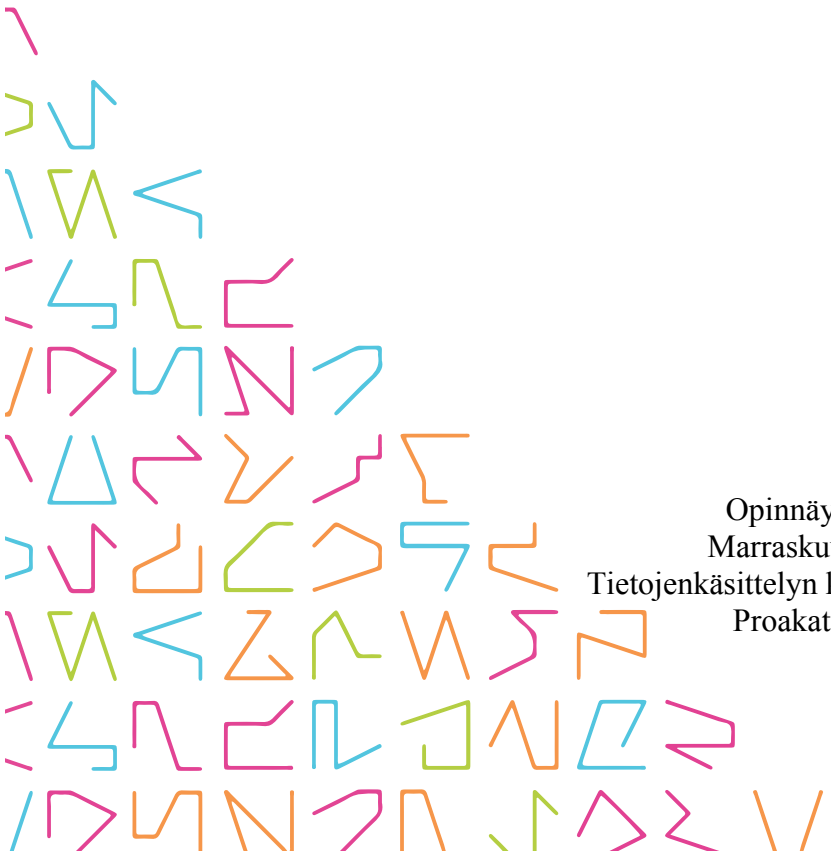
TAMPEREEN
AMMATTIKORKEAKOULU

WORDPRESS-TEEMAN KEHITYS DUSTPRESS-SOVELLUSKEHYKSELLÄ

Anttoni Lahtinen



Opinnäytetyö
Marraskuu 2016
Tietojenkäsittelyn koulutusohjelma
Proakatemia

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

LAHTINEN ANTTONI:
Wordpress-teeman kehittäminen Dustpress-sovelluskehyksellä

Opinnäytetyö 58 sivua, joista liitteitä 10 sivua
Marraskuu 2016

Työn toimeksiantajana on tamperelainen ohjelmistotalo Geniem, joka on erikoistunut enterprise-tason WordPress-palveluihin sekä Ios-, Android ja Wp-mobiilisovelluksiin. Geniem on kehittänyt Dustpress-sovelluskehystä Wordpress-teemojen kehitykseen keväästä 2015 alkaen. Se on julkaistu avoimen lähdekoodin GPLv2-lisenssillä.

Kesäkuussa 2016 julkaistiin ensimmäinen vakaa versio. Kehitys jatkuu yhä, ja tähtäimessä on kehittäjien mukaan muunmuassa asynkroninen sivunlataus Javascriptilla. WordPressin perinteisestä arkkitehtuurista poiketen Dustpress perustuu MVVM-malliin ja käyttää HTML-sivujen renderöintiin DustPHP-kirjastoa, joka perustuu DustJS-ulkoasumoottoriin.

Opinnäytetyössä tutustutaan Wordpressiin teemojen kehityksen näkökulmasta, Dustpressin ominaisuuksiin ja sen taustalla olevaan MVVM-arkkitehtuuriin. Tavoitteena on esitellä Dustpressiä sekä helpottaa kynnystä tutustua Dustpressiin niin Geniemin sisäisesti kuin yleisesti Wordpress-kehittäjien keskuudessa.

Kerätyn teoriatiedon sekä Wordpress-kehittäjäkyselyn pohjalta on tarkoitus suunnitella ja toteuttaa tutoriaali teeman kehittämiseen Dustpressillä. Tutoriaali julkaistaan verkossa, ja on sovelluskehyksen tavoin avoimesti käytettävissä.

Johtopäätöksenä voidaan todeta, että Dustpress tarjoaa toimivan tavan kehittää modulaarisia teemoja Wordpressille. Kun otetaan huomioon sen tarjoamat apufunktiot, välimuisti sekä tulevaisuuden kehitys, on kyseessä varteenotettava vaihtoehto sovelluskehykseksi teemojen kehittämiseen.

Raportissa ei tulla juurikaan perehtymään Wordpressiin julkaisujärjestelmänä tai PHP-ohjelmointikieleen, jolla teemoja kehitetään. Onhan sen lopputuotoksena syntyvä tutoriaali kohdistettu nimenomaan Wordpress-kehittäjille. Lukijalla tulisi siis olla jonkinlainen tietämys näistä asioista.

Asiasanat: Wordpress, teeman kehitys, php-ohjelmointi, verkkosivutuotanto

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Computer Science

LAHTINEN ANTTONI
Wordpress Theme Development with Dustpress Framework

Bachelor's thesis 58 pages, appendices 10 pages
November 2016

The commissioner of this thesis was a company called Geniem located in Tampere which specialises in enterprise level Wordpress services and mobile applications for Ios, Android and WP. Since spring 2015 Geniem has developed Dustpress Framework for Wordpress theming. It is published under the open source GPLv2 license.

In June 2016 the first stable release (v.0.4.0) was launched. Development is continuing and there are plans to add asynchronous page loading with Javascript. Dustpress differs from the original architecture of Wordpress and is based on the MVVM development model. It uses the DustJS based DustPHP library to render the HTML-pages.

This thesis examines the Wordpress theme development, Dustpress properties and the underlying MVVM model. The aim was to introduce Dustpress and lower the bar to start using it both inside Geniem as well as among other Wordpress developers

A tutorial for developing with Dustpress will be created on basis of theoretical background research and the data obtained through a Wordpress developer questionnaire. The tutorial will be published on the web and like the framework it will be freely available.

In conclusion, Dustpress offers a good way to develop modular themes for Wordpress. When considering in the helper functions, caching and future development one can count Dustpress as a noteworthy framework candidate for theme developing.

This thesis does not cover Wordpress or PHP basics because the target group of the tutorial will be Wordpress developers. The reader should have some previous knowledge of these subjects.

Keywords: Wordpress, theme development, php programming, web developing

SISÄLLYS

1 JOHDANTO

Työn tausta

Wordpress täytti tämän vuoden toukokuussa 16 vuotta. Se on maailman suosituin sisällönhallintajärjestelmä yli 26% osuudella verkon sivuista. Oma matkani Wordpressin parissa on alkanut valmisteemojen asennuksesta ja koodin varovaisesta muokkailusta noin viisi vuotta sitten.

Monen kehittäjän tarina on alkanut samalla tavoin. PHP:ta ei osata koodata riviäkään ja yrityksen ja erehdyksen menetelmällä kurkitaan ja sörkitään Wordpress-teemojen pellin alle. Lueskellaan dokumentaatiota ja kysellään viisaammilta. Wordpressin pariin houkuttelee sen viiden minuutin asennus, selkeä ylläpitonäkymä ja valtava yhteisö. Jälkimmäisen ansiosta lähes jokaiseen ongelmaan löytyy apua jo valmiiksi foorumeilta.

Wordpress ei ole ongelmaton alusta ja matalan kynnyksen lähdöllä sivuja kehittämään on seurauksensa. Moni Wordpress-sivu onkin valmisteemojen ja huonosti koodattujen lisäosien liitos. Tämä oli totta myös omissa tuotoksissani.

Valmisteemojen jälkeen tulivat starttiteemat, joista mainittakoon Rootsin Sage. Tässä vaiheessa koodin kirjoittaminen sujui jo paremmin. Vaihtaessani Geniemille viime keväänä tutustuin Dustpressiin ja teeman kehittämiseen sen avulla.

Wordpressin toinen pääkehittäjistä Matt Mullenweg (2016) esitteli tämän vuoden Wordcamp Europessa Wordpressiä alustaksi, jota voi teemojen ja lisäosien avulla laajentaa moneen eri suuntaan; Wordpress itsessään tarjoaa vain perustan. Teeman kehittäjänkin on siis varauduttava turvautumaan erilaisien lisäosien apuun.

Yksi Wordpressin ongelmista on sen monoliittinen arkkitehtuuri: teemojen ulkoasukoodin sekaan kirjoitetaan sisällön hakeva logiikka. Tähän ongelmaan etsittiin ratkaisua Geniemillä, ja vuoden 2015 keväällä alkoi Dustpressiksi nimetyn sovelluskehyksen kehitystyö. Tavoitteena oli luoda MVVM-arkkitehtuuriin perustuva tapa toteuttaa teemoja.

Kuulin itse Dustpressistä ensimmäistä kertaa kesän 2015 Wordcamp Finlandissa Tampereella, jossa Geniemin työntekijät olivat esittelemässä suunnitelmiaan. Silloin en

osannut aavistaa päätyväni myöhemmin työskentelemään Geniemillä Dustpressin parissa.

Työn toimeksiantaja on tamperelainen ohjelmistotalo Geniem. Geniem tuottaaa enterprise-tason WordPress-palveluja sekä Ios-, Android ja Wp-mobiilisovelluksia.
Geniem on ketterä ohjelmistotalo, joka näkyy esimerkiksi Scrum-projektihallinnan viitekehyksen käytössä. Geniem on Suomessa 14. nopeimmin kasvanut teknologiayritys 2016. (Geniem 2016.)

Työn tavoitteesta

Wordpress on 16 vuotta vanha julkaisujärjestelmä. Tämä on web-tekniikoiden mittapuulla lähes ikuisuus. Pitkäikäisyyden ja suosion ansiosta alustan ympärille on kuitenkin syntynyt valtava yhteisö. Tämä yhteisö ei ole piiloutunut vain nimimerkkien taakse ja väittele verkossa, vaan kokoontuu säännöllisesti yhteen paikkakuntatasolla (Wordpress Meetupit), valtakuntatasolla (Wordcamp Finland) sekä maiden rajat ylittävästi (Wordcamp Europe ja Wordcamp US). Tampereella kokoontuu kuukausittain noin viidestä viiteentoista kehittäjää – tämän vuoden Wordcamp Europe keräsi yhteen yli 2 000 kehittäjää ympäri maailmaa.

Yhteisö on tiivis ja kommunikoi aktiivisesti pääosin Slackin ja Twitterin välityksellä. Wordpress on avointa lähdekoodia, jonka seurauksena työkaluja jaetaan ja kehitetään kilpailevienkin yritysten välillä. On kaikkien hyöty, että yhteisessä käytössä olevat työkalut kehittyvät. Olisikin hyvä, jos myös Dustpress otettaisiin käyttöön Geniemin ulkopuolella, sillä useammat käyttäjät tarkoittavat enemmän bugikorjauksia sekä muuta ulkopuolista kehitystä. Käyttäjiä ei tietääkseni vielä ole ainakaan kovin monia ja tähän haasteeseen etsin ratkaisua.

Toinen haaste on Geniemin sisäinen. Dustpressin työjärjestys eroaa MVVM-mallin takia myös Wordpressin perinteisestä teemankehityksestä, jonka seurauksena uudet työntekijät joutuvat opettelemaan tekniikan aloittaessaan työt Geniemillä. Kasvavassa yrityksessä aikaa kuluu Dustpressin käytön koulutukseen

Tämän työn tavoitteena onkin siis madaltaa kynnystä tutustua Dustpressiin. Työn seurauksena syntyy tutoriaali, jossa opastetaan teeman kehittämiseen Dustpressillä. Tästä on hyötyä kaikille, jotka haluavat alkaa käyttää Dustpressiä.

Minulla on myös henkilökohtainen tavoite: työllä jaan jo oppimaani ja perehdyn lisää Dustpressin saloihin. Koen sen olevan vahva työkalu Wordpress-teemankehitykseen ja haluaisin sen käytön leviävän myös Geniemin ulkopuolelle. Olen myös saanut Wordpress-yhteisöltä paljon ja tämä on tapani antaa takaisin.

Teoriaosuudesta

Työn teoriaosuudessa perehdyn Wordpressin teemankehitykseen sekä toimintaan teemojen näkökulmasta. Lisäksi käyn läpi Dustpressin ominaisuuksia ja MVVM-teoriaa.

Wordpressin teoriaan päälähteenäni toimii kirja Professional Wordpress: Design and Development (Damstra, Stern, & Williams 2013). Kirjan kirjoittajat ovat Wordpress-ammattilaisia ja kirjasta onkin tehty useita eri painoksia. Käyttämäni painos on vuodelta 2013, joka on pitkä aika nopeasti kehittyvien verkkoteknologieoiden maailmassa. Osa asioista on ehtinyt muuttua Wordpress-kehityksessäkin tänä aikana, jonka takia käytän tukenani myös paljon verkkoartikkeleita. Voin sanoa tuntevani Wordpress-maailmaa suhteellisen hyvin ja käytän lähteinäni esimerkiksi Wordcampeissa puhuneita arvostettuja kehittäjiä.

Dustpressin lähdemateriaali on hieman ongelmallisempaa. Suoraa alustasta kertovia lähteitä ei ole Dustpressin oman Readme-tiedoston lisäksi tarjolla. Siksi olen yrittänyt hyödyntää esimerkiksi LinkedinIn lähdeaineistoa Dustjs-ulkoasumottorista puhuttaessa. Lisäksi haastattelin kahta Dustpressin pääkehittäjää, ja tarkistin heiltä jotain asioita liittyen kirjoittamaani.

Yleisesti ottaen olen varsin tyytyväinen käyttämääni lähdeaineistoon. Saatan ottaa joidenkin arvostamieni kehittäjien sanan liiankin kritiikittömästi vastaan, mutta toisaalta luotan heidän sanaansa.

2 WORDPRESS

2.1 Wordpressin historia

Vuonna 2001 korsikalainen ohjelmoija, Michel Valdrighi, kehitti PHP-pohjaisen blogialustan, jolle hän antoi nimen *b2*. Blogialustan ympärille kasvoi yhteisö, joka lähti kehittämään alustaa yhteistyössä Michelin kanssa. Alustan kehitys joutui kuitenkin ongelmiin Michelin huonojen koodiratkaisujen takia. (McKeown & Stevens 2014.)

Huonoista ratkaisuista huolimatta kehittäjäyhteisö kasvoi. Vuonna 2003 Michel joutui vetäytymään kehityksestä henkilökohtaisten syiden vuoksi. Kehittäjäyhteisössä mukana ollut Matt Mullenweg ehdotti b2-blogialustan koodin kopioimista uudeksi projektiksi, mikä oli mahdollista sen GPL-lisenssin ansiosta. Mike Little vastasi Mattin ehdotukseen luvaten lähtevänsä mukaan kehitykseen. Huhtikuussa 2003 b2:sta perustettiin uusi haara, jolle annettiin nimi Wordpress. (McKeown & Stevens 2014.)

