Utveckling av Wordpress Tema för kodlös webbutveckling

Oliver Granlund

Examensarbete
Informations- och medieteknik
2016

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	5841
Författare:	Oliver Granlund
Arbetets namn:	Utveckling av Wordpress tema för kodlös webbutveckling
Handledare (Arcada):	Johnny Biström
Uppdragsgivare:	Ab Levelup Oy

Sammandrag:

Examensarbetet behandlar utvecklingen av ett Wordpress Tema för Ab Levelup Oy. Företaget behöver ett verktyg för att snabbare bygga nätsidor baserade på Wordpress, med hjälp av färdiga blockelement och mycket inställningar. Målet var att kunna göra färdiga nätsidor utan att behöva modifiera kod och att snabbt få något att visa kunden. Projektet baserar sig på Wordpress egna API och använder sig också av ett par tilläggsprogram som bas för projektet. Resultatet testades med en kund och i arbetet framgår hur resultatet kan användas i framtida projekt. En av de viktigaste sakerna med resultatet är dess användar- och utvecklarvänlighet i framtida bruk.

Nyckelord:	Wordpress, Tema, Bootstrap, Responsiv, Levelup
Sidantal:	31
Språk:	Svenska
Datum för godkännande:	7.12.2016

DEGREE THESIS	
Arcada	
Degree Programme:	Information and media technology
Identification number:	5841
Author:	Oliver Granlund
Title:	Development of a Wordpress Theme for codeless web development
Supervisor (Arcada):	Johnny Biström
Commissioned by:	Ab Levelup Oy

Abstract:

The thesis deals with the development of a Wordpress Theme for Ab Levelup Oy to accelerate development of sites for customers. Every new project at Levelup often required custom built solutions, which was time consuming. This theme would serve as a base for most new projects, it is based on repeating elements in the desired order to make websites as dynamic as possible. The goal for the work is to have the possibility to make whole websites for customers without committing any code. The development process includes planning of repeating elements and building the structure for them. This resulted in a theme which accelerates the development of projects with less code to write, but still needs collaboration with the designers to be able to make it codeless.

Keywords:	Wordpress, Theme, Bootstrap, Responsive, Levelup
Number of pages:	31
Language:	Swedish
Date of acceptance:	7.12.2016

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Informations- och medieteknik
Tunnistenumero:	5841
Tekijä:	Oliver Granlund
Työn nimi:	Wordpress pohjan kehitys koodittomalle verkkosivukehitykselle
Työn ohjaaja (Arcada):	Johnny Biström
Toimeksiantaja:	Ab Levelup Oy

Tiivistelmä:

Opinnäyte käsittelee Wordpress teeman kehitystä Ab Levelup Oy:lle. Teeman tarve liittyy toivomukseen voidakseen rakentaa verkkosivustoja nopeammin Wordpress-pohjalla käyttäen myös valmiita lisäosia ja ilman tarvetta koodaukselle. Tavoitteena on saada nopeasti valmistettua sivuja, tai ainakin esimerkkejä verkkosivujen ulkonäöstä asiakkaalle. Opinnäyte käsittelee koko kehitysprosessin sekä syyt harkituille valinnoille. Tärkeimmistä ominaisuuksista lopullisessa tuotteessa on sen käyttäjä- ja kehittäjäystävällisyys.

Avainsanat:	Wordpress, Teema, Bootstrap, Responsiivinen, Levelup
Sivumäärä:	31
Kieli:	Ruotsi
Hyväksymispäivämäärä:	7.12.2016

INNEHÅLL

Figurer

1 INLEDNING

Utveckling av en webbplats för en kund hos Levelup tar ca en månad. Utvecklarna var trötta på detta för att de tvingades göra ungefär samma arbete flera gånger på lite olika sätt beroende på projekt. Det fanns inte mycket globala värden utan element kunde variera mycket på sidbasis. De finns dock ett par koncept med vars hjälp arbetet kunde utföras mera effektivt. Det tidigare systemet gick ut på att skapa en visuell bild av hur sidan skulle se ut, och koda sajten efter bildens specifikationer. Detta orsakade mycket strid med normalisering av kodsnuttar och allt måste göras skilt anpassat. Det krävdes därmed mera tid att utveckla sidorna.

Det nya systemet skulle basera sig på färdiga blockelement som man bygger upp sidan med. Detta skulle också medföra en normalisering av olika element med definiering av så mycket globala variabler som möjligt, och försök att undvika ensamma och alltför specifika stilregler. Blockelementen skulle också byggas och innehålla mycket inställningar för visuellt varierande framträdanden i olika projekt.

2 CMS OCH BIBLIOTEK

Hela arbetet byggs upp på Wordpress och man försöker därmed följa dess dokumentation så bra som möjligt. Arbetet byggs på ett gammalt Levelup-tema och har som syfte att göra största delen av modifikationerna via Wordpress egna API (Application programming interface). Vissa tilläggsprogram krävs i varje fall för att inte göra arbetsmängden för stor.

2.1 Wordpress

Hela projektet byggs som ett Wordpress-tema. Wordpress är de mest populära CMS-verktyget (Content Managemed System), ett innehållshanteringssystem som ständigt utvecklas vidare. För ett par år sedan skulle något annat CMS ha varit mera behändigt

för projektet eftersom Wordpress ursprungligen är en plattform för bloggande. Under de senaste åren har det utvecklats vidare och det passar också bra för mer avancerad utveckling av nätsidor. (WPBeginner 2015)

2.2 WP:s filosofi

En av de viktigaste delarna i produkten är att den är lätt att använda och bland annat ger användaren möjlighet att justera inställningar från Wordpress egna kontrollpanel. Som utvecklare tänker man inte alltid på hur andra ser på ditt system. “Many end users of Word-Press are non-technically minded” måste därför beaktas vid utvecklingen av element. “Every time you give a user an option, you are asking them to make a decision. When a user doesn’t care or understand the option this ultimately leads to frustration.” (Wordpress Philosophy 2016) Detta innebär att innehållet är det viktigaste på nätsidorna och hen som bestämmer över innehållet borde inte behöva arbeta med inställningar för att något ska se bra ut.

