Izrada teme za Wordpress CMS korištenjem SASS-a i Gulp-a

Žganec, Domagoj

Sveučilište
Sjever

Završni rad br. 496/MM/2015

Izrada teme za Wordpress CMS korištenjem SASS-a i Gulp-a

Domagoj Žganec, 5422/601

Varaždin, lipanj 2016. godine

Sveučilište
Sjever

Odjel za multimediju, oblikovanje i primjenu

Završni rad br. 496/MM/2015

Izrada teme za Wordpress CMS korištenjem SASS-a i Gulp-a

Student

Domagoj Žganec, 5422/601

Mentor

Vladimir Stanisavljević, mr. sc

Varaždin, lipanj 2016. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Domagoj Žganec | MATIČNI BROJ: 5422/601

DATUM: | KOLEGIJ: Programski alati 3

NASLOV RADA: Izrada teme za Wordpress CMS korištenjem SASS-a i Gulp-a

NASLOV RADA NA ENGL. JEZIKU: Development of a Wordpress theme by using SASS and Gulp

MENTOR: mr.sc. Vladimir Stanisavljević | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. doc.dr.sc. Dejan Valdec - predsjednik
2. dr.sc. Ladislav Havaš, v.pred. - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. dr.sc. Robert Logožar, v. pred. - zamjenski član
5.

Zadatak završnog rada

BROJ: 496/MM/2016

OPIS

Wordpress CMS danas je zasigurno najznačajnij sustav za upravljanjem sadržajima na internetu. Već osnovna inačica tog sustava otvorenog koda nudi brojne mogućnosti za izradu web stranica. Međutim, u stvarnom radu nerijetko se pojavljuje potreba za izradom vlastitih proširenja za neke specifične potrebe koje nisu podržane u osnovnoj inačici ili javno dostupnim dodacima. U tom slučaju potrebno je pristupiti izradi vlastitih priključaka (engl. plugin) ili tema za Wordpress. Razvoj tema može se ubrazati korištenjem suvremenih tehnologija za oblikovanje web stranica poput preprocesora oblikovnih predložaka (engl. stylesheet) poput SASS-a, a sam postupak testiranja rješenja automatizirati alatima poput Gulp-a.

U ovom radu potrebno je:
* osnovnu arhitekturu Wordpress sustava upravljanjima sadržajima i načine dopune pomoću priključaka i tema,
* isprobati i opisati izradu jednostavnih priključaka na tipskim primjerima,
• razmotriti dodatne web tehnologije koje mogu olakšati izradu priključaka i usporediti ih,
* opisati osnove preprocesora oblikovnih predložaka i detaljno obraditi SASS,
* opisati sustav Gulp i kako se može iskoristiti za postupak izrade Wordpress tema,
• izraditi složeniji priključak za Wordpress za primjenu u nekom realističnom projektu primjenom svih prikazanih tehnologija.

Detaljno opisati sve korištene tehnologije i korake u radu potrebne da bi se ostvarilo traženo te detaljno opisati stečena iskustva i postignute rezultate.

ZADATAK URUČEN: 23.09.2016. | POTPIS MENTORA:

REPUBLIKA HRVATSKA 5 SVEUČILIŠTE SJEVER KOPRIVNICA SVEUČILIŠTE SJEVER

Predgovor

Tema ovog završnog rada izabrana je zbog osobne želje da se prikažu novije tehnologije koje se danas koriste u web developmentu. Ovim radom želim steći više iskustva te proširiti svoje znanje. Siguran sam da će mi novo iskustvo dobiveno pisanjem ovog rada dobro doći u budućnosti.

Današnje tehnologije razvijaju se nevjerojatnom brzinom. Već sam čuo za neke koje su ogroman korak naprijed s obzirom na one koje obrađujem u ovom radu. Jedini način da držimo korak sa novim tehnologijama je da nikad ne prestanemo učiti i raditi na vlastitom razvoju.

Sažetak

Ovaj rad opisuje izradu teme za CMS Wordpress s ACF dodatkom. ACF dodatak (engl. *Advanced Custom Fields Plugin*) proširuje administratorovo sučelje tako da admin može dodavati gotove i stilizirane elemente (npr. klizač, citat, CTA link itd.) koji su definirani u samoj temi od strane autora teme. Osim mogućnosti dodavanja gotovih elemenata na stranicu, ACF dodatak nam omogućuje da konfiguriramo i dodatne globalne postavke (poput izbornika u zaglavlju, teksta u podnožju i sl.).