Tammikuussa 2006 Wordpress oli saanut mukaansa jo teemat, ja silloin tehdyn Googlen verkkokatsauksen perusteella Wordpressiä käytti 0,8 % verkkosivuista. (McKeown & Stevens 2014.) Alustan suosion lisääntyminen alkoi houkutella mukaan myös yrittäjiä, ja verkkoon syntyi erilaisia teemamarkkinapaikkoja. (McKeown & Stevens 2014.)

Kymmenessä vuodessa Wordpressin kasvu on ollut valtavaa: W3Techs-sivuston (2016) tutkimuksen mukaan elokuussa 2016 26,6 % kaikista maailman verkkosivuista käytti alustanaan Wordpressiä. Wordpress onkin kehittynyt blogialustasta sisällönhallintajärjestelmäksi, jolla voidaan toteuttaa niin sivustoja kuin verkkokauppojakin. Toinen Wordpressin pääkehittäjistä, Matt Mullenweg (2016), puhui visiostaan nähdä Wordpress avoimen verkon käyttöjärjestelmänä - alustana, jonka päälle voidaan rakentaa vieläkin enemmän.

2.2 Teemat

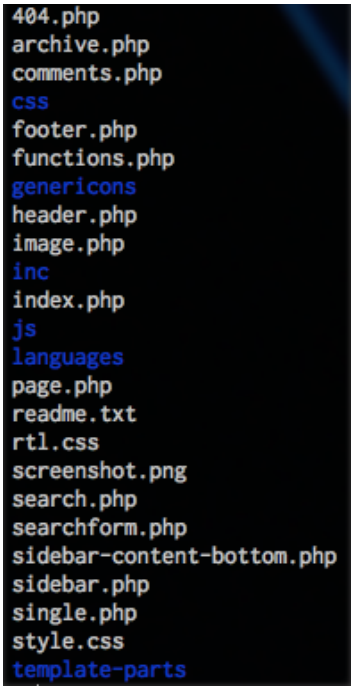
Kuten aiemmin todettiin, on Wordpress kasvanut blogialustasta sisällönhallintajärjestelmäksi. Tämä näkyy myös sen tarjoamissa ominaisuuksissa. Se tarjoaa WYSIWYG-tekstieditorin (What You See Is What You Get), jonka avulla voidaan kirjoittaa sisältöä sivuille sekä artikkeleihin. Lisäksi tarjolla on mediakirjasto, käyttäjähallinta, kommentointi sekä teemajärjestelmä, joka mahdollistaa sivuston ulkoasun vaihtamisen. (Wordpress.org 2016.) Teemat siis määrittävät sen, miten sisällönhallintajärjestelmään syötetty sisältö esitetään käyttäjälle. (Damstra ym. 2013.) Kun puhutaan verkkosivun kehittämisestä Wordpressillä, viitataankin lähes aina teeman kehittämiseen.

2.2.1 Teeman rakenne

Damstra ym. mukaan (2013) teeman voidaan katsoa koostuvat neljänlaisista elementeistä. Ulkoasutiedostot ovat PHP-tiedostoja, jotka hakevat sisällön tietokannasta. Tiedostot kääntyvät HTML-tiedostoiksi Wordpressiä suorittaessa. Myös osa CSS-koodista saatetaan generoida ulkoasutiedostoissa, vaikka tätä ei suositellakaan. Teeman tuleekin sisältää vähintään index.php-ulkoasutiedosto.

Toinen teeman ydinelementeistä on CSS-koodi. Wordpress-manuaalin teemankehitysohjeen mukaan (Wordpress Codex 2016a.) teemaan täytyy sisällyttää vähintään style.css-tiedosto, jotta se voidaan ottaa käyttöön sivustolla. Tiedostoa käytetään ulkoasumäärittelyiden lisäksi myös teeman nimeämiseen sekä muiden teeman tietojen antamiseen.

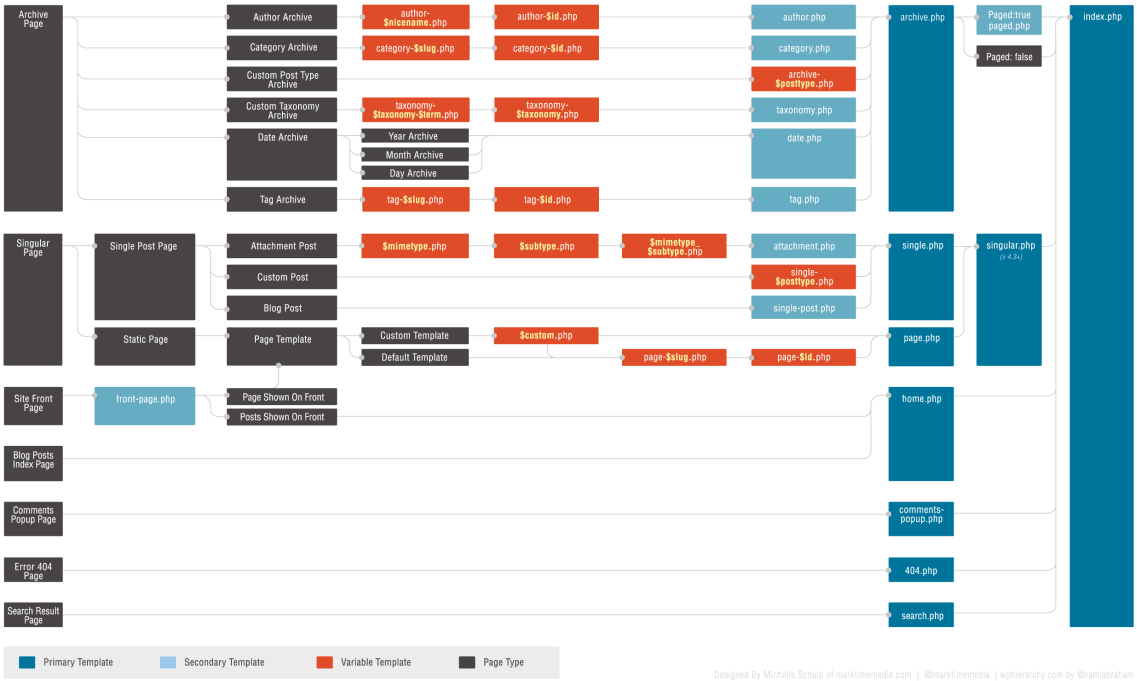
Näiden kahden ydinelementin lisäksi Damstra ym. (2013) mainitsevat teeman elementeiksi myös kuvat ja muut resurssit, kuten Javascript-tiedostot, sekä teeman toiminnalle olennaiset lisäosat, joita käytetään yleensä teeman hallintaan. Siinä missä kuvat ja Javascript-tiedostot on yleensä pakattu teeman kanssa samaan kansioon, tulee lisäosat ladata erikseen. Jotkin teemat vaativat yhden tai useamman lisäosan lataamisen toimiakseen. Kuvassa 1 on listattu Wordpressin vuoden 2016 *Twentysixteen*-oletusteeman sisältö.



KUVA 1: Wordpressin vuoden 2016 oletusteeman kansion sisältö

2.2.2 Teemojen ulkoasuhierarkia

Ennen tietokantahakua parsitaan Wordpressin suorituksen aikana http-pyynnöstä kyselyparametrit, jotka sijaitsevat osoitteessa kysymysmerkin jälkeen. Huomiona mainittakoon, että yleensä Wordpress-sivustolla on käytössä mukautettu osoiterakenne artikkelin nimen (engl. *pretty permalinks)* mukaan, jolloin parametrit on eroteltu kauttaviivoilla .htaccess-tiedoston muutoksen seurauksena, eikä kysymysmerkki ole näkyvissä. Kuvassa 2 on nähtävissä Wordpressin ulkoasuhierarkia, joka etenee vasemmalta spesifeistä ulkoasuista oikealle kohti index.php:tä.



KUVA 2: Wordpressin ulkoasuhierarkia (Theme Handbook 2016)

Parametrien avulla päätellään sivu, jota ollaan hakemassa, minkä jälkeen ulkoasuhierarkian mukaisesti etsitään aktiivisen teeman kansiosta oikeaa ulkoasutiedostoa. Hierarkian mukaan valitaan sopivin ulkoasutiedosto. Mikäli sopivaa ei löydy, päädytään lopulta hierarkian mukaan käyttämään index.php-ulkoasutiedostoa. Valittua ulkoasua käytetään HTML-koodin generointiin sekä sisällön hakemiseen. (Theme Handbook 2016.)

2.2.3 Valmisteemat

Wordpress-teeman kehitystä ei tarvitse suinkaan aina aloittaa nollasta. Valmiita teemoja on saatavilla lukuisasti niin ilmaiseksi kuin maksua vastaan. Wordpress.org:in teemakirjastosta löytyy ilmaisia teemoja kirjoitushetkellä yli 2 200. (Wordpress.org 2016.) Suositusta teemamarkkinapaikasta Themeforestista on ostettavissa yli 7 500 erilaista teemaa. (Themeforest 2016.)

Osa valmisteemoista on valmistettu tiettyä tarkoitusta varten. Esimerkiksi Auto Trader (Themefuse 2016.) on autokauppoja varten suunniteltu teema. Se tarjoaa hyviä työkaluja autojen esittelyyn, mutta mikäli sivuston omistaja päättää alkaa automyynnin lisäksi vuokrata veneitä, ei autokauppateema välttämättä sovellu tähän. Monipuolisista vaihtoehdoistakin huolimatta ulkoasun muokkaaminen on rajallista.

Kaikista ostetuin valmisteemakategoria on monikäyttöteemat (engl. *multi-purpose themes*.) (Themeforest 2016.) Teemat soveltuvat nimensä mukaisesti moneen ja niiden ulkoasut ovat laajasti muokattavissa ilman koodiin koskemista. Monikäyttöisyytensä takia niissä ei kuitenkaan tule mukana erikoisominaisuuksia - aikaisemman esimerkin autokauppias saattaa jäädä kaipaamaan työkaluja autojen ominaisuuksien esittelyyn.

Valmiit teemat sopivat projektiin, jossa budjetti on pienempi, ja kustomoinnista ollaan sen takia valmiita joustamaan. (Tolvanen 2015a.) Toisena vaihtoehtona ovat projektit, joissa halutaan tehdä itse, eikä taitoa tai aikaa ohjelmoinnille ole. Myös verkkosivun suunniteltu ikä tulee ottaa huomioon: kaksi kuukautta verkossa pyörivään laskeutumissivuun ei välttämättä kannata investoida nelinumeroisia summia.

2.2.4 Valmisteemojen kritiikkiä

Valmista teemaa hankkiessa tulee ottaa huomioon, että alkuperäinen kehittäjä ei välttämättä sitoudu käyttötukeen tai päivittämään teemaa tulevaisuudessa, mikä voi johtaa yhteensopivuusongelmiin esimerkiksi lisäosien kanssa. Lisäksi saattaa olla kallista palkata ohjelmoija lisäämään ominaisuuksia valmiiseen teemaan, jonka alkuperäinen koodi on hänelle tuntematonta. (WPbeginner 2016.)

Wordpress-kehittäjä Teemu Suoranta (2016) puhui Tampereen Wordpress Meetup -tapahtumassa sivustojen pitkästä eliniästä. Suorannan mukaan sivuston valmistaminen alkaa kauan ennen verkkosivukoodin tai grafiikan tuottoa suunnittelulla. Sivuston kehitys alkaa sivuston tavoitteen, kohdeyleisön sekä tavoitteen onnistumisen seurannan suunnittelemisella. Kun lähdetään liikkeelle valmisteemalla, on ulkoasuvaihtoehtoja vain muutama ja suunnittelu sen vuoksi paljon rajatumpaa.

Toiveiden mukaan kustomoitu sivusto, joka suunnitellaan sivun käyttötarpeeseen, on pidemmällä aikavälillä lähes aina kannattavampaa toteuttaa alusta lähtien itse kuin lähteä muokkaamaan valmista koodia. Näin mukaan saa myös ohjelmoijan, joka tuntee koodin ja kykenee vastaamaan sen jatkokehityksestä. Suomessa kustomoitujen Wordpress-sivustojen budjetit lähtevät yleensä noin 5 000 eurosta ylöspäin, vaikka 2 000 eurollakin on mahdollista saada jotain aikaan. (Tolvanen 2015b.)

2.2.5 Lapsiteemat ja WYSIWYG

Damstra ym. (2013) esittelee kaksi tapaa tuottaa teema Wordpress-sivuille. Ensimmäinen vaihtoehto on käyttää Wordpressin tukemaa lapsiteema-ominaisuutta. Lapsiteemat perivät isäntäteemansa koodin. Lapsiteeman koodi suoritetaan isäntäteeman koodin jälkeen, ja näin esimerkiksi lapsiteemassa olevat CSS-säännöt näkyvät verkkosivulla. Lapsiteemaa suositellaankin, mikäli on löytänyt itselleen sopivan teeman, jota haluaa muokata vain hieman.

Tarkoitukseen valmistettuja isäntäteemoja on tarjolla niin maksullisina kuin ilmaisinakin. Suosituin isäntäteema on Studiopressin kehittämä Genesis. Se tarjoaa WYSIWYG-editorin sivupohjien rakentamista varten sekä monia asetuksia, joilla sivua voi kustomoida. Studiopressin verkkosivuilla oleva kauppa tarjoaa myös lukuisia lapsiteemoja Genesikselle. (Studiopress 2016.)

Vaikka kehittäjät vierastavat WYSIWYG-editoreita, toteaa Williamson (2016), että niille on paikkansa ja käyttäjäkuntansa, joka on vielä valmis maksamaan tuotteesta. Samassa artikkelissa Williamson kuitenkin listaa myös WYSIWYG-sivunrakennuslisäosien ongelmia. Kritiikin mukaan kyseiset lisäosat ovat standardisoimattomia, joten teema- ja lisäosakehittäjät eivät voi ottaa suunnittelussaan huomioon niiden tuomia muutoksia. Tämä saattaa aiheuttaa yhteensopivuusongelmia teeman tai lisäosan sekä ulkoasulisäosan välillä.

Williamson (2016) kohdistaa kritiikkinsä myös sivunrakennuslisäosien toimintatapaan. Yleisesti editorit sijoittavat sisällön joukkoon Wordpressin lyhytkoodeja (engl. *shortcodes*), jotka lisäosa kääntää HTML-elementeiksi sivun generoinnissa Wordpressin suorituksen aikana. Kun lisäosa otetaan pois käytöstä, jäävät lyhytkoodit sisällön sekaan tekstimuodossa niiden poistaminen saattaa olla erittäin työlästä. (Williamson 2016.) Käytännön ongelmien lisäksi ulkoasusääntöjen tallentaminen sisällönjoukkoon on hyvien ohjelmointitapojen vastaista, kuten seuraavassa luvussa tullaan toteamaan. Yleisesti hyväksytyn periaatteen mukaan ulkoasun säännöt ja sisältö tulisi pitää erillä toisistaan. (engl. *Separation of presentation and content*.)

McCollin (2014) vertailee artikkelissaan *Choosing a Wordpress Theme Framework – the Ultimate Guide* eri isäntäteemoja. Hän kehuu Genesistä myös kehittäjäystä-

välliseksi: muutoksia on helppo tehdä myös koodin avulla, sillä teemaa on mahdollista käyttää myös sovelluskehyksenä ilman valmiita lapsiteemoja.

