja kehitettyä toimeksiannon mukaisesti noudattaen WordPress-lisäosan kehityksen parhaita käytänteitä. Lisäosan suunnittelussa ja kehityksessä otettiin erityisesti huomioon sivuston loppukäyttäjän sekä ylläpitäjän näkökulmat. Ylläpitäjän käyttöliittymässä panostettiin erityisesti joustavuuteen ja skaalautuvuuteen, jotta työkalua voitaisiin hyödyntää mahdollisimman monenlaisilla verkkosivustoilla. Kehittäjän näkökulma jätettiin vähemmälle huomiolle, koska suunnitteluvaiheessa päätettiin panostaa ensisijaisesti loppukäyttäjän sekä ylläpitäjän käyttökokemuksiin. WordPress-lisäosat ovat kuitenkin jo oletusarvoisesti nopea asentaa, joten kehittäjänkin käyttökokemuksen nähtiin olevan melko vaivatonta.

Lisäosaa voitaisiin jatkokehittää parantamalla ainakin sivuston ylläpitäjän sekä kehittäjän käyttökokemuksia. Ylläpitäjää varten voitaisiin luoda näkymiä datan tarkasteluun ja analysointiin sivuston sisäisesti, ettei ylläpitäjän tarvitsisi ladata ja käsitellä CSV-tiedostoa datan käsittelemiseksi. Kehittäjää varten kyselyn HTML-rakenne voitaisiin luoda käyttäen pohjatiedostoa, joka voitaisiin ylikirjoittaa sivuston teemasta. Pohjatiedoston ylikirjoittamisella kehittäjät voisivat helpommin muotoilla kyselyistä sivuston teeman mukaisia.

Jatkokehitystä varten työkalusta tulisi myös kerätä dataa kyselyiden käytöstä sekä palautetta sivustojen ylläpitäjiltä. Tätä varten lisäosa aiotaan pilotoida toimeksiantajan omilla verkkosivuilla.

Lähteet

Delicious Brains. Advanced Custom Fields for WordPress Developers. Viitattu 1.3.2022. Saatavissa https://www.advancedcustomfields.com/

Garret, J. 2005. Ajax: A New Approach to Web Applications. Adaptive Path. Viitattu 24.2.2022. Saatavissa https://web.archive.org/web/20150910072359/http://adaptive-path.org/ideas/ajax-new-approach-web-applications/

Mullenweg, M..2003. WordPress Now Available. WordPress.org. Viitattu 26.10.2021. Saatavissa https://WordPress.org/news/2003/05/WordPress-now-available/

Ravoof, S. 2022. Using AJAX and PHP in Your WordPress Site Creating Your Own Plugin. WP Guy News. Viitattu 2.3.2022. Saatavissa https://wpguynews.com/using-ajax-and-php-in-your-WordPress-site-creating-your-own-plugin/

Warner, A. 2009. The Biography of WordPress – With Matt Mullenweg. Mixergy. Viitattu 18.10.2021. Saatavissa https://mixergy.com/interviews/the-biography-of-WordPress-with-matt-mullenweg/

WordPress.com. Viitattu 29.3.2022. Saatavissa https://WordPress.com/

WordPress.org a. About. Viitattu 26.10.2021. Saatavissa https://WordPress.org/about/

WordPress.org b. Actions. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/hooks/actions/

WordPress.org c. AJAX Viitattu 24.2.2022. Saatavissa https://developer.WordPress.org/plugins/javascript/ajax/

WordPress.org d. Best Practices. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/plugin-basics/best-practices/

WordPress.org e. Creating Tables with Plugins. Viitattu 1.3.2022. Saatavissa https://codex.WordPress.org/Creating_Tables_with_Plugins

WordPress.org f. Database Description. Viitattu 1.3.2022. Saatavissa https://codex.WordPress.org/Database_Description

WordPress.org g. Filters. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/hooks/filters/

WordPress.org h. Hooks. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/hooks/

WordPress.org i. Introduction to Plugin Development. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/intro/

WordPress.org j. Lataa WordPress. Viitattu 1.3.2022. Saatavissa https://fi.WordPress.org/download/

WordPress.org k. Lisäosat. Viitattu 15.3.2022. Saatavissa https://fi.WordPress.org/plugins/

WordPress.org l. Managing Plugins. Viitattu 9.11.2021. Saatavissa https://WordPress.org/support/article/managing-plugins/

WordPress.org m. Options API. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/settings/options-api/

WordPress.org n. Plugin Basics. Viitattu 4.4.2022. Saatavissa https://developer.WordPress.org/plugins/plugin-basics/

WordPress.org o. Post Types. Viitattu 11.3.2022. Saatavissa https://WordPress.org/support/article/post-types/

WordPress.org p. Requirements. Viitattu 9.11.2021. Saatavissa https://WordPress.org/about/requirements/

WordPress.org q. Server Side PHP and Enqueuing. Viitattu 2.3.2021. Saatavissa https://developer.WordPress.org/plugins/javascript/enqueuing/

WordPress.org r. Settings. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/settings/

WordPress.org s. Settings API. Viitattu 26.10.2021. Saatavissa https://developer.WordPress.org/plugins/settings/settings-api/

WordPress.org t. Shortcodes. Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/shortcodes/

WordPress.org u. What is a Plugin? Viitattu 9.11.2021. Saatavissa https://developer.WordPress.org/plugins/intro/what-is-a-plugin/

WordPress.org v. WordPress Nonces. Viitattu 1.3.2022. Saatavissa https://codex.WordPress.org/WordPress_Nonces

WordPress.org w. wp_localize_script(). Viitattu 2.3.2022. Saatavissa https://developer.WordPress.org/reference/functions/wp_localize_script/

W3Techs. Usage statistics of content management systems. Viitattu 2.3.2022. Saatavissa https://w3techs.com/technologies/overview/content_management