2.3 Bibliotek

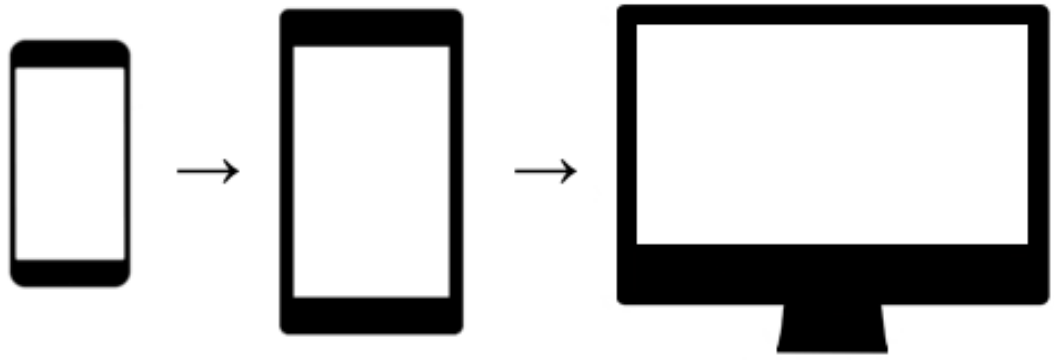
Idag använder man sig speciellt i kodning av webbprodukter, mycket färdiga moduler och kod som någon annan gjort. Wordpress är också känt för att ha en hel mängd färdiga tilläggsprogram tillgängliga. Det finns oavsett en risk att sidoladdning blir slöare ifall man implementerar för många tilläggsprogram, så det är viktigt att tänka efter om det lönar sig att implementera dem eller inte. Vissa tilläggsprogram bygger på varandra men de kan också totalt förstöra varandra. Det är därmed viktigt att kolla kompabilitet och hålla programmen uppdaterade. (Instantshift 2015)

2.3.1 Javascript/jQuery

I modern design kan man då och då behöva något mer komplexa funktioner, och det skulle inte vara möjligt utan JavaScript(JS) och utan dess bibliotek jQuery. jQuery är en betydligt användarvänligare extension till JS och har mycket färdiga funktioner i sig. Det som i Javascript kan kräva flera rader kod, kan göras med jQuery betydligt kortare och mera effektivt. Nackdelen är att koden inte körs lika snabbt men så länge som det inte handlar om stora processer så kommer man inte att märka någon skillnad. (jQuery 2016)

2.3.2 Bootstrap

Bootstrap är det mest kända och använda CSS-ramverket. (KeyCDN 2016) Med hjälp av det kan man lätt koda responsiva sidor. Sidorna byggs med tanken "mobile-first" (se figur 1), så man ger ut element i den ordning man vill ha dem att synas på en mobiltelefon. När skärmbredden därefter blir bredare när man använder tabletter och datorer, flyttar elementen på ett bestämt sätt bredvid, under och ovanpå varandra. För projektet valde man att använda Bootstrap CSS-ramverket för att det redan har använts tidigare på Levelup och nästan kan ses som en standard för webbdesign i dag. Om någon annan skulle koda vidare på bottnen så skulle hen mera sannolikt behärska Bootstrap än något annat ramverk från tidigare. Alla CSS-ramverk liknar varandra så valet spelar inte en så stor roll. (Bootstrap 2016)



Figur 1. Sidor byggs med Bootstrap från mobil till desktop för responsivitetens skull.

2.3.3 FontAwesome

En ensam utvecklare har satt sin tid på att utveckla en font med endast ikoner. Då grundades FontAwesome. Det har en massa ikoner som i dagens mode inom nätsidoutveckling är ett måsta att ha. Tack vare ACF:s (Advanced Custom Fields) samhälle har det också utvecklats en extension för ACF-tilläggsprogrammet som ger användaren en visuell representation av ikonerna redan i Wordpress administrationspanelen. Användargränssnittet gör det betydligt lättare för kunden att handskas med ikoner om hen så önskar det. (FontAwesome 2016)

2.4 Tilläggsprogram

I huvudsaken använder vi oss av bara ett tilläggsprogram ACF. Det ger oss möjligheten att grunda specialgjorda fält för de olika sidorna. (ACF 2016) Denna plugin utgör grunden för hela examensarbetets koncept att snabbt bygga sidor utan att behöva röra någon kod.

3 UTVECKLING

När målsättningen var klar så var det dags att göra en utvecklingsplan. Det är bättre att planera allt klart, göra bevis på koncept och testa saker innan man bestämmer sig för något. Planeringen gick ut på att bestämma ordningen som saker skulle göras.

- Basinstallationen och inställningar
- Studerande av Wordpress Theme API
- Användning av Wordpress Theme API
- CSS-stilar med variabler
- Cache
- Blockelement

3.1 Bevis på koncept

Levelup hade redan från tidigare ett par bevis på att det planerade konceptet fungerar någorlunda, men de hade aldrig utvecklat det desto vidare. I ett par projekt hade konceptet utvecklats vidare i små delar men man hade aldrig gjort det hela till ett helt paket. Kodarna hade gemensamt beslutat att detta paket är något som borde utvecklas. Möjligheten att ha mera globala variabler och återanvändbara blockelement gör byggande av sidor betydligt snabbare.

3.2 Grunderna

Allt började med en installation av Wordpress samt Advanced Custom Fields på en tom utvecklingsserver som vi har på Levelup. Valet att använda en dedikerad server istället för en gratis lokal server bygger på säkerhet samt funktion. Med en lokal server kan man aldrig riktigt veta hur bra den fungerar eller om den fungerar överhuvudtaget. (Serverfault 2012) Genom att ha filerna på en extern server kan man alltid få tag på dem via FTP och testsidan fungerar som en normal nätsida. Många föredrar lokala servrar över skilda utvecklingsservar för att inte behöva föra data över nätet.

3.3 Wordpress Teman

Varje Wordpress sida använder sig av ett tema, “theme”. Dessa teman bestämmer inte bara hur sidbottnen ser ut utan de kan ha flera funktioner inuti. Funktionerna kan vara väldigt extensiva och göra stora ändringar på sidor. Med hjälp av functions.php-filen kan man bygga olika funktioner till Wordpress och köra dem vart de behövs med olika hooks. Dessa hooks är en slags kopplingssystem med vilka man kan koppla funktioner till olika platser i Wordpress egna kod. Vårt tema har planerats att använda sig mycket av functions.php för att bygga till UI-element i “Theme Customizer” sektionen.

3.4 Wordpress Theme API

Wordpress Theme API ger utvecklaren möjlighet att utveckla Wordpress kontrollpanelens delar som i sin tur kan påverka olika element på sidan. Man kan bl.a. bestämma en helt ny sektion till kontrollpanelen som har bara temats egna inställningar.

Vi beslöt oss för att använda oss av "Theme Customizer" för att kunna ändra på de mera globala inställningarna av vår Wordpress tema utan att behöva grunda någon ny meny. En ny meny gjordes ändå för användning av information som oftast kommer upp på varje sida, såsom headern (sidhuvudet) och footern (sidfoten). Resten av sidans inställningar kan varje element själv bestämma över.