Ohjelmoijaystävällisyydessään Genesiksen ohittaa McCollinin arvion mukaan vain Theme Hybridin *Hybrid Core*. Kyseessä on Dustpressin toimintaa lähempänä oleva sovelluskehys, ei varsinainen isäntäteema, eikä se sisällä lainkaan WYSIWYG-sivunrakennuskomponettia. Hydrid Core tarjoaa esimerkiksi murupolut, mikrodatan integraation sekä numeroidun sivutuksen. Nämä kaikki ominaisuudet ovat käytössä yksittäisillä funktioilla, ja sovelluskehys hoitaa loput. (Theme Hybrid 2016.)

Isäntäteeman suurin etu on kirjoitettavan koodin vähäisyys. Mikäli sopiva teema löytyy, voidaan välttyä kirjoittamasta riviäkään koodia. Lapsiteemalla kehittäjä voi nopeasti poistaa tai lisätä joitain ominaisuuksia tai visuaalisia elementtejä teemasta. Jos isäntäteeman kehittäjä on sitoutunut kehittämään teemaansa Wordpressin päivittyessä ja koodipohja on hyvä, saattaa isäntäteeman hankinta tulla kaikista edullisimmaksi vaihtoehdoksi.

Tulee kuitenkin muistaa, että toiveet verkkosivua kohtaan saattavat muuttua tulevaisuudessa. Ei tarvita kovinkaan suuria ominaisuusmuutoksia, kun valtavat määrät isäntäteeman tuntemattomia koodirivejä kääntyvät sivuston kehittäjälle taakaksi, ja olisi ollut nopeampaa kehittää koko sivusto ilman valmista ratkaisua. (Damstra ym. 2013 luku 9: What to look for in a starter theme.)

2.2.6 Starttiteemat

Mikäli halutaan kehittää teema alusta asti, on lapsiteeman vaihtoehtona starttiteema (engl. *Starter theme*). Voidaan ajatella, että kaikkia GPL-lisenssin alaisia teemoja voidaan käyttää starttiteemoina. Teeman voi ladata vaikkapa Wordpress.orgin teema-arkistosta ja käyttää sen koodia oman projektinsa pohjana. On olemassa kuitenkin myös juuri teemankehityksen pohjalle suunniteltuja starttiteemoja. Damstra ym. (2013) nostavat esiin Automatticin Underscores-starttiteeman.

Underscores tarjoaa teeman kehittäjälle valmiin 404-sivupohjan, responsiivisen navigaationskriptin sekä HTML-pohjia. Teeman mukana tulee myös esimerkiksi CSS-sääntöjen yhtenäistämiskirjasto, joka pyrkii tasoittamaan eri selainten aukkoja

(Stewart 2012.) Starttiteemat pyrkivät uudelleenkäyttämään koodia tarjoamatta kuitenkaan suuria muutoksia Wordpressin toimintatapaan. (Damstra ym. 2013.)

Esimerkkinä muutoksista mainittakoon esimerkiksi Rootsin Sage-starttiteema. Sage muokkaa sivupohjien rakennetta lisäämällä base.php-tiedoston, joka toimii kaikkien sivupohjien alustana. Tähän ulkoasutiedostoon ladataan erilaisia ulkoasupalasia sivusta riippuen. Näin vältetään esimerkiksi otsakkeen ja alapalkin lisääminen useisiin sivupohjiin. (Roots 2016.)

2.2.7 Sovelluskehykset

Wordpress-teemankehityksen kontekstissa sovelluskehys on joukko määrityksiä ja työkaluja, joiden avulla teemoja kehitetään. Määritelmä on epätarkka, sillä sovelluskehyksiä on olemassa moniin eri tarkoituksiin. Kyseessä voi olla esimerkiksi ulkoasumoottori tai joukko uudelleenkäytettäviä funktioita.

Sovelluskehys eroaa kehitystyökalusta siinä, että sen tarjoama koodipohja on teeman perustana, ja sen mukanaolo on ehtona teeman toiminnalle. Se ei kuitenkaan itsessään ole teema, sillä se ei sisällä valmiiksi ulkoasutiedostoja. Joissakin sovelluskehyksissä saattaa tulla mukana oma starttiteema. (Tadlock 2010.)

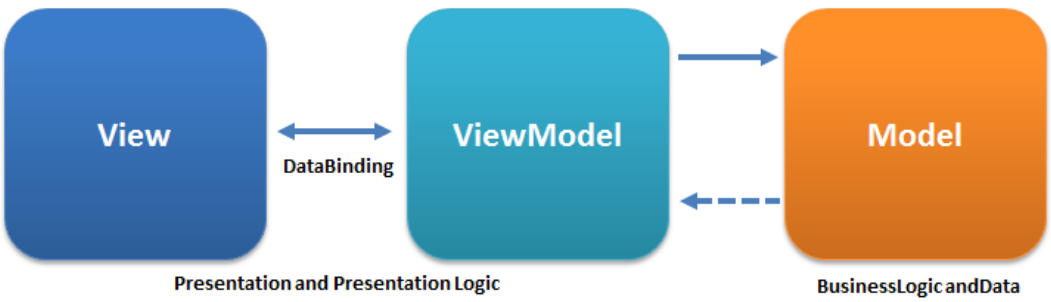
3 Miksi MVVM?

Haasteiden eriyttäminen (Separation of Concerns) on sovelluskehityksessä käytetty periaate. Ohjelmisto jaetaan arkkitehtonisesti erilaisiin osiin, joista jokainen suorittaa omaa tehtäväänsä. (Laplante 2007, 85) Tämän niin kutsutun hajota ja hallitse -strategian pohjalta on syntynyt erilaisia ohjelmistoarkkitehtuureita. Näitä ovat esimerkiksi Model View Controller (MVC), Model View Parser (MVP) sekä tässä luvussa käsiteltävä MVVM. (Vice & Shujaat 2012, 7.)

Ohjelmistoja on tietenkin mahdollista luoda myös ilman arkkitehtuureita: tätä kutsutaan monoliittiseksi suunnitteluksi. Siinä esimerkiksi datan käsittelyä ei eroteta sen esitystavasta laisinkaan. Monoliittisestä tavasta toteuttaa ohjelmistoja voi kuitenkin seurata monenlaisia ongelmia. Vicen & Shujaatin (2012, 11–15, 28.) mukaan esimerkiksi koodin uudelleenkäytettävyys ja päivitys, projektin rakenne ja skaalautuvuus kärsivät. Testaamisesta tulee hankalaa, kun koodi ei ole jaoteltu selkeästi. Epäselvän koodipohjan päälle on myös todella hankala rakentaa uusia ominaisuuksia.

3.1 MVVM-arkkitehtuurin esittely

Model View ViewModel (MVVM) -arkkitehtuuri on Microsoftin Windows Presentation Foundation (WPF) Data Services -tiimin kehittämä ohjelmistoarkkitehtuuri. Voidaan sanoa, että MVVM-arkkitehtuuri perustuu Model View Presenter -arkkitehtuuriin. Se korjaa joitakin MVC- sekä MVP-arkkitehtuurin puutteita säilyttäen samalla niiden vahvuudet. (Vice & Shujaat 2012, 80.) Kuvassa 3 esitelllään MVVM-arkkitehtuuri graafisesti.



KUVA 3: MVVM-arkkitehtuurin graafinen esitys (Wikimedia Commons 2016)

Mallin tarkoituksena on kerätä liiketoiminnalliseen kerrokseen kuuluva data. Data on tallennettuna esimerkiksi tietokantaan, eikä se ota millään tavalla kantaa sen esitys- tai käsittelytapaan. (Vice & Shujaat 2012, 11 – 15.)

Näkymämalli hakee datan mallista välimuistiin ja käsittelee sen näkymälle sopivaan muotoon. Näkymämalliin sisällytetään kaikki ohjelman logiikka ja datankäsittely. Se vastaa myös käyttäjän toimien välittämisestä takaisin mallille ja mallissa olevan datan ajantasaisuudesta. (Vice & Shujaat 2012, 11 – 15.)

Kuten sekä MVC:ssä että MVP:ssä, myös MVVM-arkkitehtuurissa **näkymä** vastaa datan esittelystä toimien ohjelman käyttöliittymänä. Tämän lisäksi näkymä välittää tiedot käyttäjän syötteestä Model-osiolle. Lisäksi näkymä voi vastata näkymien ja näkymämallien yhdistämisestä toisiinsa. (Vice & Shujaat, 2012, 11 – 15.)

3.3 MVVM:n hyödyt

Vice & Shujaat (2012, 14) esittävät MVVM-arkkitehtuurin tuovan ohjelmointiin lukuisia hyötyjä. Kun koodi on modulaarisesti tuotettua, on näitä pienempiä ohjelmakokonaisuuksia helpompi testata käyttötesteillä. Koodia voi myöskin käyttää uudestaan, kun eri osat toteuttavat selkeitä tehtäviä. Esimerkiksi listaparsija voi parsia kaikki listat, eikä vain tiettyä listaa suoraa näkymään koodattuna. Tämä vaatii toki enemmän suunnittelua ohjelmiston kehitysvaiheessa.

Eriytetyt osiot voidaan toteuttaa erikseen esimerkiksi luomalla näkymämalli, joka palauttaa oikeantyylistä puppudataa näkymään. Suunnittelijat voivat rakentaa ohjelmiston näkymää samalla, kun varsinainen malli ja näkymämalli vielä valmistuvat. (Vice & Shujaat 2012, 25)

3.4 MVVM:n kritiikkiä

Yksi arkkitehtuurin pääkehittäjistä, John Gossman (2006), nostaa esiin myös kritiikkiä MVVM-arkkitehtuuria kohtaan blogikirjoituksessaan *Advantages and disadvantages of M-V-VM*. Gossmanin mukaan saattaa olla ylilyönti ottaa MVVM käyttöön projektissa, jonka käyttöliittymä on todella yksinkertainen.

Gossman jatkaa, että isommissa projekteissa näkymämallin suunnittelu voi olla aluksi mahdotonta, ja siihen täytyy tehdä isojakin rakennemuutoksia projektin edetessä. Lisäksi ohjelman tarvitseman välimuistin suuri koko saattaa muodostua ongelmaksi.

4 DUSTPRESS

Dustpress on sovelluskehys Wordpress-teemojen kehitykseen. Se on avointa lähdekoodia (GPLv2) ja on saatavilla Github-verkkopalvelusta. Dustpress muuttaa Wordpressin toimintaa eriyttämällä datankäsittelyn ja sen esityksen MVVM-mallin mukaisesti. Dustpress käyttää Dust.js-sivupohjamoottoriin pohjautuvaa DustPHP-kirjastoa HTML-tiedostojen renderöintiin. Lisäksi Dustpress laajentaa sekä Wordpressin että Dust.js:n toiminnallisuuksia joukolla funktioita niin datan hakemista kuin esittämistäkin varten. (Dustpress Readme 2016.)

Haastattelussa toinen Dustpressin kehittäjistä, Miika Arponen (2016), kertoi kehityksen saaneen alkunsa, kun PHP- ja HTML-koodin keskenään sekoittamisesta haluttiin siirtyä pois. Yrityksellä oli tässä vaiheessa jo kokemusta Linkedinin kehittämästä Dust.js-ulkoasumoottorista, ja siihen pohjautuva Dustphp-kirjasto tarjosi helpon mahdollisuuden hyödyntää tätä kokemusta myös teemakehityksessä.

4.1 Asennus ja käyttöönotto

Dustpressin ohjeistuksessa suositellaan asennusta Composerin avulla. (Dustpress Readme 2016.) Composer on PHP-ohjelmointikielelle suunniteltu paketinhallintaohjelmisto, joka mahdollistaa PHP-ohjelmistopakettien hallinnan komentoriviltä. Käyttäjän päivittäessä määrittelemänsä paketit, tarkistaa Composer myös niiden taustalla olevien pakettien päivityksen. Näin käyttäjän ei tarvitse huolehtia lukuisista päivityksistä. (Composer 2016.)

Helpomman päivityksen lisäksi Composeria voi hyödyntää ohjelmistopakettien automaattisessa käyttöönotossa. Dustpress tukee tätä ominaisuutta, eli Composerin kautta asennettaessa sitä ei tarvitse sisällyttää projektiin käsin. Vaihtoehto Composer-asennukseen on tiedostojen lataaminen Githubista manuaalisesti. Tiedostojen lataamisen jälkeen Dustpress tulee sisällyttää projektiin PHP:n require-funktiolla. Sisällyttämisen jälkeen sovelluskehys aktivoidaan kutsumalla *dustpress*-funktiota. (Dustpress Readme 2016.)

4.2 Mallit ja näkymät

Dustpress perustuu aiemmin esiteltyyn MVVM-arkkitehtuuriin. Mallit ovat PHP-tiedostoja, joissa data haetaan ja käsitellään. Mallit laajentavat Dustpressin Model-luokkaa, ja niiden latauksessa hyödynnetään Wordpressin ulkoasuhierarkiaa. Dustpress lataa automaattisesti päämallin http-kutsun parametrien mukaisesti, kuten aiemmin on esitelty (alaluku 2.2.2). Ladattavan päämallin lisäksi mallissa voidaan käyttää alamalleja, jotka muodostavat uudelleenkäytettäviä komponentteja päämallienkäyttöön. laajentamalla näkymälle välitettävää dataa. (Siltala 2016.)

Dustpressin ydinluokka välittää datan näkymälle muodostaen MVVM-mallin mallinäkymän osan. Mallin julkisten funktioiden palauttama data kootaan globaaliin olioon, joka lähetetään näkymälle sivun renderöinnissä. Ajon aikana on mahdollista suorittaa myös funktioita, joissa dataa ei palauteta ollenkaan. Tämä tapahtuu määrittelemällä funktiot yksityisiksi (engl. *private*) tai suojatuiksi (engl. *protected*.) (Dustpress Readme 2016.)

4.3 Osamoduulit (Submodules)

Dustpress pyrkii noudattamaan *Dont repeat yourself* (DRY) -periaatetta. Periaatteen mukaan kaiken systeemin informaation tulee esiintyä siinä vain kerran, ja muuten systeemin sisällä viitataan tähän yhteen esiintymään. (Thomas 2003.) Dustpressin tapauksessa DRY-periaate ilmenee osamoduuleissa, joita voidaan käyttää uudelleen eri malleissa ja näkymissä. Dustpress helpottaa myös modulaarista ohjelmointia vähentäen näin koodin toistoa. (Dustpress Readme 2016.)

Kuvassa 4 on esimerkki etusivumallista, jonka init-funktiossa liitetään otsakkeen ja ala- ja sivupalkin malliin.

```php
<?php
/*
Template name: Frontpage
*/
class PageFrontpage extends \DustPress\Model {

    public function init() {

        $this->bind_sub("Header");
        $this->bind_sub("Sidebar");
        $this->bind_sub("Footer");
    }
}
```

KUVA 4: Etusivuun liitettävät alanäkymät

Toimintatapa muistuttaa hyvin paljon Wordpressin omia include-funktioita. Funktioilla, kuten *get_footer*, *get_header* ja *get_sidebar,* haetaan ulkoasupohjaan tarvittavia ulkoasun osia. (Wordpress Codex 2016b.)