Naglasak rada je na samoj izradi teme za CMS Wordpress. Pri izradi teme koristio se *Gulp*, alat koji služi za automatiziranje repetitivnih zadataka u web developmentu poput osvježavanja preglednika, minimiziranja koda i slično. Umjesto klasičnog CSS-a koristio se SASS (*Syntactically Awesome Style Sheets*) s kojim možemo pisati CSS koristeći novu, bolju sintaksu. SASS se mora kompilirati u klasičan CSS. To je također riješeno primjenom *Gulpa*. Osim toga korišteni su *Bourbon* i *Neat* (SASS dodaci koji u sebi sadrže mnoštvo predefiniranih i često korištenih elemenata).

Od ostalih programskih alata, korišten je sustav virtualizacije Vagrant i JavaScript programska zbirka Modernizr. S Vagrantom smo pokrenuli drugi operacijski sustav (Linux) unutar postojećeg (Windows) radi boljeg razvojnog okruženja. Modernizr je alat s kojim se mogu provjeriti mogućnosti Internetskog preglednika (poput podrške za vektorsku grafiku, CSS3 animacije i tranzicije, HTML5 videa i sl.) te implementacije alternative ukoliko preglednik ne podržava pojedinu mogućnost.

Ključne riječi: Gulp, SASS, ACF, Vagrant, Modernizr, Wordpress, Tema

Summary

This final paper describes the procedure of making a Wordpress CMS theme with ACF plugin. ACF plugin (Advanced Custom Fields) expands admin user interface so that admin can add defined and stylized elements (like Slider, Quote, CTA link etc.) which are defined in the theme itself and created by the author of the theme. Besides the feature to add defined elements on the page, ACF allows us to configure additional global settings (like links in a header, text in a footer etc.).

The paper focuses on making the theme itself. While developing it, Gulp was used, a tool that automates repetitive tasks in web development like browser refreshing, code minimizing etc. Instead of classic CSS, we used SASS (Syntactically Awesome Style Sheets) with whom it is possible to write CSS using new, improved syntax. SASS has to be compiled into classic CSS. Compiling was also automated with Gulp. Bourbon and Neat are SASS frameworks that contain a lot of predefined elements that are often used.

Other tools that were used are Vagrant and Modernizr. Vagrant was used to start another operating system (Linux) within our current (Windows) to make a better work environment. With Modernizr, it is possible to check the abilities of a browser (like support of vector graphics, CSS3 animations and transitions, HTML5 video etc.) and define alternatives in case browser doesn't support certain feature.

Keywords: Gulp, SASS, ACF, Vagrant, Modernizr, Wordpress, Theme

Popis korištenih kratica

HTML	HyperText Markup Language Sintaksa za obilježavanje hipertektualnih dokumenata.
CSS	Cascading Style Sheets Stilski jezik za definiranje izgleda elemenata web-stranice.
SASS	Syntactically Awesome Style Sheets Unaprijeđena CSS sintaksa.
CMD	Command prompt Windows-ovo tekstualno sučelje.
SVG	Scalable Vector Graphics Format slike vektorske grafike.
ACF	Advanced Custom Fields Wordpress-ov plugin razvijen od strane Elliota Condona.
NPM	Node Package Manager Velika kolekcija gotovih JavaScript modula.
CMS	Content Management System Sustav za upravljanje sadržajem web mjesta.

Sadržaj

1. Uvod

U praksi se često pojavljuje potreba za izradom vlastitih proširenja iz razloga što klijenti mogu imati različite zahtjeve i želje koje se tiču funkcionalnosti ili dizajna stranice. Srećom, postoje tehnologije koje mogu ubrzati i olakšati izradu takvih proširenja te osigurati njezinu optimizaciju za web.

Započet ćemo s detaljnim opisom svake od korištenih tehnologija. Nakon toga ćemo opisati izradu teme. Ovaj završni rad (pogotovo poglavlje 7) je zamišljen kao neka vrsta vodiča (ili *tutorial*). Za potpuno razumijevanje svega prikazanog predlaže se pomno čitanje i razumijevanje (pogotovo programskog koda). Isto tako čitatelj će imati najviše koristi ako sam isproba kod razvijen u praktičnom dijelu rada.

Današnji web development je puno više od statičnog HTML-a i CSS-a. Danas je neizbježno znati osnove programiranja i programski jezik JavaScript da bismo mogli zadovoljiti moderne standarde. Moram napomenuti da ovo možda i nisu najmodernije tehnologije. Sve tehnologije koje će se koristiti u ovom radu na snazi su već barem nekoliko godina.