Första testerna med Theme API gick ut på att använda Wordpress egen exempelkod, och på att justera det för önskat resultat. För att göra ett element till Theme Customizer och ha det värdet sparat i databas krävs det tre olika element som arrays:

- En inställning (Setting)
- En sektion (Section)
- En kontrollenhet (Control)

(Wordpress Theme Customization API 2016)

3.4.1 Inställning

Inställningar krävs för att spara värdet på den variabel som kan ändras via Theme Customizer (se figur 2). Genom att använda en kontroll "wp_customize" behöver man inte göra någon databashämtning utan det görs automatiskt vid sidoladdning.

```
$wp_customize->add_setting( 'levelup_setting',
    array(
        'default' => 'theme_1',
        'type' => 'theme_mod',
        'capability' => 'edit_theme_options',
        //'transport' => 'postMessage',
    )
);
```

Figur 2. Wordpress Theme API wp_customize inställning.

Värdet ”transport” är bortkommenterat för att det ska använda standardvärdet ”refresh”. Hela ”transport”- värdets mening är att justera hur innehållet ändras i Theme Customizerns förhandsvisning av sidan. ”postMessage”- värdet skulle kräva en JavaScript funktion för att kunna hantera bara ändringarna, och därmed klara av att göra ändringarna snabbt. ”refresh”- standardvärdet tvingar däremot fram en omladdning av sidan för att ändringarna ska synas. På grund av sättet vi gör ändringar på kommer detta inte heller att fungera för att den laddar om sidan via Ajax (Asynchronous JavaScript and XML), utan att göra ett nytt HTTP-anrop (Hyper Text Transport Protocol). Ifall man efter ändringarna laddar om hela Theme Customizer-sidan så syns ändringarna i förhandsvisningen. Alla ändringar fungerar oberoende direkt på live-sidan så det handlar mera om kosmetiska detaljer för administratörer.

3.4.2 Sektion

En sektion definierar en inställningsplats i menyn i Theme Customizer. Detta är bra att ha ifall man vill gruppera sina anpassade inställningar. ”priority” värdet definierar vad inställningens prioritet är i Theme Customizerns lista och definierar ordningen för hur inställningarna skall synas (se figur 3).

```
$wp_customize->add_section( 'levelup_options',
    array(
        'title' => __( 'Levelup Theme Options' ),
        'priority' => 150,
        'capability' => 'edit_theme_options',
        'description' => __('Pick the theme you like or customize theme colorschemes'),
    )
);
```

Figur 3. Wordpress Theme API wp_customize section.

3.4.3 Kontroll

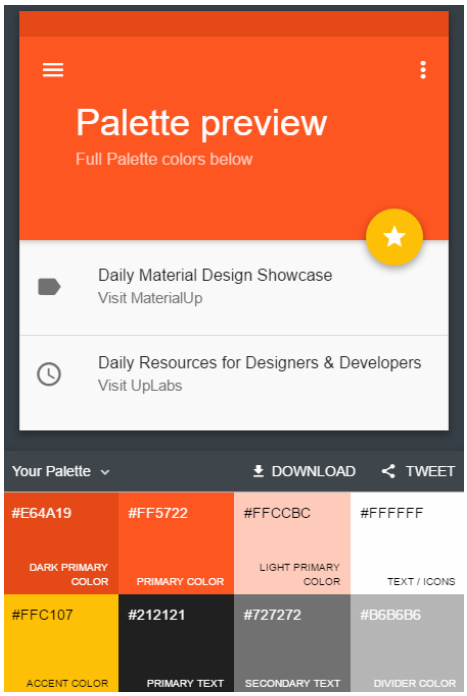
Kontrollen är det som binder ihop inställningarna och sektionerna. Den utgör också inställningen för vad man vill justera i Theme Customizer (se figur 4).

```
$wp_customize->add_control( 'levelup_control',
    array(
        'type' => 'select',
        'label' => __( 'Select theme' ),
        'section' => 'levelup_options',
        'settings' => 'levelup_setting',
        'priority' => 150,
        'choices'    => array(
            "theme_1" => "Theme 1",
            "theme_2" => "Theme 2",
            "theme_3" => "Theme 3"
        ),
    )
);
```

Figur 4. Exempel på hur inställning och sektion kombineras för bruk i Wordpress Theme API.

3.5 Färgpalett

För att få en passande färgpalett användes Materialpalette.com, som genererar färdiga färgscheman för Androidapplikation (se figur 5), men färgerna kan lika bra användas också i webbdesign. Tjänsten ger åtta färger som passar ihop med varandra. Därmed krävs det också flera element i functions.php, så det enda kloka att göra i detta skede var att dela upp varje färgschema i sin egen fil. Som tur är det väldigt enkelt att med PHP inkludera filer. (Materialpalette 2016)



Figur 5. En färgpalett på Materialpalette.

3.6 Style.php

Wordpress exempelkod orsakade utmatning av CSS-stilar i headern vilket inte är önskvärt då den antagligen inte överskrider all annan CSS på sidan. Detta är också väldigt ineffektivt och en dålig vana. Vi behövde alltså en CSS-fil som stöder utmatade variabler från functions.php. Genom sökning hittade vi en dynamisk funktion som kunde fungera som lösning för att överföra variabler från functions.php till ett separat dokument style.php. Detta fungerar eftersom PHP stöder HTML-kod, och därmed också CSS. För att få filen länkad in som en CSS-fil fastän den är PHP, användes en hook, *wp ajax (action)* som delvis fungerar på samma sätt som Wordpress normala egna sätt att länka filer, *wp_enqueue_script/style*.

wp ajax (action) är en ganska speciell hook vars action-namn varierar dynamiskt. Den varierar dynamiskt med ”*nopriv*” beroende på om du är inloggad eller inte. Genom att inkludera filen med PHP-scriptet ”*include*” i ”*-get_style*” action, kan PHP-filen läsas som en stilfil (se figur 6). Style.php-filen läses därmed först av servern och behandlas som vilken PHP-fil som helst. Därefter skickas filen till klienten som tar emot den som en stilfil. (Wordpress wp_ajax_(action), Codex 2016)

```php
add_action('wp_ajax_nopriv_get_style', 'parse_style');
add_action('wp_ajax_get_style', 'parse_style');
function parse_style()
{
    include "style.php";
    wp_die();
}
```

Figur 6. Dynamisk action för stilinkludering.

Ett litet problem var också caching av style.php-filen, för att som tidigare nämnts, kör PHP på servern då sidan laddas, och detta cachas då inte. Med olika ”header”- variabler i style.php kan man ändå göra den cachebar som figur 7 visar.

```php
header("Content-type: text/css; charset: UTF-8");
$expiry = 604800; // (60*60*24*7)
header('Expires:'.gmdate('D, d M Y H:i:s', time() + $expiry) .' GMT');
header("Cache-Control: public, max-age=$expiry, pre-check=$expiry");
header("Pragma: cache");
```

Figur 7. Tvingande av caching i en fil.

Detta gör att filen laddas på första sidoladdningen, men efter det finns den kvar i webbläsarens minne för framsidan och alla undersidor. För felsökning kan man i webbläsaren stänga av caching av sidornas innehåll, och då tvingas webbläsaren alltid att få det senaste innehållet.