Dustpressin *bind_sub*-funktioon voidaan syöttää toinenkin parametri osamoduulin nimen lisäksi. Toisella parametrilla moduulille voidaan lähettää dataa listamuodossa. Tämä data on saatavilla moduulin lisäksi myös sen alamoduuleissa. Moduuleiden liitto voi tapahtua myös esimerkiksi ehtolauseen sisällä, ja osamoduuleita voidaan liittää toisiinsa. Nämä ominaisuudet mahdollistavat monimutkaistenkin mallien rakentamisen modulaarisesti. (Dustpress Readme 2016.)

Kuvassa 5 on esimerkki Header-osamoduulin mallista. Mallin funktio palauttaa sivuston nimen, ja tieto on käytettävissä kaikissa malleissa, joihin se on liitetty.

```php
<?php
class Header extends \DustPress\Model {

    public function SiteTitle() {

        $blog_title = get_bloginfo();
        return $blog_title;
    }

}
```

KUVA 5: Osamoduuliesimerkkinä yksinkertainen otsakemalli

Dust-ulkoasutiedostot toteuttavat MVVM-arkkitehtuurin näkymiä (*view*.) Dust-ulkoasutiedostot käännetään selaimelle ymmärettävään HTML-muotoon DustPHP-kirjaston avulla. (Dustpress Readme 2016.) DustPHP pohjautuu Linkedinin kehittämään DustJS-ulkoasumoottoriin. (Retz 2013.) DustPHP:n kehittäjä (Retz 2013) kertoo käyttöoppaassa, että hänen tavoitteenaan oli kehittää tehokas ja ytimekäs Javascriptiin perustuva ulkoasumoottori.

DustPHP korvaa dust-tiedoston muuttujaviittaukset niitä vastaavalla datalla. Dataa, joka näkymälle annetaan renderöintiä varten, kutsutaan kontekstiksi (engl. *context*.) Kuvassa 6 näkyy viittaus title-muuttujaan sekä silmukka, joka käy läpi names-listan tulostaen kaikki sen name-muuttujat. Kuvassa 7 esitetään kolme eri datarakennetta, joita DustPHP osaa parsia: kaikkien kontekstien tuloste on sama lista nimiä. (Retz 2013.)

```
{title}
<ul>
{#names}
  <li>{name}</li>{~n}
{/names}
</ul>
```

KUVA 6: Dust-tiedostot koostuvat HTML-tageista ja JSON-syntaksin mukaisista muuttujaviittauksista

```
$context = [
    'title' => 'Famous People',
    'names': [
        ['name' => 'Larry'],
        ['name' => 'Curly'],
        ['name' => 'Moe']
    ]
];
```

```
$context = (object)[
    'title' => 'Famous People',
    'names': [
        (object)['name' => 'Larry'],
        (object)['name' => 'Curly'],
        (object)['name' => 'Moe']
    ]
];
```

```
$context = json_encode('{
    "title": "Famous People",
    "names": [
        { "name": "Larry" },
        { "name": "Curly" },
        { "name": "Moe" }
    ]
}');
```

KUVA 7: Vasemmalta oikealle sama data esitettynä listana, objekteina ja JSON-olioina

DustJS etsii muuttujia ensin kontekstin ylätasolta ennen siirtymistä alatasolle. Mikäli muuttujaviittausta ei löydy alatasolta silmukoinnin yhteydessä, palataan takaisin ylätasolle. (DustJS-manuaali 2016) Tämän seurauksena esimerkiksi kuvan 6 koodissa voisimme määritellä kuvan seitsemän datapuuhun muuttujan *lastname* ja käyttää sitä names-silmukan sisällä tulostaen jokaiselle nimelle lastname-muuttujan arvon.

DustJS tarjoaa myös erilaisia apufilttereitä ja -funktioita datan tulostamiseen. Nämä hyödylliset koodinpätkät ovat mukana DustPHP:ssä (Retz 2013) ja tätä kautta käytössä myös Dustpressissä. (Dustpress Readme 2016.)

4.4 Apufiltterit (Filters)

Filttereitä DustPHP:ssä on kaiken kaikkiaan yhdeksän. Filtterit lisätään muuttujaviittauksen perään aaltosulkeiden sisään ja pystyviivalla erotettuina. Esimerkkinä filttereistä mainittakoon h-filtteri, joka estää HTML-koodin siistimisen (engl. *escaping*) ennen sen tulostusta. DustPHP siistii kaiken koodin automaattisesti ennen tulostusta Cross-site-scripting-hyökkäysten ehkäisemiseksi. (Retz 2013.) Filtteriä käytetään Dustpressissä paljon, sillä esimerkiksi sivujen sisältökentissä on Wordpressin WYSIWYG-editorin jäljiltä HTML-tunnisteita.

4.5 Apufunktiot (Dust Core Helpers)

Apufunktiot tuovat dust-tiedostoihin kevyttä logiikkaa (engl. *less logic*.) Tyypillisesti MVVM-mallissa näkymä ei sisällä lainkaan logiikkaa. (Vice & Shujaat 2012, 28.) Luvussa Logic in Templates (suom. *Logiikka näkymissä*) Retz (2013) myöntää olevansa samaa mieltä siitä, että liiketoimintalogiikka (engl. *business logic*) eli datan käsittely ei

kuulu näkymien tehtäväksi. Hän kuitenkin argumentoi, että datan esitystapaan liittyvä kevyt logiikka voidaan toteuttaa näkymätasolla.

Retzin (2013) mukaan mallissa sijaitseva liiketoimintalogiikka sekoittuu liiaksi, mikäli yksinkertaiset toiminnot, kuten taulukon tiettyjen rivien värittäminen, toteutetaan ehtolauseilla mallissa ja välitetään muuttujien avulla näkymälle. Dust tarjoaakin mahdollisuuden kevyeen logiikkaan ja luottaa ohjelmoijaan, jotta saatavilla olevilla ehtolauseilla ei toteuteta liian monimutkaisia logiikkarakenteita.

Esimerkiksi tietyn muuttujanviitauksen puuttuessa Dustille annetusta datasta, voidaan näkymään tarjota vaihtoehtoinen tuloste {:else}-merkinnällä. Kuvassa 8 tulostetaan array-listan data-jäsenet, tai listan ollessa tyhjä tulostetaan {:else}-merkinnän sisältö. (Retz 2013.)

```
<ul>
{#array}
  <li>{data}</li>
{:else}
  <p>Array is empty!</p>
{/array}
</ul>
```

KUVA 8: else-apufunktion käyttö.

Else-apufunktiota voidaan käyttää myös Dustin tarjoamien vertailuoperaattoreiden kanssa. Esimerkkinä tästä on eq-apufunktio, jolla verrataan lukujen tai kirjainjonojen yhdenkaltaisuutta. Kuvassa 9 vertaillaan name-muuttujan arvoa ”Tim”-kirjainjonoon ja tulostetaan ”Your name is Tim.” vertailun ollessa tosi tai muussa tapauksessa teksti ”Your name is not Tim.”

```
{@eq key=name value="Tim"}
    Your name is Tim.
{:else}
    You are not Tim.
{/eq}
```

KUVA 9: eq-apufunktiolla testataan name-muuttujan arvo

4.6 Välimuisti

Verrattuna pelkkään Wordpressin suoritukseen, vaatii Dustpressin suorittaminen enemmän resursseja. Erityisesti näkymien kääntäminen HTML-tiedostoiksi on iso operaatio. Ongelman helpottamiseksi Dustpress hyödyntää Wordpressin lyhytaikaista tallennusta. (engl. *transient cache*.) Lyhytaikaisessa tallennuksessa Wordpress tallentaa wp_options-tietokantaan dataa. Tälle datalle voidaan asettaa vanhenemisaika, jonka jälkeen se poistetaan tietokannasta. (Dustpress Readme 2016.)

Dustpress hyödyntää tätä ominaisuutta tallentaen automaattisesti kaikkien ulkoasutiedostojen käännetyt versiot. Tällä vältetään sivujen alusta asti kääntäminen. Tallennetulle datalle ei aseteta vanhenemisaikaa ollenkaan, sillä tarkoituksena on, että tuotantopuolella ei tehdä muutoksia näihin tiedostoihin. Kehitysympäristössä tallentaminen voidaan estää filtterin avulla, tai välimuisti voidaan nollata vaikkapa WP-CLI-aputyökalulla. (Dustpress Readme 2016.)

Sivuston käyttäjälle lähetettävät valmiit HTML-sivut tallennetaan myös automaattisesti. Erona ulkoasutiedostoihin on se, että HTML-näkymän sisällöstä ja koodista muodostetaan välimuistitallennukselle avain, jolloin sisällön muuttuessa myös välimuistiin tallennettu HTML-tiedosto päivitetään. (Dustpress Readme 2016.)

Automaattisten ulkoasutallennusten lisäksi Dustpress tarjoaa mahdollisuuden tallentaa myös mallien palauttama data välimuistiin. (Dustpress Readme 2016.) Näin kehittäjän on mahdollista tallentaa vaikkapa ulkopuolisesta lähteestä haettua dataa omalle palvelimelleen väliaikaisesti.

5 KYSELY WORDPRESS-KEHITTÄJILTÄ

5.1 Kyselytutkimuksen teoriaa

5.1.1 Lomakkeen rakenne

Lomakesuunnittelussa on tärkeää ottaa huomioon kyselyn kohderyhmä. Kuinka paljon vastaajalla on aikaa, halua tai taitoja vastata kyselyyn. Lisäksi on tärkeää huomioida, miten kysely ollaan toteuttamassa. (Tampereen Yliopisto 2010.)

Lomaketta suunniteltaessa kannattaa ottaa huomioon kyselyn pituus. Liian pitkä lomake saattaa ajaa vastaajia pois. On parempi saada monia vastauksia muutamiin kysymyksiin, kuin muutamia vastauksia moniin kysymyksiin, jotta tulokset ovat relevantteja. Pituuden lisäksi kyselyn graafinen ulkoasu vaikuttaa vastaajan mielipiteisiin vahvasti. Mikäli lyhyehkökin kysely vaikuttaa täyteen ahdetulta, ei siihen välttämättä vastata. (Tampereen Yliopisto 2010.)

5.1.2 Hyvä kysymys?

Kysymykset lomakkeessa kannattaa järjestää siten, että alkuun laitetaan muutamia lämmittelykysymyksiä: kysymyksiä joihin on helppoa vastata. Kyselyn kannalta irrelevantti tieto kannattaa kuitenkin jättää kysymättä. Kyselyissä tulee myös välttää anonymiteettiä vähentäviä kysymyksiä mahdollisimman hyvin, sillä ne saattavat herättää negatiivisia tuntemuksia vastaajassa. (Tampereen Yliopisto 2010.)

Kyselyä valmisteltaessa tulee miettiä myös kysymysten tarkkuustasoa. Yksi yleisimmistä kysymyksistä liittyy vastaustapaan. Onko kyselyssä avoimet kysymykset vai suoritetaanko se monivalintana? Vastaajan on helpompi vastata monivalintakysymyksiin ja vastauksia on myös helpompi käsitellä. (Tampereen Yliopisto 2010.)

Avoimet kysymykset saatetaan jättää tyhjiksi tai niiden vastaukset saattavat poiketa kyselynlaatijan odotuksista. Toisaalta avoin kysymys mahdollistaa laajemman vastaus-

vaihtoehdon ja mikäli esimerkiksi kohderyhmän voi olettaa olevan kirjallisesti aktiivi, on avointen kysymysten käyttö perusteltua. (Tampereen Yliopisto 2010.)

5.1.3 Kysymysten sisältö

Kysymysten sanojen asettelu vaikuttaa suuresti sen vastauksiin. Asettelu voi olla usein johdatteleva, jos siihen sisällytetään tunteita herättäviä sananvalintoja. Tätä voidaan pyrkiä välttämään esimerkiksi kysymällä kysymys usealla eri tavalla. tai sisällyttämällä vastausvaihdoiksi vaihtoehtoja, jotka eivät sulje toisiaan pois. (Tampereen Yliopisto 2010.)

Mikäli kysymyksissä on vastausvaihtoehdot, kannattaa kyselijän pohtia esimerkiksi "en osaa sanoa" ja "en tiedä" -vaihtoehtojen käyttöä. Niitä ei tulisi antaa vastaajille liian herkästi, sillä vastaaja saattaa hieman hankalan kysymyksen kohdatessaan vastata vain nopeasti ”en tiedä” vaihtoehdon. Toisaalta monta kertaa valittu ”en osaa sanoa”- vaihtoehto saattaa paljastaa ongelmia kysymyksessä. Tämä voi olla hyväkin tulos, jos sitä verrataan tilanteeseen, jossa vastaajat eivät ymmärrä kysymystä, mutta joutuvat silti vastaamaan jonkin muun vaihtoehdon. (Tampereen Yliopisto 2010.)

5.2 Kyselyn taustaa ja tavoitteet

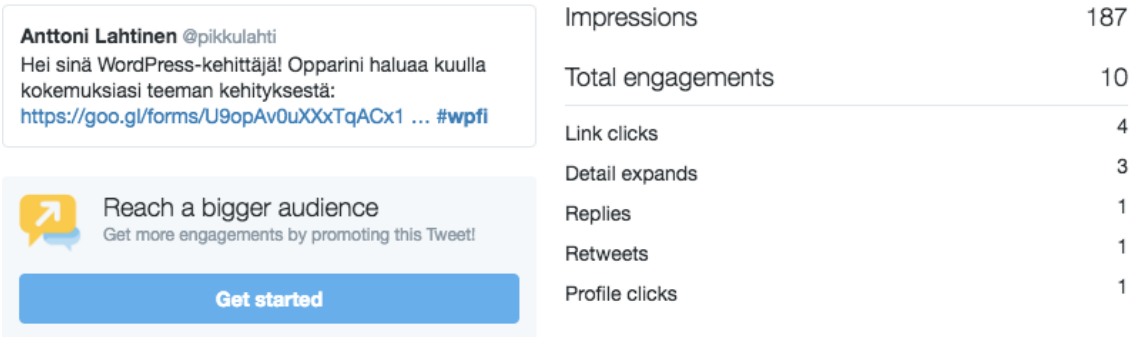
Tutoriaaliin haluttiin mukaan ajankohtaisuutta ja käytännönläheisyyttä. Niiden pohjalta toteutettiin Wordpress-kehittäjille kysely, jonka tarkoituksena oli tuoda esiin todellisia ongelmia ja pohtia niihin ratkaisuja. Dustpress-tutoriaalissa voidaan sitten ottaa huomioon kehittäjien teemankehityksen ongelmat sekä ottaa paremmin kantaa siihen, minkälaisia asioita oppaaseen kannattaa sisällyttää ja mitä jättää siitä pois.

Kysely oli luontevaa jakaa Suomen Wordpress -yhteisön Slackissa, sillä työn kieli on suomi ja Slack-kanava on suomalaisten Wordpress-ammattilaisten kokoontumispaikka. Kuvassa 10 on kuvankaappaus viestistä, jolla kyselylinkki jaettiin Slackiin.