Za početak, osvrnut ćemo se na Vagrant [1]. S obzirom da Windows ima neke nedostatke [2] što se tiče developmenta. Vagrant nam omogućava da unutar trenutnog radimo na drugom operacijskom sustavu (Linux [3]). Nakon Vagranta objasnit ću SASS (od engl. *Syntactically Awesome Style Sheets*) [4]. SASS je u svojoj osnovi samo napredni CSS [5]. Možemo pisati i klasični CSS u SASS-u, no SASS nam nudi mogućnost bolje sintakse (opisano u poglavlju 3) i još puno toga. SASS se prije prikaza mora kompilirati u klasičan CSS da bi ga preglednik razumio. Isto tako koristit ćemo Bourbon i Neat, *frameworke* pisane u SASS-u koji nam donose puno predefiniranih, često korištenih elemenata (primjer: stilizirani gumb). Da ne bismo morali ručno kompilirati SASS svaki put kad spremimo datoteku koristit ćemo Gulp koji nam pomaže pri automatizaciji našeg *radnog toka* (engl. *workflow-a)*. Osim kompiliranja SASS-a, pomoću Gulpa njim možemo i automatski spojiti i minimizirati .js datoteke, konfigurirati si automatsko osvježavanje preglednika čim napravimo neke promjene i još mnogo toga. Nakon Gulp-a slijedi Modernizr [6], *library* s kojim provjeravamo mogućnosti preglednika. Modernizr ima u sebi mnogo tih mogućnosti (primjeri: podržavanje SVG [7] formata, CSS3 animacija i tranzicija [8], HTML5 video [9] i još mnogo toga). Naravno ne treba nam sve, na stranici Modernizr-era možemo odabrati koje mogućnosti želimo. Mi smo odabrali samo mogućnost podržavanja SVG formata. Zatim dolazimo do ACF plugina [10] za Wordpress. Pretpostavit ćemo da neki klijent koji je kupio našu temu ne zna kodirati stoga treba imati konfigurirano sučelje s kojim može upravljati sadržajem. Iako Wordpress ima nekakvo takvo sučelje, mi želimo bolje od toga, tj.

želimo da klijent ima mogućnost ubacivanja gotovih elemenata poput klizača, velikog naslova itd. Za svaki element moći će birati neke ključne vrijednosti poput slika koje idu u klizač ili teksta koji će pisati u naslovu. Pa počnimo onda s Vagrantom.

2. Vagrant

Vagrant [1] je alat za izradu gotovih razvojnih okruženja. Da bismo ga objasnili, prvo moramo objasniti sljedeća tri pojma: Virtualni stroj, VirtualBox i Provisioning.

2.1. Virtualni stroj (Virtual machine)

Virtualni stroj je računalni program koji stvara virtualni računalni sustav. To virtualno računalo radi kao proces u prozoru na našem trenutnom operativnom sustavu. [11]

Virtualni stroj je izolirani dio našeg sustava koji "misli" da je računalni sustav za sebe. Npr. ako imamo procesor sa četiri jezgre, 12 GB RAM-a i 50 GB prostora na hard disku, možemo dodijeliti jednu jezgru, 4 GB RAM-a i 20GB prostora u virtualni stroj. Taj virtualni stroj će misliti da ima na raspolaganju upravo te resurse. To ima nekoliko prednosti. Ukoliko slučajno skinemo virus unutar našeg stroja, samo će stroj biti u opasnosti od tog virusa jer stroj nema povezanosti sa stvarnim sustavom, te može lako biti ugašen i ponovno konfiguriran, bez posljedica. Isto tako, možemo ga koristiti za testiranje aplikacija na drugim operacijskim sustavima [11].

2.2. VirtualBox

VirtualBox [12] je program s kojim možemo lako i brzo napraviti virtualne strojeve. Vrlo je popularan jer je *open-source* i potpuno besplatan. Od drugih takvih programa možemo izdvojiti Vmware Player.

2.3. Provisioning

Kad jednom napravimo virtualni stroj, u njemu nema ničega osim operacijskog sustava kojeg smo instalirali (bez, dodatnih aplikacija, drivera i slično). Jedan od načina da se to riješi je upravo *provisioning*, tj. korištenje predefinirane skripte koja će sve instalirati za nas. *Provisioner* je poseban program koji radi upravo to za nas.