Själva CSS-variablerna fungerar genom att man först definierar dem, samt har en reservplan ifall något inte fungerar som det ska. Man låter hellre PHP skriva ut en ständig variabel eller en tom sträng istället för en felrapport som söndrar sidan. Efter att variablerna har definierats så körs det en kollning på vilken färgschema som har valts i Theme Customizer. Den definierar sedan om standardvärdena för variablerna. Figur 8 förklarar förhållandet mellan värden lite klarare

```php
//Define Levelup color scheme choice
$theme      = get_theme_mod('levelup_setting');

//Default values, defined by http://www.materialpalette.com/grey/grey
$dark_primary_color     = '#616161';
$primary_color          = '#9E9E9E';
$light_primary_color    = '#F5F5F5';
$icons                  = '#212121';
$accent_color           = '#9E9E9E';
$primary_text           = '#212121';
$secondary_text         = '#727272';
$divider_color          = '#B6B6B6';

switch ($theme) {
    case 'theme_1':
        //Theme grey/grey http://www.materialpalette.com/grey/grey ;
        $dark_primary_color     = get_theme_mod('scheme_1_color_1_setting' , '#616161');
        $primary_color          = get_theme_mod('scheme_1_color_2_setting' , '#9E9E9E');
        $light_primary_color    = get_theme_mod('scheme_1_color_3_setting' , '#F5F5F5');
        $icons                  = get_theme_mod('scheme_1_color_4_setting' , '#212121');
        $accent_color           = get_theme_mod('scheme_1_color_5_setting' , '#9E9E9E');
        $primary_text           = get_theme_mod('scheme_1_color_6_setting' , '#212121');
        $secondary_text         = get_theme_mod('scheme_1_color_7_setting' , '#727272');
        $divider_color          = get_theme_mod('scheme_1_color_8_setting' , '#B6B6B6');
        break;

    case 'theme_2':
        //Theme blue/blue http://www.materialpalette.com/blue/blue ;
        $dark_primary_color     = get_theme_mod('scheme_2_color_1_setting' , '#1976D2');
        $primary_color          = get_theme_mod('scheme_2_color_2_setting' , '#2196F3');
        $light_primary_color    = get_theme_mod('scheme_2_color_3_setting' , '#BBDEFB');
        $icons                  = get_theme_mod('scheme_2_color_4_setting' , '#FFFFFF');
        $accent_color           = get_theme_mod('scheme_2_color_5_setting' , '#448AFF');
        $primary_text           = get_theme_mod('scheme_2_color_6_setting' , '#212121');
        $secondary_text         = get_theme_mod('scheme_2_color_7_setting' , '#727272');
        $divider_color          = get_theme_mod('scheme_2_color_8_setting' , '#B6B6B6');
        break;

    case 'theme_3':
```

Figur 8. Hur style.php definierar variabler och bruk av färgschema.

Figur 9 visar hur vi får ut ett värde. Ifall man önskar ta bort alla variabler och ha en ren stil-fil så kan man använda sig av en lätt ersättningsfunktion för varje variabel.

```
/* Variables as own colors, no inverted features */
.dark_primary_color {
    color: <?php echo $dark_primary_color ?>;
}
.primary_color {
    color: <?php echo $primary_color ?>;
}
.light_primary_color {
    color: <?php echo $light_primary_color ?>;
}
.icons {
    color: <?php echo $icons ?>;
}
.accent_color {
    color: <?php echo $accent_color ?>;
}
.primary_text {
    color: <?php echo $primary_text ?>;
}
.secondary_text {
    color: <?php echo $secondary_text ?>;
}
.divider_color {
    color: <?php echo $divider_color ?>;
}
```

Figur 9. Kod på vart variablerna ger ut definierade färgkoder.

Inspirationen till namnen för de olika variablerna kom från en artikel på nätet. Färgschemana innehöll ganska många olika namn. För att bättre skilja på de olika elementen var det bäst att ge väldigt förklarande namn för elementen, såsom artikeln *7+1 tips for naming variables* säger: ”[…]when having descriptive variable names, the overall quality of the software will increase because it will be easier to modify and read the code.” (makinggoodsoftware.com 2009) Beskrivande namn är till stor hjälp för utvecklaren. När meningen därtill är att lätt kunna modifiera koden i efterhand, är beskrivande namn väldigt behändiga att använda.

3.7 PHP OPcache – Caching data

I utvecklingsskedet dök det en dag upp ett problem då inga ändringar man gjorde via FTP (File Transfer Protocol) kunde ses via HTML-laddning. Det kom fram att alla utvecklare på Levelup klagade på samma sak. Efter att ha varit i kontakt med vår servervärd så kom det fram att de i alla deras servrar hade tagit i bruk OPCache, som cachar innehåll på servernivå. Detta i sig tog 3 timmar innan de skickade information om hur vi

kunde kringgå cachen via htaccess, en serverkonfigureringsfil. (PHP Manual - OPcache 2016)

3.8 Tilläggsprogrammet Advanced Custom Fields

Advanced Custom Fields (ACF) är ett tilläggsprogram för Wordpress som ger en möjlighet att göra fältgrupp med flera anpassade fält, och visa detta på alla slags posts. Levelup har skaffat Pro-versionen av programmet, som ger dig möjligheten att använda ett par mera avancerade anpassade fält, ”repeater” och ”flexible content”. Med ”repeater” kan man repetera de fält som är definierade för den så många gånger man vill, och därmed spara tid i stället för att kopiera och klistra kod. ”Flexible content” ger användaren igen möjligheten att välja vilken block-element hen vill ha som följande på sidan. Detta visar då de fält som behövs för det elementet, samt visar elementet på sidan. Eftersom sidorna byggs ”mobile-first”, är det väldigt simpelt att implementera detta.

Eftersom varje sida har element som följer efter oberoende av vilken undersidan man är på, bestämdes det att dela upp ACF-fälten till två, en banderoll och en “sectionator”. ACF-fält är alltså lätta att utveckla vidare. Detta kräver speciellt anpassade fält på sidan, och deras motsvarande positioner i sidbottnen, eller sidobottendelen. De matas ut med korta PHP-taggar. Det är därmed ganska lätt att från hårdkodande flytta sig över till bruket av ACF och dess fält. (Advanced Custom Fields 2016)

3.8.1 Banderoll

Varje sida har oftast en slags banderoll i bruk, och denna kommer alltid först på sidorna efter headern. Därmed ansågs det bättre att ha bannerns fält först på varje sida, och först efter det “sectionatorn”. Banderollen består av ett ”select”-fält, med vilket man väljer hurdan typ av banderoll man kommer att använda. Till en början kodades ett par olika banderoller:

- Default size

- Fullscreen Image
- Fullscreen Video
- Custom Shortcode

För sidan med alla Wordpress posts, dess paginering och alla enskilda posts, beslöts att ha standardbanderoll (Default size), och för posts en ännu mindre banderoll då deras innehåll inte finns eller kommer och finnas i banderollen. ”Fullscreen Video” krävde ett par olika format av videoclip, samt något statiskt att visa till de med sämre maskiner i bruk, alltså en bild. Detta är en av de få delarna som kräver att man justerar på filer via FTP. På grund av behovet av olika videoformat samt för undvikande av för stora video-filer, ska man ersätta video/bild-filerna manuellt på servern. ”Custom Shortode” visar för användaren endast en WYSIWYG-editor (What You See Is What You Get). Denna stöder ”Short codes” vilka är korta kodsnuttar som anropar funktioner definierade att göra något önskat. Flera tilläggsprogram för banderoller har som inställning att visa en sådan banderoll med hjälp av sådana kodsnuttar. Editor stöder ”short codes” och utför deras funktion(er).