KUVA 10: Kyselyn viesti Slackissa

Slack on erilaisille yhteisöille tarkoitettu viestintäpalvelu, joka mahdollistaa reaaliaikaisen viestinnän eriaiheisissa kanavissa. Myös Suomen Wordpress-kehittäjäyhteisöllä on Slackissa oma kanavansa, jossa keskustellaan WordPressiin liittyvistä aiheista, kuten kehityksestä, ongelmanratkaisusta, teemoista, lisäosista ja käännöksistä. Yhteisön pääkanavalla on kirjoitushetkellä 344 jäsentä, ja keskustelua käydään aktiivisesti. Jäseneksi pääsee tilaamalla sähköpostiosoitteeseensa kutsun. Linkkiä tähän jaetaan Wordpress Suomen sivuilla sekä erilaisissa yhteisön tapahtumissa. (Wordpress Suomi 2016.)

Kysely jaettiin myös Twitterissä, jossa kyselyn tekijän profiilia seuraa noin parikymmentä Wordpress-kehittäjää. Twiitin tietojen perusteella sitä kautta linkki avattiin kuitenkin vain neljä kertaa, kuten kuvassa 11 näkyy.



KUVA 11: Twitterissä jaetun linkin statistiikat

Kyselyn tavoitteena oli siis selvittää, minkälaisia työkaluja teemankehitykseen käytetään ja miksi. Toisena tavoitteena oli saada kartoitettua ainakin muutama ongelma, joka liittyy nimenomaan kehitysympäristöön. Näin raportissa voitaisiin tutkia, ratkaisisiko Dustpress näitä ongelmia.

Esimerkiksi ongelma *asiakkaan budjetti on liian pieni* voi helpottua koodin uudelleenkäytettävyyden mahdollistavalla työkalulla. *Asiakas ei vastaa puheluihini* tyyppiset ongelmat eivät koske kehitysympäristöä, joten toiveena oli, etteivät kaikki vastaukset olisi tällaisia.

Kyselyn viimeisenä tavoitteena oli selvittää, millä tavalla kehittäjät haluavat vastaanottaa tietoa uusiin työkaluihin tutustuessa. Tätä tietoa aiotaan hyödyntää oppaan tekemisessä.

5.3 Kyselyn sisältö

Kyselyssä oli yhdeksän kysymystä. Kolmella ensimmäisellä kysymyksellä pyrittiin määrittelemään vastaajan kokemusta Wordpress-kehityksestä sekä sen, työskentelevätkö he yksin vai tiimissä.

5.3.1 Vastaajan profilointi

Ensimmäisellä kysymyksellä selvitettiin vastaajan työnkuvaa. Työnkuva vaikuttaa muun muassa siihen, miten aktiivisesti vastaaja kehittää teemoja. Työntekijä tehnee työtä teemojen parissa harrastajaa enemmän. Tämä vaikuttanee positiivisesti vastausten reliabiliteettiin.

1. Kehitätkö Wordpress-teemoja *(valitse yksi)*
a) Työntekijänä
b) Freelancerina
c) Harrastuksena
d) Muu, mikä?

Toisella kysymyksellä selvitettiin kehittäjän työnkuvaa. Yksin työskentelevä kehittäjä on yleensä vastuussa laajemmasta osa-alueesta teemankehityksessä, kun taas tiimissä työskentelevä saattaa keskittyä vaikkapa tiettyyn vaiheeseen teeman kehityksessä.

2. Kehitätkö Wordpress-teemoja *(valitse yksi)*
a) Yksin
b) Osana tiimiä

Kolmas kysymys liittyy vastausten reliabiliteettiin. Työkokemus lisää haastateltavan luotettavuutta, ja viisi vuotta teemojen kanssa työskennellyt työntekijä onkin lähes aina luotettavampi lähde, kuin alle vuoden teemojen parissa puuhastellut harrastaja.

3. Kuinka pitkään olet kehittänyt teemoja? *(valitse yksi)*
a) Alle vuoden
b) 1-2 vuotta
c) 3-5 vuotta
d) 5-10 vuotta
e) 10 vuotta tai enemmän

5.3.2 Aineistoa keräävät kysymykset

Kysymykset 4–9 keskittyivät varsinaisen aineiston keräämiseen. Kysymykset 5–9 olivat avoimia, jotta vältyttiin rajaamasta mitään vastausta pois vaihtoehdoilla. Vapaasti vastaaminen mahdollisti myös laajemman vastauksen antamisen vastaajan näin halutessa. Tältä osin aineistoa keräävät kysymykset muistuttivatkin kvalitatiivista strukturoitua haastattelua.

Neljäs kysymys ohjasi vastaajan vastaamaan joko 5. tai 6. kysymykseen.

4. Käytätkö teemojen kehitykseen starttiteemaa/frameworkia? *(valitse toinen)*
Kyllä/Ei

Viides ja kuudes kysymys selvittivät syitä starttiteeman tai frameworkin käyttöön tai käyttämättömyyteen. Jotta kysymysten määrä pysyisi pienenä, kysyttiin viidennessä kysymyksessä sekä työkalun nimi että syyt sen käyttöön. Viidennestä kysymyksestä harkittiin tehtävän monivalintakysymys, mutta vaihtoehtojen suuren määrän takia päädyttiin avoimeen kysymykseen.

5. Jos vastasit kyllä: Mitä starttiteemaa/frameworkia käytät ja miksi? *(avoin vastaus)*
6. Jos vastasit en: Miksi et? *(avoin vastaus)*

Seitsemäs kysymys pyrki kartoittamaan vastaajan teemankehityksen työjärjestystä. Tämäkin kysymys pidettiin avoimena. Vaihtoehtona olisi ollut listata työvaiheet ja pyytää lisäämään niiden eteen numero, mikä jätettiin kuitenkin tekemättä, jottei jotain työvaihetta rajata pois. Avoimet vastaukset kiinnostivat myös siksi, että vastaaja saattaa paljastaa teemankehitysprosessistaan asioita myös työkalujen tai ongelmien kautta. Saatesanoiksi lisättiin, että vastauksen tarkkuudella ei ole väliä, jottei vastaaja jättäisi kyselyyn vastaamista kesken, vaan vastaisi edes jotain.

7. Kuvaile lyhyesti workflowtasi teemankehityksessä. *(avoin vastaus)*
Voit olla niin tarkka tai epätarkka kuin haluat.

Kahdeksas kysymys selvitti ongelmia teemankehityksen aikana. Kysymys oli avoin, jottei mitään rajattaisi pois. Saatesanat ohjasivat ajattelemaan myös ei-teknisiä ongelmia, sillä osaan niistäkin saattaa olla tekninen ratkaisu. Ei-teknisyys tuotiin esiin myös siksi, että vastaaja oli pohtinut koko kyselyn ajan asioita teknisistä lähtökodista, ja vastaaja haluttiin havahduttaa ajattelemaan ongelmia laajemmin.

8. Mainitse jokin ongelma, johon olet törmännyt teeman kehittämisen aikana. *(avoin vastaus)*
Vastaus voi liittyä tekniseen ongelmaan (eism. editori ei toimi, jokin työkalu puuttuu) tai ei-tekniseen (esim. kommunikointi asiakkaan kanssa on vaikeaa)

9. Tutustessasi uuteen työkaluun, minkälaista ohjeistusta kaipaat?
esim. videotutoriaalit, readme.md, koodiesimerkit

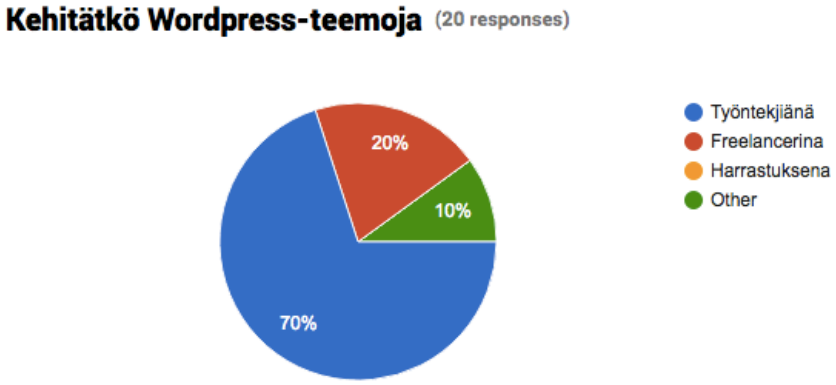
Viimeinen kysymys liittyi tutoriaaliin. Sen vastauksilla pyrittiin parantamaan oppaan sisältöä ja selvittämään, millä tavalla tieto voitaisiin tuoda esiin ja mitä tietoa kehittäjät kaipaisivat.

5.4 Kyselyn vastaukset ja pohdintaa

Kysely tavoitti Slackin kautta yli 300 Wordpress-aktiivia, mutta sai vastauksia ainoastaan 20:ltä. Vastauksia voi kuitenkin pitää suhteellisen luotettavina, sillä kuten aiemmin mainittu, kyseessä ei ole täysin avoin yhteisö vaan sinne pääseminen vaatii hieman työtä. Lisäksi kysymykset olivat avoimia, joten vastausten laatua voitiin arvioida lukemalla vastaukset, jotka vakuuttivat ainakin siitä, ettei joukkoon ollut eksynyt ketään tahallisesti väärin vastaamaan.

Huolenaiheita vastausten luotettavuudesta herättää ainoastaan vastausten pieni määrä. Tämän seurauksena osa teknologioista tai kehityksen ongelmakohdista on voinut jäädä kokonaan pois. Alla on käsitelty kysymyksiä ja niiden vastauksia yksityiskohtaisemmin.

Ensimmäiseen kysymykseen vastanneista 70 % kehitti teemoja työntekijänä ja 20 % freelancerina, kuten kuvassa 12 näkyy. Muu-kohdan valinneet kertoivat toimivansa yrittäjänä tai johtavansa kehitystä.

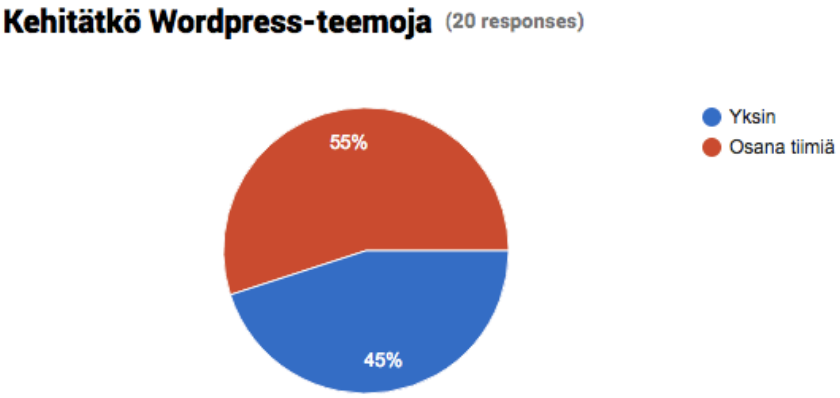


KUVA 12: Työsuhde oli hallitseva syy teemankehitykseen

Wordpress-ammattilaisten keskuuteen lähetetty kysely antoi toiseen kysymykseen odotettuja vastauksia. Kahdelta vastanneelta löytyi kokemusta yli viiden vuoden ajalta, ja suurimmalta osalta vähintään kolme vuoden ajalta. Kuva 13 avaa jaottelua tarkemmin.

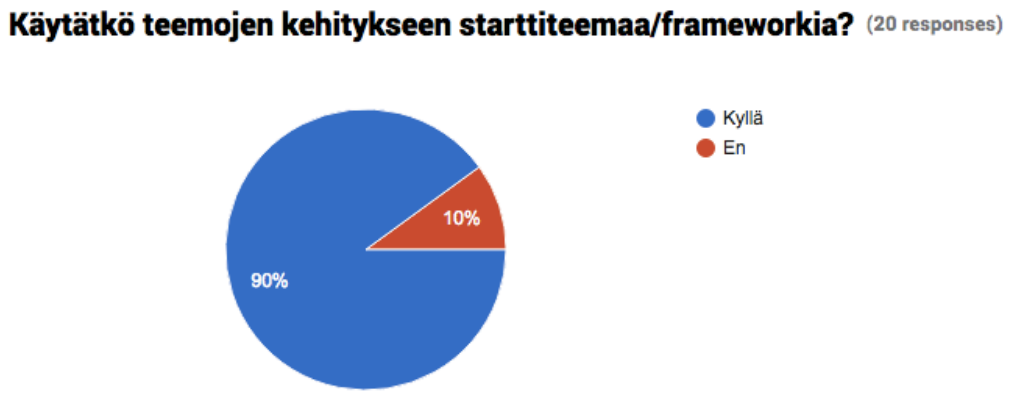
KUVA 13: Vastauksien perusteella kyselyyn vastasi kokeneita ammattilaisia

Kysymys kolme tuotti hieman yllättäviä vastauksia, kun tulosta verrataan ensimmäiseen kysymykseen, sillä vain 55 % vastaajista kehitti teemoja osana tiimiä, kun taas 70 % ensimmäiseen kysymykseen vastaajista kertoi kehittävänsä teemoja työntekijänä. Syitä vähäiseen tiimityöskentelyyn voi olla useita. Osa vastaajista ei laskenut vastauksessaan tiimin osaksi esimerkiksi graafikkoa tai projektipäällikköä vaan oletti kysymyksen tiimin koskevan nimenomaan teeman kehityksen ohjelmointityötä. Esimerkiksi eräs ”Yksin” -vaihtoehdon valinnut kertoi kuitenkin saavansa graafikolta layoutin. Kysymyksen muotoilu paremmin olisi saattanut muuttaa tuloksia. Kuvassa 14 on näkyvissä vastausten esitys graafisesti.



KUVA 14: Teemoja kehitetään hieman useammin osana tiimiä

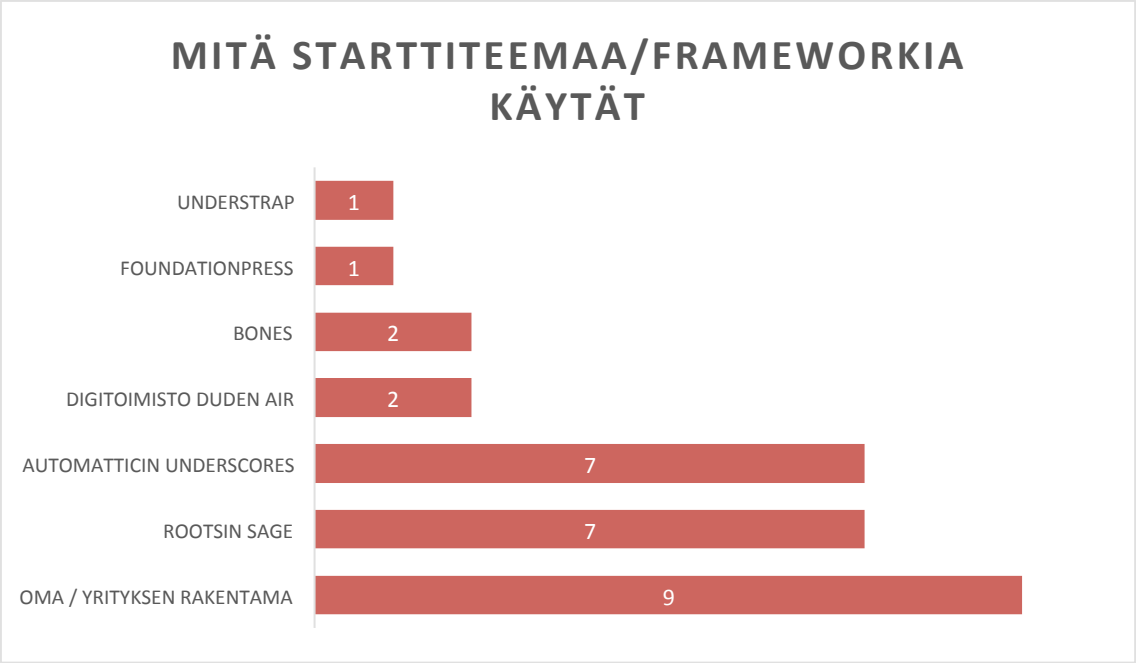
Kuvassa 15 näkyvä tulos neljänteen kysymykseen oli odotettavissa, kehittäjien pitkä kokemus huomioon ottaen.