2.4. Vagrant

Vagrant je program koji kombinira VirtualBox i Provisionera, te konfigurira virtualni stroj za nas. Obično bi trebalo puno vremena da se virtualni stroj učita jer tu govorimo o cijelom operacijskom sustavu. Vagrant makne sve one stvari koje nama ne trebaju jer je orijentiran samo za razvoj aplikacija. Vagrantovi virtualni strojevi nemaju nikakvih grafičkih elemenata.

Koristimo ih isključivo preko naredbene ljuske ili neke druge vrste terminala s lokalnog ili udaljenog računala.

Kod predefiniranih paketa za PHP, Apache, MySQL poput XAMPP-a često dolazi do situacije da nešto radi na našem računalu, no ne i na nekom drugom. S Vagrantom imamo alat za automatizirano testiranje koda na više različitih konfiguracija servera.

2.5. Korištenje Vagranta

Jednom kad ga preuzmemo i instaliramo[1] moramo odabrati virtualni stroj koji želimo. Možemo skinuti gotovu konfiguraciju [13] ili kreirati svoju. U našem slučaju, kreirali smo svoju koristeći PuPHPet [14]. Odaberemo koju Linux distribuciju želimo koristiti (Ubuntu [15], Debian [16] itd.), bazu podataka, programske jezike i drugo. Na kraju skinemo našu konfguraciju u .rar formatu te je izvezemo u željeni direktorij. U terminalu odemo u taj direktorij i upišemo *vagrant up*. Vagrant će tada početi skidati sve pakete potrebne za rad. Inicijalni *vagrant up* može potrajati 20-tak minuta. Jednom kad uspješno obavimo inicijalni vagrant up, imamo na raspolaganju virtualni stroj koji ćemo koristiti za naš projekt.

3. SASS

Jednostavnost CSS-a je uvijek bila jedna od njegovih glavnih karakteristika. CSS stilovi su samo dugačke liste pravila koje se sastoje od selektora i stilova koji se za njih primenjuju. [17] Međutim, s vremenom su web stranice postale sve kompliciranije pa je došlo i do kompliciranja koda. SASS (Syntactically Awesome Style Sheets) [4] donosi novu sintaksu koja je u potpunosti kompatibilna sa klasičnom CSS sintaksom, no omogućava i neke nove, izrazito korisne mogućnosti poput varijabla, mixina, operatora itd.

SASS je preprocesor što znači da ga je potrebno kompilirati kako bi dobili klasičnu CSS datoteku koju onda uključimo u naš projekt. Da bi SASS radio, potrebno je instalirati Ruby. Instalacija SASS-a moguća je preko CMD-a i preko aplikacija. Ako smo instalirali Ruby u CMD možemo samo upisati *gem install sass*. Poželjno je provjeriti instalaciju upitom *sass -v* (izbacuje verziju SASS-a).

3.1. Organizacija datoteka

Radi brzine učitavanja stranice poželjno je imati jednu CSS datoteku sa svim oblikovanjima na jednom mjestu umjesto u više razdijeljenih datoteka sa dijelovima stranice. Štoviše, poželjno je da ta jedna datoteka bude minimizirana radi manje veličine datoteke (to smo postigli kasnije sa Gulpom). To nam, naravno, ne ide u prilog jer, radi snalaženja, želimo imati jasno raspoređene dijelove stranica u zasebnim datotekama (npr. kod za izbornike u jednoj, za zaglavlja u drugoj te za podnožja u trećoj datoteci). SASS nam to omogućava opcijom *@import*. Naše zasebne datoteke ćemo staviti u podmapu *common* a u glavnoj datoteci *style.scss* ćemo uvesti te datoteke naredbom *@import*. To nam i olakšava posao jer sad moramo kompilirati samo jednu glavnu datoteku umjesto više manjih. Preporučeno je da imena poddatoteke započinju znakom '_' (npr. _header.scss).

Primjer:

style.scss

```
...
@import 'common/header';
@import 'common/footer';
@import 'common/nav';
...
```

3.2. Varijable

U SASS-u postoje varijable čije vrijednosti možemo mijenjati, no najčešća upotreba je spremanje jedne vrijednosti u jednu varijablu te korištenje te vrijednosti više puta u našem kodu (nešto za što je dovoljna i konstanta). Jedan od problema klasičnog CSS-a je nepostojanje varijabli. Ukoliko smo željeli promijeniti tu vrijednost, morali smo potražiti svaki zapis te vrijednosti te je promijeniti. SASS uvodi mogućnost varijabli u naš kod što znatno olakšava naš posao. Kao rezultat toga napravili smo posebnu datoteku _variables.scss u kojoj su popisane sve varijable.