3.8.2 Sectionator

Detta är tyngdpunkten i hela projektet. Namnet är bara påhittat för att man lättare ska kunna diskutera om saken med kolleger och kunder. Första namnet var “PageBuilder” men det fanns en plugin som använde sig av samma namn och därmed ansåg vi att det var bättre att byta namnet till något annat.

Idén går ut på att man bygger blockelementen och implementerar dem i ACF ”flexible content”. När ”flexible content” har flera färdiga element i sig, är det väldigt lätt att bygga sidor snabbt utan att koda. Elementen förbrukar färgschemat som har valts och är avsett att vara stilmässigt lika varandra. Planen är för detta projekt att ha en god mängd färdiga element som fungerar på rak arm, men att i framtiden ha ännu mera enskilt anpassade element.

3.8.3 Sectionatoruppbyggnad

Hela sectionator är egentligen en stor *if/while* loop med flera regler för olika händelser (se figur 10).

```
if( have_rows('sectionator_blocks') ):
    // loop through the rows of data
    while ( have_rows('sectionator_blocks') ) : the_row();
        $rowlayout = get_row_layout();
        if( $rowlayout == '1_column' ): ?>

            <?php /* 1 column code */ ?>

        <?php elseif( $rowlayout == '2_columns' ): ?>

            <?php /* 2 column code */ ?>

        <?php elseif( $rowlayout == '2_column_cards' ): ?>

            <?php /* 2 column cards code */ ?>

        <?php elseif( $rowlayout == '3_column_cards' ): ?>

            <!-- 3 Column Cards -->
```

Figur 10. Sectionatorns if/else funktion.

För att minska databasanrop kopplades radsökningen till en variabel. När sidan laddas granskar servern först om det finns sparade rader, ifall det finns granskar den vilken rad som kommer först, och går därefter igenom alla ”elseif”-påståenden. När den hittat rätt påstående körs koden inne i påståendet. I figur 10 är blockelementkoden gjord till kommentarer för att mera tydligt visa strukturen av påståenden.

Koden i sig själv innanför varje påstående börjar med att definiera alla variabler som behövs. I koden behandlas kollningar på alla inställningar som matas in i kontrollpanelen på en sidas innehåll. Det är betydligt tydligare att arbeta med enkla variabler än att ha långa påståenden mitt i HTML-koden. Då man önskar använda en variabel, kan man bara använda PHP ”echo” funktion och få dess värde i koden.

I slutet av varje blockelement körs det ”unset()” av PHP, för att förstöra variablerna som har blivit gjorda för blockelementet. Detta orsakade problem när de inte fanns, när vari-

abler sparades från ett blockelement till en annan. Genom att använda ”unset()” kan man vara säker på att varje blockelement hanterar endast sina egna variabler.

3.9 Blockelementen

Enligt WP-filosofin ska en Wordpress sida vara så lätt att använda som möjligt, och detta speglas också på blockelementen vi byggt. Detta betyder att användaren vill se endast innehållet, inte frågor eller inställningar. Varje element är byggt för att fungera med en slags stil när man endast fyller på innehåll och inte byter inställningar. Inställningarna blev ett litet bekymmer för sig, men det är ändå bra att de finns i blockelementen för att simplifiera arbetet ifall man vill justera något. Därmed beslöts att det endast ska finnas en liten inställning i varje element som heter “Show Advanced Settings”. Detta gömmer då inställningarna för innehållsmataren men ger en mer möjligheter för en att justera på elementen.

3.9.1 Test av koncept med en kund

Konceptet testades med en kund, Makery (Makery.fi), med en tidig version av sectionatorn som då hette “Page Builder 2.3 - Makery edition”. Första feedbacken kom genast vid undervisningstillfället. I undervisningen förklarades det åt kunden hur en sida fungerar och hur kunden själv kan påverka på dess innehåll dynamiskt. Vid utbildningen kom det inte klagomål från kunden utan man tyckte att idén om att flexibelt kunna byta ordning på innehållet var fenomenalt. Efter ett par praktiska exempel kände kunden sig redo för att administrera hens sida.

Efter en tid, ca två veckor till en månad, granskas kundernas sidor för att man ska se ifall de har kunnat använda de verktyg de fått. Med denna kund visade det sig gå bra. Hen hade själv kunnat ändra på innehållet, skapa mera innehåll och dessutom justera på färdigt satta block med avancerade inställningar.

Det fanns dock vissa problem med avancerade inställningar av tomrum runt blocken som inte hade ställts till så fint som i designen. Oberoende var arbetet med kunden en succé, men eventuellt kan det också bara ha varit god tur med en bra kund. Man borde använda slutliga produkten med flera andra kunder för att få mera mångsidig feedback.