KUVA 15: Starttiteemat ovat huomattavan laajasti käytössä vastanneiden keskuudessa.

Kuten Tadlock (2008) mainitsee blogiartikkelissaan *Why I created a WordPress theme framework*, tuovat starttiteemat vauhtia kehitykseen mahdollistaen koodin uudelleenkäytettävyyden. Kolme vastaajaa mainitsikin viidennessä kysymyksessä starttiteeman käytön syyksi juuri nopeamman alun projektille. Myönteisiä vastauksia oli odotettavissa, koska kokeneet kehittäjät ovat todennäköisimmin huomanneet starttiteemojen hyödyn käytännössä.

Kuvassa 16 on nähtävillä kysymyksen viisi vastausten jakautuminen. Mikäli kehittäjän oma teema pohjautui vahvasti johonkin kuvassa mainituista, on se sisällytetty sekä kyseisen teeman nimen kohdalle että *oma / yrityksen rakentama* -kohtaan.



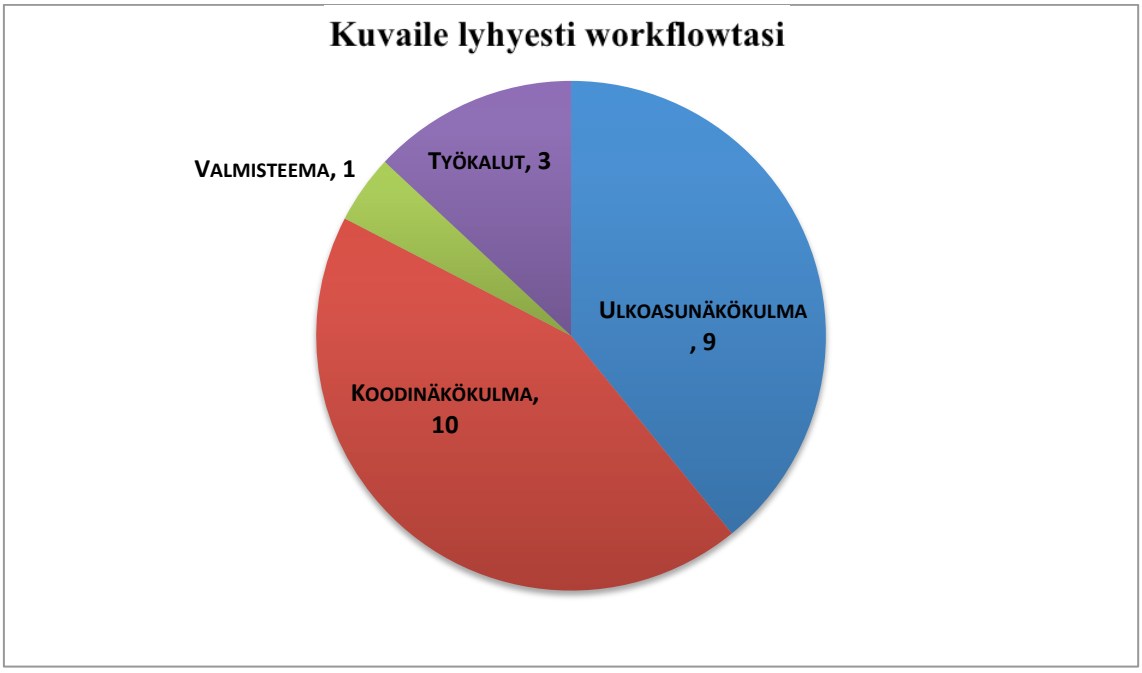
KUVA 16: Vastauksissa mainitut starttiteemat

Päätös jättää viides kysymys avoimeksi oli hyvä, sillä kysymyksen ensimmäiseen osaan vastattiin monisanaisesti. Kolme vastaajista kertoi, että heillä on käytössään oma teema, mutta se pohjautuu joko Sageen tai Underscoresiin.

Mainittakoon vielä, että listassa ei ole lainkaan sovelluskehyksiä, vaan kaikki yllä mainitut ovat starttiteemoja. Kysymykset olisi ehkä voitu jakaa kahteen osaan, jolloin olisi varmistuttu siitä, käyttääkö joku vastaajista sovelluskehyksiä. Toisaalta näyte on niin pieni, että voitaneen todeta sovelluskehysten olevan vain harvinaisemmin käytössä kuin starttiteemojen. Toisaalta osa apufunktiokirjastoista, joita vastaajat kertoivat käyttäneensä, voidaan laskea sovelluskehyksiksi.

Siinä missä viidennen kysymyksen avoimeksi jättäminen osoittautui hyväksi päätökseksi, olisi sen miksi-osuus pitänyt erottaa omaksi kysymyksekseen. Aikaisemmin mainittujen kolmen nopeampaan aloitukseen viittavan vastauksen lisäksi muut vastaajat eivät nimittäin ottaneet siihen kantaa. Kielteisesti neljänteen kysymykseen vastanneet perustelivat starttiteeman tai sovelluskehyksen poisjättämistä valmisteemojen käytöllä tai sillä, että ne tuovat mukaan liikaa ylimääräisiä ominaisuuksia.

Kuvassa 17 on nähtävillä vastaukset kysymykseen seitsemän, jotka on jaoteltu eri kategorioihin.



KUVA 17: Työjärjestysten kategorisointia

Kysymys koskien työjärjestystä (kysymys 7) sai koko kyselyn laajimmat vastaukset. Puolet vastaajista (10) ei maininnut työprosessin alkavan ulkoasun tai rautalankamallin suunnittelusta, vaan sen sijaan vastauksissa puhuttiin pohja- tai starttiteemasta, jonka päälle sivusto rakennetaan. Tämä ei kuitenkaan suoraan tarkoita, etteivät kyseiset vastaajat käyttäisi omaa tai jonkun muun suunnittelemaa ulkoasua kehityksen pohjana. Kysymyksessä ollut *workflowtasi teemankehityksessä* saattoi vain rajata nämä pois.

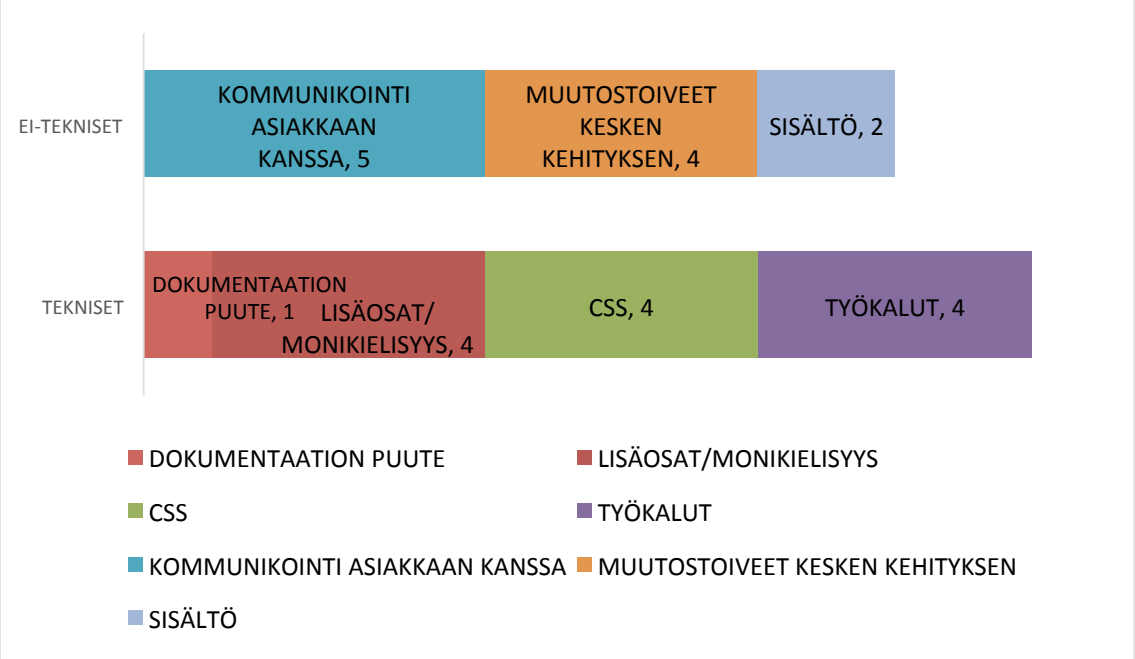
Toki on mahdollista, että ainakin osa kehittäjistä kehittää teeman niin, että asiakas esittää toivomukset, jonka jälkeen hän ottaa kantaa kehittäjän rakentamaan ulkoasuun vasta koodatessa. Kahdessa vastauksessa kuvattiin selkeästi juuri tällainen työjärjestys.

Yhdeksän vastaajaa mainitsi työjärjestyksen lähtevän liikkeelle joko rautalankamallin suunnittelusta tai suoraan ulkoasusta. Näistä kuusi kertoi yrityksen graafikon tai jonkun ulkopuolisen tahon suunnittelevan ulkoasun. Tällaisessa tilanteessa on luonnollista, että suunnittelu jää työjärjestyksestä pois, sillä se on todennäköisesti tehty ennen kuin graafikko suunnittelee ulkoasun, eikä kehittäjä itse ole ollut siinä mukana.

Kolme vastaajista ei kuvannut varsinaisesti omaa työnkulkuaan, vaan listasi muutamia työkaluja, joita he käyttävät kehityksen aikana. Yksi vastaajista kertoi, että asiakas valitsee valmisteemoista mieluisen, jonka pohjalta sivusto rakennetaan.

Avoimen kysymyksenasettelun tavoite oli nostaa esiin myös ongelmia teemankehityksessä. Kolme vastaajista mainitsikin viimeistelevänsä työtä asiakkaan kanssa paljon. Yhden vastauksen sanojen asettelusta paistoi läpi turhautuneisuus tilanteeseen. Tämä ei tietysti välttämättä ole aina ongelma, saatetaanhan kehittäjän ja asiakkaan välisessä sopimuksessa tarjota mahdollisuutta tällaiseen menettelyyn.

Lisää ongelmia nousi esiin kysymyksessä kahdeksan. Tämä oli myös ensimmäinen kysymys, johon oli jätetty vastaamatta: kaksi vastaajista oli jättänyt kohdan tyhjäksi. Osa kuitenkin nosti esiin myös useamman ongelman. Ongelmia nostettiin esiin yhteensä 24, jotka on jaoteltu alla olevaan kuvaan 18.



KUVA 18: Teemankehityksen ongelmat

Eniten ongelmia mainittiin olevan asiakkaan kanssa kommunikoinnissa. Käsitteistön tai kehittäjän omien näkemysten viestiminen asiakkaalle koettiin vastausten perusteella hankalaksi. Vaikeaa oli myös kommenttien tai vastausten saaminen asiakkaan suunnalta. Asiakkaat aiheuttivat ongelmia myös esittämällä muutostoiveita sivuston ulkoasuun tai sisältöön kesken teeman kehityksen. Myös sisällön toimituksen venyminen tai sen heikko laatu nähtiin ongelmallisiksi.

Teknisellä puolella ongelmat jakautuivat lisäosien, erityisesti monikielisyyslisäosien yhteensopivuusongelmien, ulkoasun CSS-ongelmien ja työkalujen toimimattomuuden kesken. Lisäksi ongelmaksi koettiin dokumentaation puute, kun aikaisemmin tehtyjä päätöksiä oli vaikea muistaa vanhan projektin ääreen palattaessa.

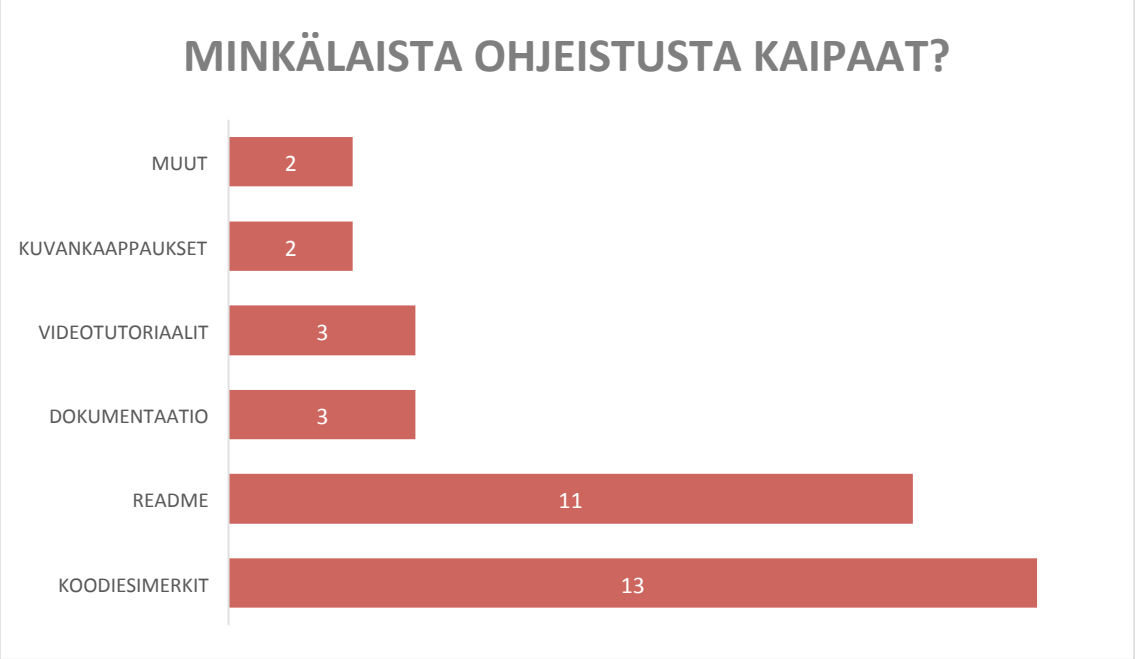
CSS-ongelmien joukossa esiin nostettiin responsiivisuus, yhteensopivuus eri selainten kanssa ja elementtien keskitys pystysuunnassa. Työkalu-kategorian alle laitettiin lokaalin ympäristön ongelmat, teemojen testaamisen tehottomuus sekä niiden ylläpidon vaikeus. Myös olemassaolevien komponenttien uudelleenkäyttö oli hankalaa ja tämän takia vähäistä.

Vastaukset kahdeksanteen kysymykseen olivat monipuolisia, ja avoin kysymys osoittautui hyväksi ratkaisuksi. Kysymykseksi jää, olisivatko tyhjäksi kohdan jättäneet vas-

tanneet, ettei ongelmia ole tai etteivät he osaa niitä nimetä, mikäli kysymys olisi ollut pakollinen.