3.9.2 Färdiga blockelementen

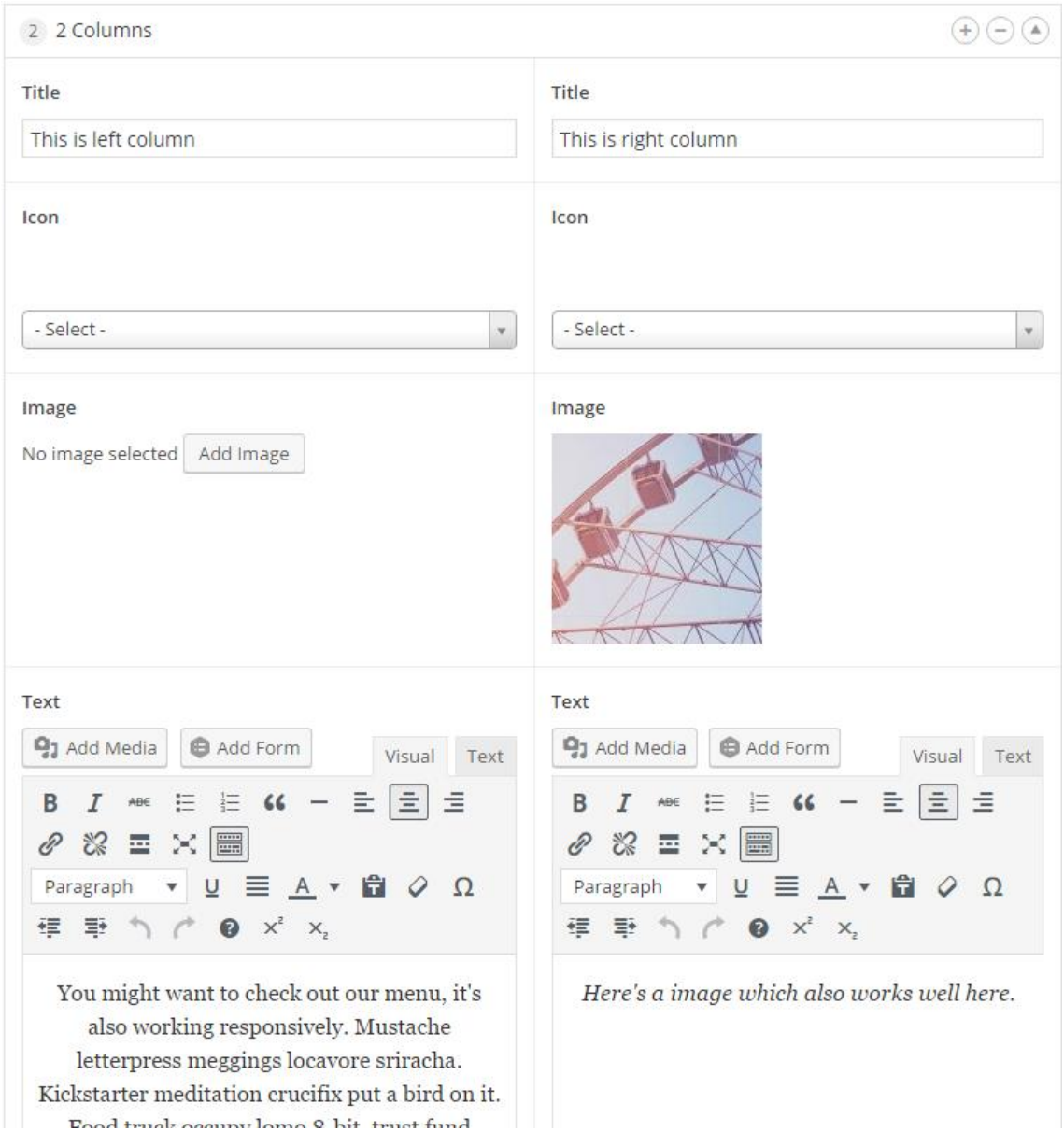
Ofta när man har innehåll till nätsidor så måste man justera det för att de bättre ska passa in. Till en början byggdes de mest vanliga blocken man kan tänka sig:

- 1 Column
- 2 Columns
- 2 Column cards
- 3 Column cards
- 4 Column cards
- News

Med dessa klarar man sig redan ganska långt med byggande av sidor. Varje block-element fungerar för sig själv och har inställningar för att ändra på bl.a. stil, färg, storlek och ikoner. Man kan också vid behov lägga in tilläggsprogrammets ”GravityForms” formulär, och det är färdig en stil för att passa in i varje blockelement.

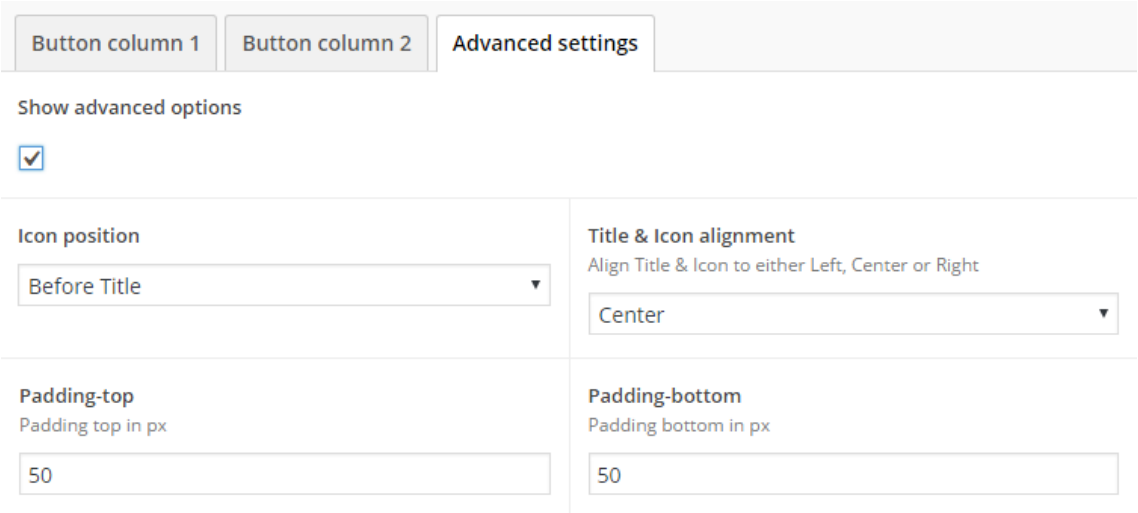
3.9.3 Uppbyggnad av blocken

Med lite hänsyn till hur kunderna använde tidigare konceptet, var det ganska klart hur blockelementen skulle se ut för hen som matar innehållet. Detta är justerbart i ACF med procentuella värden för att skapa system liknande som Bootstrap, dock utan responsivitet. Det var klart att innehållet var det viktigaste, så det kom i första prioritet, och inställningarna fick gömmas i en flik efter allt innehåll. Då blev utseende på block-elementen betydligt bättre och klarare. För element som fungerar i kolumner, delades också inställningarna i kolumner. Figur 11 presenterar hur ett 2-kolumns blockelement ser ut för administratören.



Figur 11. Två kolumners utseende när man ändrar på innehållet på en sida.

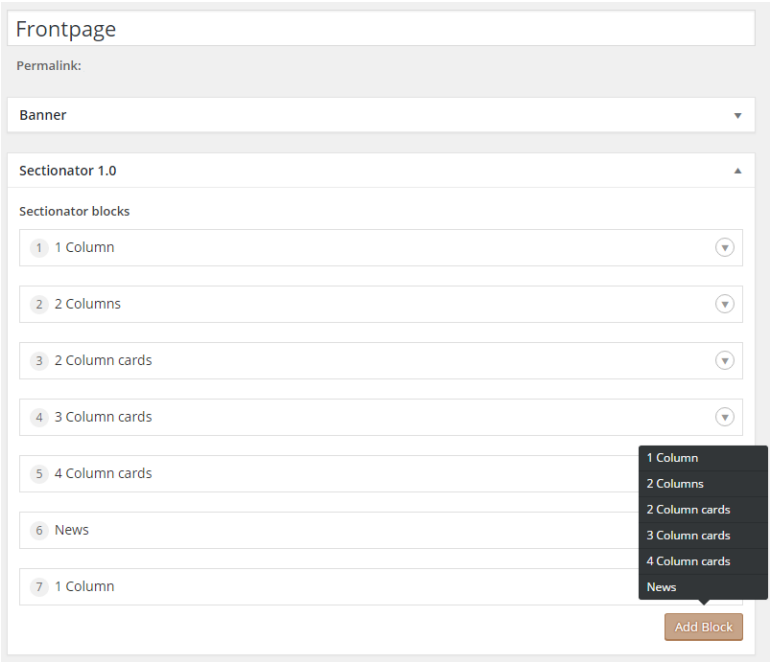
Avancerade inställningarna är gömda till en början. Ifall en kund vill justera på dem kommer de nog åt inställningarna utan problem. Figur 12 visar de avancerade bilderna då ”show advanced options” är aktiverat.



Figur 12. Inställningarna för ett block-element av två kolumner

4 BRUK

När sectionator är inställt för att synas på en sida, kan man lätt sätta mera element per block. Elementen sorteras sedan i den ordning man vill att de syns på sidan. Figur 13 visar detta tydligare.

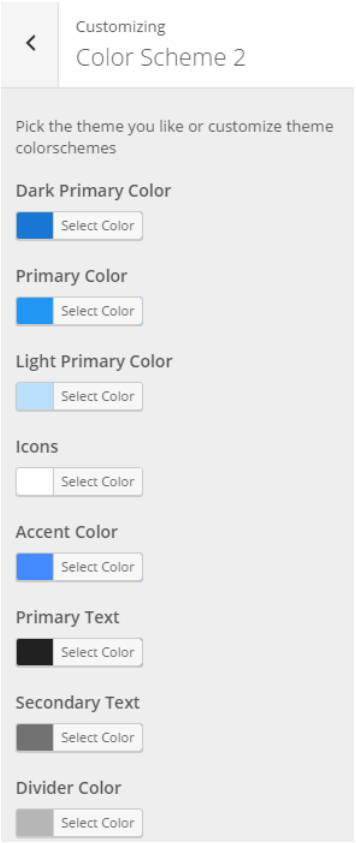


Figur 13. Sectionatorns backend för informationsmatning.

Vart och ett av blocken är kollapsade för att föra mindre utrymme. Vid behov kan man öppna dem och justera på deras innehåll enligt eget tycke.

4.1 Theme Customizer

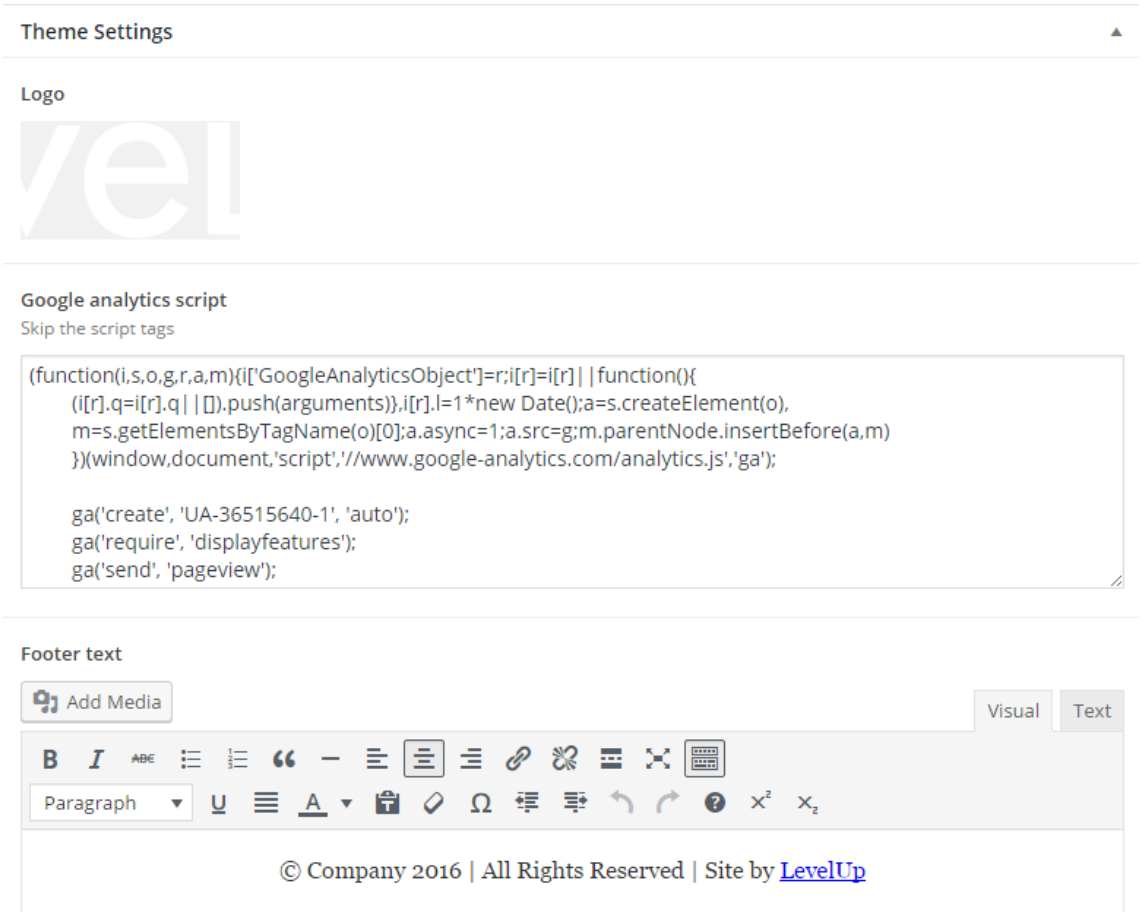
I Theme Customizer kan vi justera på färgscheman och vilket färgschema vi vill använda. Dessa element skapades med koden tidigare, och nu kommer de i bruk. Man kan välja mellan tre olika färgscheman, och dess färger är helt upp till var och en att justera. Det finns totalt åtta olika färger att använda så det torde vara mera än tillräckligt. Färgerna kan ändras med en jQuery färgväljare som hör till Wordpress Theme API. Figur 14 kan lätt ihopkopplas med Figur 5 med hjälp av färgernas benämning.



Figur 14. En färgschemas uppsättning i Theme Customizer.

4.2 Temans inställningar

Med ACF och lite PHP-kod i functions.php kan man göra ett nytt menyelement till kontrollpanelen. Detta är sättet som föreslås användas ifall man vill använda något ACF-element på alla sidor, utan att deras innehåll ändrar. Därmed har det gjorts en ”Theme settings”-meny som innehåller sådant som inte ändrar så ofta. Till en början implementeras logo, Google Analys script och footer innehållet (se figur 15). Det kan ändå lätt tilläggas flera fält vid behov.



Figur 15. Theme Settings, inställningarna för mera statisk innehåll på sidor.