Yhdeksännen kysymyksen vastaukset olivat selvästi keskittyneet kahteen vaihtoehtoon. 13 kertaa vastauksissa mainittiin tärkeäksi koodiesimerkit sekä 11 kertaa readme. Lisäksi kattava dokumentaatio mainittiin muutamaan kertaan. On otettava huomioon, että yleensä readme sisältää toivottuja koodiesimerkkejä, ja yksinkertaisille työkaluille se voi käydä kattavasta dokumentaatiosta. Vastausten rajat ovat siis häilyviä. Tekstimuotoisista tavoista tutustua työkaluihin mainittiin vielä wikit.

Videotutoriaalit olivat kolmen vastaajan mielestä erittäin tärkeitä, mutta toisaalta kahden muun vastaajan mielestä epämieluisa tapa tutustua uuteen työkaluun. Kuvankaappaukset työkalujen käytöstä sekä toisten työkalua käyttävien ihmisten apu mainittiin myös mieluisiksi vaihtoehdoiksi. Kuvassa 19 esitellään vastaukset graafisesti.



KUVA 19: Kehittäjien kaipaama ohjeistus jaoteltuna

5.5 Kyselyn tavoitteiden arviointi

Kyselyn tulokset vastasivat sen tavoitteisiin tyydyttävästi. Starttiteemoja listattiin laajalti, mutta toisaalta kävi ilmi, että sovelluskehyksiä ei käytetä ainakaan tämän näytteen

joukossa. Miksi-kysymystä olisi pitänyt painottaa enemmän, jotta käytännönläheisiä syitä olisi saatu nostettua esiin.

Toisena tavoitteena oli saada kartoitettua joitain ongelmia, jotka liittyisivät nimenomaan kehitysympäristöön. Niitä nousikin esiin muutamia. Esimerkiksi Dustpress voi helpottaa komponenttien uudelleenkäyttöä, kun ulkoasukoodi on erotettu mallista. Myös teemojen ylläpito helpottuu samasta syystä.

Kyselyn viimeisenä tavoitteena oli selvittää, millä tavalla kehittäjät haluavat vastaanottaa tietoa. Kyselyn vastauksiin perustuen opas toteutetaan verkkoon tekstimuodossa, ja se sisältää runsaasti koodiesimerkkejä.

6 TUTORIAALIN SUUNNITTELU

Kuten kyselykin osoittaa, on kattava dokumentaatio koodiesimerkein ohjelmoijille mieluinen tapa tutustua uusiin työkaluihin. Dustpressille on jo olemassa readme-tiedosto sen Github-repositoriossa (Dustpress Readme 2016), joten sen uudelleen luominen ei ole tarpeen. Sen sijaan tarkoituksena on luoda tutoriaali, jossa on Dustpressin toimintojen esittelyn lisäksi vaihe vaiheelta etenevä selonteko teeman kehittämisestä. Tutoriaali avaa myös MVVM-ajattelua sekä sisältää linkkikirjaston jo olemassa oleviin materiaaleihin, kuten DustPHP:n. dokumentaatioon. Tässä luvussa tutustutaan hyviin verkossa julkaistavan oppaan ominaisuuksiin.

6.1 Aiheen rajaus

Seeleyn (2013) mukaan tutoriaalin aiheen rajaus on tärkeää sekä lukijan että kirjoittajan kannalta. Asioiden välillä hyppiminen hankaloittaa lukijan oppimisprosessia, ja syvempi katsaus yhteen aiheeseen on lukuisia pintaraapaisuja hyödyllisempää. Tekijän kannalta rajaus mahdollistaa voimavarojen ja huomion keskittämisen tietyn tavoitteen saavuttamiseksi parantaen tutoriaalin laatua.

Aiheen rajaus teoriatasolla ei usein ole tarpeeksi, sillä tutoriaalin tulisi sisältää myös käytännön esimerkkejä. Ohjelmointitutoriaalin sisältämä teoria on hyödyllistä, mutta jollei lukija opi soveltamaan teoriaa käytännön tasolla, tutoriaali ei ole ollut hyödyllinen. Tutoriaalin aiheen rajausta voi lähestyä tavoitteen näkökulmasta ja miettiä, mitä haluaa lukijan oppivan käytännössä. Tämä kysymys auttaa käytännön esimerkkienkin rajauksessa. Sisältöä voi myös arvottaa kysymällä, edistääkö sisältö oppijan tavoitetta vai ei. (Lee 2015.)

6.2 Rakenne

Tutoriaalin tavoitteen määrittely helpottaa sen sisällön jäsentelyä. Tutoriaalin ollessa vähänkään pidempi, kannattaa sille luoda sisällysluettelo, jotta lukija on selvillä etenemisestään. Lisäksi sisällysluettelo helpottaa tutoriaalissa liikkumista, jos lukija haluaa

palata takaisin kertaamaan aiempaa asiaa tai vaikkapa etsii tiettyä kohtaa. Seeley (2013) ehdottaa verkkotutoriaaleille kolmiosaista rakennetta.

Johdannossa perehdytetään lukija tutoriaalin aiheeseen ja kerrotaan, mitä tutoriaali opettaa ja kuinka pitkä se on. Lisäksi olisi hyvä kertoa, mitä ennakkotietoja tai välineitä tutoriaalin suorittaminen vaatii. Johdanto on myös kirjoittajan tilaisuus houkutella lukija jatkamaan tutoriaalin parissa. (Seeley 2013.)

Johdannon jälkeen tulee varsinainen sisältöosuus (Seeley 2013.) Mikäli sisältöä on runsaasti, tulisi se jakaa pienempiin osiin. Jokaisen osan tulisi olla mahdollisimman tiivis, mutta kuitenkin sisältää selkeä kokonaisuus. Liikkeelle lähdetään perusasioista, ja uusi asia rakentuu jo opitun päälle. (Lee 2015.)

Yhteenveto sijoittuu tutoriaalin loppuun, ja siellä summataan tutoriaalissa opetettua asiaa. Alussa kerrottu tavoite voidaan tuoda esiin uudelleen, jolloin lukija voi arvioida, onko hän saavuttanut tutoriaalin asettamat tavoitteet. Lisäksi voidaan tuoda esille jatkolukemistoa samasta aiheesta. (Seeley 2013.)

6.3 Sisältö

Tutoriaalin kielen tulee olla ihmisläheistä ja lukijaa puhuttelevaa. On tärkeää, ettei lukijaa karkoteta ylimieliseltä kuulostavalla tekstillä tai hukuteta informaatiota hienojen termien alle. Samalla tutoriaalin kielen tulee kuitenkin kuulostaa ammattitaitoiselta, jotta lukijan luottamus säilyy. Asia esitellään rauhassa palanen kerrallaan, eikä anneta lukijalle valtavaa määrää informaatiota kerralla. (Lee 2015)

Seeley (2013) huomauttaa, että tutoriaalissa *miten jokin asia tehdään* -ohjeistuksen lisäksi mukana tulee olla jatkuvasti perustelut sille, miksi asiat tehdään juuri näin. Tämä helpottaa teorian hyödyntämistä myös tutoriaalin kontekstin ulkopuolella, mikä onkin yksi tutoriaalin onnistumisen mitta. Tutoriaalin lukijan tulisi kyetä soveltamaan oppimaansa myös omissa projekteissaan.

Kuvilla on tärkeä osuus tutoriaaleissa. Hyvä kuvitus voi auttaa ymmärtämään luettua tekstiä. (Seeley 2013.) Esimerkiksi projektin kansiorakenteen esittäminen kuvalla onnis-

tuu luotettavammin ja helpommin kun tekstin avulla. Toimivan ohjelmistokoodin esittely lukijalle kooditutoriaalissa on Seeleyn mukaan välttämättömyys.

7 POHDINTA

Kuten aiemmin on todettu, ei Wordpress-maailma ole ongelmista vapaa. Kyselyssä mainittiin monia teknisiä ja ei-teknisiä ongelmia. Lisäksi, kuten todettiin luvussa 2.2.5, teemoja hankittaessa voi tulla ostaneeksi huonon valmisteeman, jonka jatkokehityksestä saattaa joutua pulittamaan ison summan.

Myöskään teemankehitys ei ole täysin ongelmatonta. Teemojen ulkoasutiedostot ovat monoliittisia palasia koodia. On totta, ettei niiden koko välttämättä kasva valtavan suureksi verrattuna vaikkapa sovelluksiin, jonka ansiosta osalta monoliittisen ohjelmointitavan ongelmista vältytään.

Mielestäni yksi isoimpia ongelmia monoliittisessa kehityksessä on koodin uudelleenkäytettävyys ja modulaaristen palasten luomisen vaikeus. Tähän ongelmaan Dustpress tuo ratkaisun MVVM-arkkitehtuurinsa ansiosta. Tämän lisäksi Dustjs tuo teeman näkymien kehitykseen paljon hyviä työkaluja. Voidaankin esittää, että Dustpressin käyttämisestä teeman kehityksessä on monia hyötyjä tavalliseen kehitystapaan verrattuna.

Työn tavoitteena oli madaltaa kynnystä tutustua Dustpressiin. Uskon, että jos työn lukee Wordpress-kehittäjänä voi siitä herätä kiinnostus sovelluskehykseen tutkimiseen. Työn tarkoituksena oli tuottaa tutoriaali. Tähän ei ajanpuutteen vuoksi täysin päästy. Tutoriaali saatiin kuitenkin hyvin aloitettua ja työ sisältää paljon teoriatietoa, jota voi hyödyntää tulevaisuudessa tutoriaalia jatkokehitettäessä.

Jatkossa aion kehittää tutoriaalia enemmän, jotta se voisi syventyä myös haastavampiin aiheisiin. Lisäksi tutoriaali julkaistaan Githubissa, jonka tavoitteena on saada muutkin kommentoimaan ja muokkaamaan tutoriaalia. Kerään myös palautetta Dustpressin kehittäjiltä sekä muilta Wordpress-kehittäjiltä, jotta voin tehdä tutoriaalista yhä paremman. Yksi mahdollisuus tulevaisuudessa on myös luoda starttiteema, joka hyödyntää Dustpressiä.

Ajantasaisen Dustpress-tutoriaalin löydät osoitteesta:
https://github.com/Pikkulahti/dustpress-tutorial

LÄHTEET

Arponen, M. 2016. Haastattelu 4.10.2016. Haastattelija Lahtinen, A. Tampere.

Composer. 2016. Composer Book: Introduction. Composer-kehitystiimi. Luettu 29.9.2016.
https://getcomposer.org/doc/00-intro.md

Damstra, D., Stern, H., Williams, B. 2013. Professional Wordpress: Design and Development. 2013. Wrox Publishing.

Dustpress Readme. 2016. Github: Dustpress Readme. Luettu 27.9.2016.
http://github.com/devgeniem/dustpress/blob/master/README.md

DustJS-manuaali. 2016. Linkedin: DustJS-manuaali. Luettu 30.9.2016.
www.dustjs.com/guides

Geniem 2016. Geniem – Ketterä kumppanisi menestyvien verkkopalveluiden tuotantoon. Luettu 29.11.2016.
http://geniem.fi/

Gossman, J. 2006. Tales from the Smart Client: Advantages and disadvantages of M-V-VM. Luettu 29.9.2016.
https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/

Kekäläinen, O. 2016. XDebug – koodarin paras ystävä PHP-nopeusoptimoinnissa. WP-palvelu. Luettu 11.10.2016.
https://wp-palvelu.fi/blogi/xdebug-nopeusoptimointi/

Laplante, P. 2007. What Every Engineer Should Know about Software Engineering. CRC Press

Lee, J. 2015. Makeuseof. What Makes a Good Programming Tutorial? Luettu 13.10.2016.
http://www.makeuseof.com/tag/makes-good-programming-tutorial

McCollin, R. 2014. Choosing a WordPress Theme Framework – the Ultimate Guide. Luettu 29.9.2016.

https://premium.wpmudev.org/blog/choosing-a-wordpress-theme-framework-the-ultimate-guide

Mullenweg, M. 2016. Matt Mullenweg: Interview and Q&A – WordCamp Europe 2016. Viitattu 28.9.2016.
https://videopress.com/v/MVF9BxaG.

Retz, C. 2013. Dust PHP: A lightweight, powerful templating engine in PHP. Luettu 30.9.2016.
http://cretz.github.io/dust-php/

Seravo 2016a. Github: Seravo WordPress. Luettu 11.10.2016.
https://github.com/Seravo/wordpress

Seravo 2016b. Github: Seravo WP-palvelu Vagrant Box. Luettu 11.10.2016.
https://github.com/Seravo/wp-palvelu-vagrant

Siltala, V. 2016. Haastattelu 18.10.2016. Haastattelija Lahtinen, A. Tampere.

Stewart, I. 2012. Themesharper.com: A 1000-Hour Head Start: Introducing The _s Theme. Luettu 28.9.2016.
https://themeshaper.com/2012/02/13/introducing-the-underscores-theme/

Suoranta, T. 2016. How to ensure long lifespan of a website? Tampere Wordpress Meetup. Ravintola Teerenpeli, Tampere. 8.9.2016

Roots. 2016. Sage Documentation: Theme Wrapper. Roots. Luettu 29.9.2016.
https://roots.io/sage/docs/theme-wrapper/

Studiopress. 2016. Studiopress.com: Genesis Framework by StudioPress. Luettu 28.9.2016.
http://my.studiopress.com/themes/genesis/

Seeley, J. 2013. How to Create Good Tutorials on Any Subject. Luettu 13.10.2016.
http://justinseeley.com/tutorials/create-great-tutorials-subject/

Tadlock, J. 2010. Justintadlock.com: Frameworks? Parent, child, and grandchild themes? Luettu 18.10.2016.
http://justintadlock.com/archives/2010/08/16/frameworks-parent-child-and-grandchild-themes

Tampereen Yliopisto. 2010. Kvantitatiivisten menetelmien tietovaranto: Kyselylomakkeen laatiminen. Luettu 9.11.2016.
http://www.fsd.uta.fi/menetelmaopetus/kyselylomake/laatiminen.html

Theme Hybrid. 2016. Themehybrid.com: Hybrid Core WordPress theme framework. Luettu 4.10.2016.
http://themehybrid.com/hybrid-core

Themeforest. 2016. Themeforest: Wordpress Themes From Themeforest. Luettu 28.9.2016.
https://themeforest.net/category/Wordpress

Themefuse. 2016. Themefuse: Premium WordPress Themes and WordPress Templates. Automotive & Auto Dealer WordPress Theme - Auto Trader. Luettu 5.10.2016.
http://themefuse.com/wp-themes-shop/auto-wordpress-theme/

Theme Handbook 2016. Wordpress Developer. Theme Handbook: Template Hierarchy. Luettu 5.10.2016.
https://developer.wordpress.org/themes/basics/template-hierarchy/

Thomas. 2006. "DRY says that every piece of system knowledge should have one authoritative, unambiguous representation." - Dave Thomas, Luettu 29.9.2016.

http://www.artima.com/intv/dry.html

Tolvanen, P.. 2015a. Parin tonnin saittiprojekti Wordpressillä – kuka tekisi? Vieritys-palkki: Blogi verkkopalveluiden uudistajille ja kehittäjille. Luettu 5.10.2016. http://vierityspalkki.fi/2015/09/11/ala-osta-parin-tonnin-wordpress-saittia/

Tolvanen, P. 2015b. Parin tonnin saittiprojekti Wordpressillä – kuka tekisi? Vieritys-palkki: Blogi verkkopalveluiden uudistajille ja kehittäjille. Luettu 29.9.2016. http://vierityspalkki.fi/2015/09/07/parin-tonnin-saittiprojekti-Wordpressilla-kuka-tekisi/

Vice, R. & Siddiqi, M. 2012. MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF. Packt Publishing.

Williamson, P. 2016. WordPress Page builder plugins: a critical review. Luettu 29.9.2016. https://pippinsplugins.com/wordpress-page-builder-plugins-critical-review/

Wikimedia Commons 2016. Kuvan lisenssi: Creative Commons Nimeä-JaaSamoin 3.0 Ei sovitettu. Kuvan tekijä: Ugaya40. Haettu 1.11.2016. https://commons.wikimedia.org/wiki/File:MVVMPattern.png

Wordpress. 2016. Wordpress.org: Theme Directory. Wordpress-kehitystiimi. Luettu 27.9.2016. https://Wordpress.org/themes.

Wordpress Codex, 2016a. Anatomy of a Theme. Luettu 26.9.2016. https://codex.Wordpress.org/Theme_Development#Anatomy_of_a_Theme

Wordpress Codex. 2016b. Include Tags. Luettu 30.9.2016. https://codex.Wordpress.org/Include_Tags

Wordpress Suomi. 2016. Wordpress Suomi: Chat. Luettu 6.10.2016. https://fi.wordpress.org/chat/

Wordpress.org. 2016. About: Features. Luettu 5.10.2016. https://wordpress.org/about/features/

WPBeginner. 2016. Balkhi, S & WPBeginner toimitus. Selecting the Perfect WordPress Theme – 9 Things You Should Consider. Luettu 4.10.2016. http://www.wpbeginner.com/wp-themes/selecting-the-perfect-theme-for-wordpress

W3Techs. 2016. W3Techs: Web Technology Surveys. Usage of content management systems for websites. Luettu 1.11.2016. https://w3techs.com/technologies/overview/content_management/all

LIITTEET

Liite 1. Kysely Wordpress-teemankehityksestä

Kysely Wordpress-teemankehityksestä

Hei sinä Wordpress-teemaaja!
Olen Tampereen Ammattikorkeakoulun opiskelija ja teen opinnäytetyötä WordPress-teemankehityksestä DustPress-frameworkin näkökulmasta.

Alla olevalla lyhyellä kyselyllä pyrin kartoittamaan teemankehittämisen vaiheita, mahdollisia ongelmia sekä käytössä olevia työkaluja.

Olisin kiitollinen, jos voisit jakaa kokemuksiasi WordPress-teemankehityksestä. Osallistumisestasi on suuri hyöty työlleni, kiitos siis jo etukäteen!

Yksittäisiä vastauksia ei tulla nostamaan suoraan esiin missään vaiheessa työn aikana.

Mikäli sinulla on kysyttävää minuun saa yhteyttä:
Sähköpostilla: anttoni.lahtinen@cs.tamk.fi
Twitterissä & Wordpress Finlandin Slackissa: pikkulahti

Kehitätkö Wordpress-teemoja *

Valitse sopivin vaihtoehto

- ○ Työntekjiänä
- ○ Freelancerina
- ○ Harrastuksena
- ○ Other...

Kehitätkö Wordpress-teemoja *

Valitse yleisempi vaihtoehto

- ○ Yksin
- ○ Osana tiimiä

Kuinka pitkään olet kehittänyt teemoja? *

1. Alle vuoden
2. 1-2 vuotta
3. 3-5 vuotta
4. 5-10 vuotta
5. 10 vuotta tai enemmän

Käytätkö teemojen kehitykseen starttiteemaa/frameworkia? *

1. Kyllä
2. En

Jos vastasit kyllä: Mitä starttiteemaa/frameworkia käytät ja miksi?

Long-answer text

Jos vastasit en: Miksi et?

Long-answer text

Kuvaile lyhyesti workflowtasi teemankehityksessä.

Voit olla niin tarkka tai epätarkka kuin haluat.

Long-answer text

Mainitse jokin ongelma, johon olet törmännyt teeman kehittämisen aikana.

Vastaus voi liittyä tekniseen ongelmaan (eism. editori ei toimi, jokin työkalu puuttuu) tai ei-tekniseen (esim. kommunikointi asiakkaan kanssa on vaikeaa)

Long-answer text

Tutustessasi uuteen työkaluun, minkälaista ohjeistusta kaipaat?

esim. videotutoriaalit, readme.md, koodiesimerkit

Long-answer text

Liite 2. Dustpress-tutoriaali

How to DustPress

Tutorial for theme development with Dustpress, a MVVM architecture based WordPress theme framework.

Intro

This is a tutorial for creating a Wordpress the with Dustpress. It covers theme MVVM model, Dustpress installation, basic sites, looping content and advanced custom fields. More stuff will be added later. If you are a busy person who got no time for blabble and is just looking for a quick start please (**TL;DR**) checkout the Dustpress repository for the readme.

What you'll need

- Local developing enviroment that runs WordPress
  - I'm using Geniem's gdev with wp-project but your own will probably do just fine.
- Composer
  - You can install Dustpress manually but that is lame. Composer is cool.
- WP-CLI is not needed but handy.
  - Seriously use it if you aren't already

DustPress offers

- Dust.js templating engine
- Separate data models
- Helpers functions

Table of contents

4. Header & Footer
5. Retrieving posts
   Coming in the future:
6. Advanced Custom Fields
7. Pagination

0. MVVM explained

MVVM stands for Model View ViewModel.

Model collects the business level data. In Dustpress here you have the logic for fetching data. You can use functions like `get_posts()` to collect and return the data needed. Notice that you can fetch the data from anywhere, even outside WordPress!
View model takes care for binding the right data to the right view. There are a view model for each view. In DustPress, the framework main class takes care of the data binding so you only have to create models and views.

View is the applications user interface and the view model controls the data it displays. In Dustpress we use Dust.js templating engine for rendering the views.

1. DustPress explained

Here we discuss little bit about DustPress and I explain the basic workings of it. If you are TL;DR kind of person and just want to code you can skip to the Installation part and start doing.

Project structure

DustPress based theme folder differs on normal one. In the theme directory we have folders called `models` and `partials`

models/

In the `models` folder we write our PHP code where we retrieve and modify the data. The Wordpress template hierachy exists. We can also create custom templates and name them to almost anything, just remember to declare the

template name at the start of the file. We get to know model files more in depth later.

partials/

The `partials` folder contains the views of our MVVM architecture. They are written in Dust file format which is basically HTML mixed with curly braces. `{foo}` represents a reference to a data object named *foo*. When the .dust files are rendered to HTML, the object's data is printed in place of the reference. Dust offers helpers for example escaping the output of some reference and to implement some logic in the views. Though it is possible to create complex logical structures with dust helpers developers should use these in high consideration. Most of the logic should be written outside of the view.

Binding the data

Models are named just like normal WordPress templates would be. In the Model we create a that has the name of the template. This class extends the DustPress' Model class.

For example template `home.php` would have class Home.

```
class Home extends \DustPress\Model {   ... }
```

Template names need to match the class names. If template name contains hyphens like `single-post.php` or `custom-page.php`, the class name will be written in camelcase: SinglePost, CustomPage.
Upon execution DustPress creates global data object named with the models's name. It then gathers data from the public methods that are located inside that model's class. All returned data will be assigned in to the data object and named after the method.

Inside Home class we have methods `Title` and `User` .

```
class Home extends \DustPress\Model {   public function Title() {     return
"Cool blog";   }   public function User() {     return "username";   } }
```

The data tree of the global object would look like:

```
object(stdClass) {   ["Home"]=>   {     ["Title"]=> "Cool blog"     ["Us-
er"]=> "username"   } }
```

Code inside protected and private methods are executed, but would the method return data it will not be bound to the data object. This offers a way to create methods inside models for other use than retrieving data.

DustPress also offers function `bind` for binding the data to the global object.

```
$this->bind( $data, "DataName" );
```

Above function would bind contents of $data under "DataName" so the tree would look like:

```
object(stdClass) {   ["ClassName"]=>   {     ["DataName"]=> $data   } }
```

Printing the data

In the `partials` folder we create Dust-file named after it's model. For our `home.php` we would have `home.dust`. The reason this folders is called partials, not views is that it contains all the pieces the view is constructed from. We have Dust-file for each view but cause we also don't want to repeat ourselves (stay DRY!) we have partials for header, footer and other parts of our site that are repeated often. One beaty of the DustPress is the easy way to write modular code.
Aas mentioned before DustPress uses Geniem's fork of DustPHP for rendering the Dust-files. DustPHP is based on LinkedIn's DustJS templating engine.

In the partial file we write the HTML-structure and between the code we can point to the content references. This is a brief cheatsheet for the stuff that we can do with Dust. We will cover some of these when they are used later in this tutorial. For a more in depth view check out DustJS Docs.

```
{reference}
```

Reference is used to insert the data it is pointing to in to the partial. You can also filter the references with `|filter key`. For example `{reference|s}` turns the automatic HTML-escaping off.

```
{! comment !}
```

Very handy way to comment your code. Content between comment tags will not be included to the HTML-output. No more `<!-- This is a comment -->` in the source code!

```
{#section}  {reference1} {/section}
```

Section tags are used to set the context where the reference data for the reference is searched.

```
{>partial/}
```

Partial tag is used to insert Dust file inside other Dust file. We talked about staying DRY and this is a way of doing it. You'll learn to use Partial tag in the Header & Footer part of this tutorial.

```
{@helper/}
```

Dust helpers extend the functions for logic and more. Check out DustPHP Docs for avalaible ones. DustPress has also some helperes of it's own which you can checkout here. We will cover many of the most used ones later in this tutorial.

Submodels

- Dust.js templating engine
- Separate data models
- Helpers functions for files

2. Installation

If you don't have Composer installed install it.

Then navigate the root of your wordpress project folder and run

```
$ composer require devgeniem/dustpress
```

This adds `"devgeniem/dustpress": "*"` to your *composer.json*. If you don't have composer.json already Composer creates it with following contents:

```
{  "require": {   "devgeniem/dustpress": "*"  } }
```

You of course add these lines to *composer.json* manually and run `$ composer install` if you want to.

Dustpress uses Composer's autoload feature so we don't need to require Dustpress anywhere. We just enable it by calling it in our *functions.php*

```
dustpress();
```

Now let's install the DustPress Debugger.

Manual installation

If you don't want to use Composer you can install Dustpress manually by cloning it in to your project folder.

```
$ git clone https://github.com/devgeniem/dustpress.git
```

After that we have to require the dustpress.php file in your functions.php.

```
require('some/folder/dustpress.php');
```

After the require, we enable DustPress by calling it in the same file.

```
dustpress();
```

DustPress Debugger and DustPress.js Installation

Dustpress debugger prints out a button on the bottom of your DustPress powered page. Clicking the button opens the debugger overlay view with a data tree of the DustPress data object. This is a handy way to examine the data that DustPress is sending to your current view.

First we install DustPress.js to automatically add the data into the debugger view via AJAX. DustPress.js is a plugin for using the DustPress Model methods on the front end but is not covered more thoroughly in this tutorial, we are just using it with the debugger.

Let's install DustPress.js via composer:

```
$ composer require devgeniem/dustpress-js
```

Then we install the Dustpress Debugger.

```
$ composer require devgeniem/dustpress-debugger
```

After the installation we activate the plugins via WP-CLI or via the admin menu.

```
$ wp plugin activate dustpress-js $ wp plugin activate dustpress-debugger
```

Just one more thing! We visit our WordPress profile page in the admin menu and on the bottom of the page activate the DustPress Debugger.

3. Hello World

Now that we got DustPress running let's create a theme before hello worlding. In your theme folder you should now have `functions.php` where you call DustPress with dustpress();-function call. Create file `style.css` and add something like:

```
/* Theme Name: Dustpress theme Author: Pikkulahti Description: Dustpress powered theme License: GNU General Public License v2 or later
```

Create `index.php` and leave it empty.

Create folder `partials/` and `models/`.

Now we have a DustPress project structure!

Hello World

Let's create our first model and do the classic Hello World.

Let's create front-page and name it `home.php`. This way WordPress will show it automatically in the front-page cause of template hierarcy. The file will look like:

```
<?php class Home extends \DustPress\Model {   public function Hello() { return "Hello";   }   public function World() {     return "World!";   } }
```

We create class Index that extends Dustpress Model class. Remember that the classes name has to be same that of it's file.

In the class we create two public methods. Both return piece string from our "Hello World!". Now let's look at the `partials` folder.

In the folder let's we create file `home.dust`.

DustPress returns the data from the `home.php` to our view. The data tree looks like:

```
object(stdClass) {   ["Home"]=>   {     ["Hello"]=> "Hello"     ["World"]=> "World"   } }
```

In our `home.dust` file we write:

```
{#Home} <h1>{Hello} {World} !</h1> {/Home}
```

Here we search Hello and World object from the Home context. That matches the data tree and the view outputs the right data.

No we should see:

```
<h1> Hello world </h1>
```

In the front page of our Wordpress.

You can examine the data tree closer with the Debugger if you are logged in the profile that has the debugger activated.

4. Header & Footer

Let's create our first submodel. Create `shared/` folder to the `partials/` and `models/`.

In the `models/shared/` let's create `header.php`:

```
<?php class Header extends \DustPress\Model {   public function BlogName() { return "Dustpress blog";   } }
```

In the `partials/shared/` let's create `header.dust`:

```
{#Header} <h1> This is {BlogName} </h1> <hr> {/Header}
```

Now in the `models/shared/index.php` we bind the header to our model by adding

```
public function init() {   $this->bind_sub( "Header" ); }
```

into the Home class.

Finally we can use the subpartial in our `partials/shared/home.dust`:

```
{>"shared/header" /} {#Home} <h1>{Hello} {World} !</h1> {/Home}
```

Now the output should be:

```
<h1> This is Dustpress blog </h1> <hr> <h1> Hello world </h1>
```

Examining the data with the debugger again is recommended.

Now try to creating the footer same way! Note that you can use the same `init` method to bind the footer!

5. Retrieving posts

Lets see how Dustpress handles the post retrieving. First we can generate ten posts with WP-CLI:

```
$ wp post generate --count=10
```

Or you can create some by clicking around furiously in wp-admin, your call.

Now that we have the posts let's create a blog template `page-posts.php`. The code looks like:

```
<?php /* Template: Posts page */ class PagePosts extends \DustPress\Model { public function GetPost() {     $args = [ 'posts_per_page' => 10 ];     return get_posts( $args );   } }
```

We utilize the Dustpress' get_posts function that accepts the same arguments that Wordpress' get_post()-function. From the Dustpress Readme:

```
Post objects are queried with the WordPress get_post function and the data is
extended with metadata.
```

Our function call retrieves ten most recent posts. Let's print them in the view by creating `partials/shared/page-posts.dust`:

```
{#PagePosts}   <ul>   {#GetPost.posts}     <li>       <b>{post_title}</b>
</li>   {/GetPost.posts}   </ul> {#PagePosts}
```

GetPosts-object has posts array containing the post objects. Above code will loop through the post objects. Debugger clarifies things again. Create page and set the *Posts Page* template. Go to that page and you should see list of containing your posts. Debugger shows all the data that `get_posts()` function retrieves. Try adding some to that list!