4.3 Kloning av Wordpress installation

Överflyttning av en sajt kan vara trögt. Man måste ha koll på filer, databas-informationen och serverns inställningar. Som tur har Wordpress ett tilläggsprogram också för detta. Duplicator-programmet gör hela sajten till en .zip paket och ger med en installer.php fil. Man sätter sedan dessa två filer via FTP till det stället där index.php ska finnas, därmed oftast inne i ”public_html” mappen. Programmets installer.php-fil gör allt arbete och varnar för problem som kan komma med överflyttningen. Oftast brukar installeringen ändå gå smärtfritt, och betydligt snabbare än själva arbetet att ta backup på databas och filer. Eftersom man gör en exakt kloning av webbplatsen med Duplicator, torde webbplatsen fungera utan problem. Det som ändå kan påverka bruk och funktion är serverns egna inställningar och programvaras version. Det som har varit problem med temat är att den är byggd för PHP version 5.6 när våra kunder ofta har haft sina servrar med PHP version 5.4. Ifall det kommer fram kompabilitetsproblem så är det ändå inte mycket man måste justera på för att ha sajten att fungera igen.

5 PROJEKTETS SLUTRESULTAT

Som slutresultat har Ab Levelup Oy nu ett färdig Wordpressbotten som man snabbt kan bygga mera funktioner på. Eftersom resultatet ännu är nytt är det svårt och säga hur mycket det har hjälpt till, men flera projekt har bottnen som grund för sina sidor. Därmed kan det konstateras att projektet har varit nyttigt för företaget och det kommer att användas i framtiden.

5.1 Framtidsplaner

För att sectionator är lätt att utveckla vidare så kommer den åtminstone få nya block-element till sig. Meningen är att detta tema ska agera som en bra grund för varje projekt, och genom att bygga flera blockelement till den, kan man försnabba utvecklingen i framtiden. Ofta kommer det ändå specialönskemål av kunden som måste tas till beak-

tande, och därmed kommer det att byggas flera element till bottnen. Ifall färgschema-systemet inte fungerar, så är det lätt att koppla bort själva sectionator-delen från temat.

6 DISKUSSION

Efter att projektet var klart kunde man använda slutprodukten för olika projekt på Levelup. Efter en brukstid på tre månader har resultatet tillämpats delvis i tre projekt, och fullständigt i tre andra.

En kollega, som också är utvecklare, har gett feedback på resultatet och anser det vara bra, men han har själv lite idéer om hur man skulle kunna utveckla den ännu vidare. I ändringssyfte önskade han att en CSS-regel skulle justeras för att skala bilderna på ett annat sätt, men annars var han själv nöjd med de visuella utseende.

Mitt eget förbättringsförslag skulle vara mera dynamiska färgändrare i kontrollpanelen som faktiskt passar in bättre med sajtens egen färgvärld. Problem kom fram då sidor skulle ha flera huvudfärger, och då blev ”$primary_color” inte riktigt rätt justerat. Bottnen fungerar bättre för mera simpla och uniforma sidor. Detta har inte designer än tagit till beaktan och det har orsakat att vi oberoende måste utveckla kod speciellt för vissa projekt.

Ur Levelups designers synpunkt begränsar denna botten hens rättigheter att designa exakt vad hen vill. Eventuellt måste designern också delta i planering av nya element för bottnen och via det också försöka använda det i hens designer.

Hela konceptet slutligen som kodlös webbutveckling har visat sig bli ganska begränsat. Med vidare utveckling av bottnen kommer vi att kunna bygga sidor snabbare och med mindre justering av kod, men jag själv tvivlar om att slippa helt kodningen i projekt. Kanske ett sätt skulle vara lämna designern utanför planeringen och ha som utvecklare helt fria händer med utseende av en sajt. Då skulle man använda den botten som man har färdigt och lämpa innehållet i färdiga bottnen.

CENTRALA BEGREPP

Nedan följer en uppräkning av de centrala termerna som används i arbetet.

WP = Wordpress

CMS = Content Management System (Innehållshanteringssystem)

API = Application Program Interface

ACF = Advanced Custom Fields

CSS = Cascade Styling Sheet

HTML = Hyper Text Mark Language

Ajax = Asynchronous JavaScript and XML

PHP = Hypertext Preprocessor, Kod som körs då sidan laddas, utförs på serversidan

functions.php = Central fil för funktioner inom Wordpress

Bootstrap = Ett CSS-ramverk för sidor som är skalbara från mobil till dator

htaccess = En fil som konfigurerar Apache serverns beteende

FTP = File Transfer Protocol, sätt och överföra information mellan datorer och servrar

Hårdkodande = Kod som är skrivet manuellt och är inte lätt ljusterbart av en normal användare

KÄLLOR

WPBeginner. 2015, Why You Should Use WordPress?
Tillgänglig: http://www.wpbeginner.com/why-you-should-use-wordpress/ Hämtad 28.6.2016

Wordpress, Philosophy. Tillgänglig: https://wordpress.org/about/philosophy/
Hämtad 28.6.2016

Instantshift.com 2015, Can A Large Number Of Plugins Affect Your WordPress Site's Performance? Tällgänglig: http://www.instantshift.com/2015/12/15/plugins-and-wordpress-performance/ Hämtad 30.8.2016

jQuery, About jQuery UI. Tillgänglig: http://jqueryui.com/about/ Hämtad 28.6.2016

Bootstrap (version 3.3.6), Getting started.
Tillgänglig: http://getbootstrap.com/getting-started/ Hämtad 28.6.2016

FontAwesome (version 4.7). Tillgänglig: http://fontawesome.io/ Hämtad 28.6.2016

Advanced Custom Fields. Tillgänglig: https://www.advancedcustomfields.com/
Hämtad 28.6.2016

Materialpalette. Tillgänglig: https://www.materialpalette.com/ Hämtad 28.6.2016

PHP OPCache. Tillgänglig: http://php.net/manual/en/intro.opcache.php
Hämtad 28.6.2016

Plugin API/Action Reference/wp ajax (action), Wordpress Codex.
Tillgänglig:
https://codex.wordpress.org/Plugin_API/Action_Reference/wp_ajax_(action)
Hämtad 28.6.2016

Making good software. 2009, 7+1 tips for naming variables,
Tillgänglig: http://www.makinggoodsoftware.com/2009/05/04/71-tips-for-naming-variables/ Hämtad 28.6.2016

Wordpress, Theme Customization API. Tillgänglig:
https://codex.wordpress.org/Theme_Customization_API Hämtad 30.8.2016

KeyCDN, Front-end-frameworks. Tillgänglig:
https://www.keycdn.com/blog/front-end-frameworks/ Hämtad 23.11.2016

Serverfault, Why not use a WAMP stack? Tillgänglig:
http://serverfault.com/questions/453617/why-not-use-a-wamp-stack
Hämtad 23.11.2016