Primjer:

_variables.scss

```
$body_background:#FFF;
$text_color:#444;
$body_font: 'Open Sans', sans-serif;
```

Te varijable smo iskoristili u kodu:

```
body {
background:$body_background;
color:$text_color;
font-family:$body_font;
}
```

3.3. Gniježđenje

Kad pišemo HTML, možemo primjetiti da on ima jasno ugniježđenu vizualnu hijerarhiju:

```
<div id=“div“>
     <ul>
          <li>
               <a>
                    Neki link
               </a>
          </li>
     </ul>
</div>
```

CSS, s druge strane, nema:

```
#div {color:yellow;}
#div ul {color:blue;}
#div ul li {color:red;}
#div ul li a {color:green;}
#div ul li a:hover {color:black;}
```

SASS nudi novi, intuitivniji način dohvaćanja elemenata koje želimo stilizirati.

Primjer:

```
#div {
color:yellow;
     ul {
     color:blue;
          li {
          color:red;
               a {
               color:green;

               &:hover {
               color:black;
               }
               }
          }
     }

}
```

Ovaj način nas više asocira na klasičnu HTML sintaksu pa je stoga lakše čitati i razumijeti ovu sintaksu.

3.4. Mixin-ovi

Mixin-ovi su najkoristnija opcija SASS-a. Oni djeluju na način sličan argumentima u funkcijama. Omogućavaju nam da često korištene opcije grupiramo te da ih koristimo više puta (moguće je korištenje argumenata kao kod funkcija).

Primjer:

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

.box { @include border-radius(10px); }
```

Ovaj kod će dodati zaobljenje na rubovima svim elementima koji na sebi imaju klasu *box*.

3.5. Bourbon i Neat

Bootstrap [18] je *framework* u kojem se nalazi mnoštvo predefiniranih, često korištenih CSS klasa [19] što se pokazalo poprilično korisnim. Međutim, glavni nedostatak Bootstrap-a je taj što se učitaju sve klase (nepotrebno velika datoteka) a ne samo one koje smo koristili. Bourbon [20] je isto takav *framework*, samo što je temeljen na SASS-u. Njegova glavna prednost je ta što nam omogućava da se kompilira samo ono što smo iskoristili od njega. *Neat* [21] je grid sistem (mreža koja se sastoji od 12 stupaca, s njom definiramo duljinu elementa, odnosno koji element na kojoj veličini ekrana zauzima koliko stupaca) baziran na Bourbonu. Skinuli smo ih i dodali u web stranicu korištenjem package manager-a Bower [22] te ih ubacili u style.scss @import opcijom. Najčešće smo koristili *Bourbon*-ov *clearfix* [23] *mixin* (za čišćenje float-ova) te Neat-ovu funkciju new-breakpoint za responzivni dizajn.

Primjer:

```
//media
$tablet:new-breakpoint(min-width 768px);
$medium:new-breakpoint(min-width 992px);
$large:new-breakpoint(min-width 1200px);

@include media($medium)
{
  label[for="menuToggle"] {display:none;}
}
```

U ovom primjeru želimo dohvatiti ikonu menija. Želimo da se ona javlja samo na manjim veličinama ekrana. *@include media* provjerava veličinu ekrana. Ukoliko je veličina ekrana veća od ili jednaka 992 piksela, postavit će vrijednost svojstva *display* na *none* (maknut će vidljivost ikone).

Na taj način možemo dohvaćati bilo koji element te mu mijenjati bilo koje svojstvo na željenoj veličini ekrana. Druga najčešća upotreba je upravljanje duljinom nekog elementa (primjer: <*div*> element nejčešće na veličini ekrana za mobilne uređaje zauzima punu duljinu ekrana, dok na većim duljinama ekrana obično zauzima manje, ovisno o dizajnu stranice).

SASS ćemo koristiti prilikom izrade pojedinih elemenata koji će se pozivati uz pomoć ACF dodatka.

4. Gulp

Gulp [24] je alat koji služi za automatiziranje repetitivnih zadataka koje bi inače web developeri morali obavljati samostalno. Ovo su neki od zadataka koje Gulp rješava:

- Spajanje i minimiziranje datoteka istog tipa (CSS, JavaScript)
- Automatsko osvježavanje preglednika čim spremimo promjene u datoteci
- Kompiliranje SASS-a i automatsko dodavanje prefix-a na pojedina CSS svojstva (primjer: upišemo transition a Gulp sastavi –webkit-transition)

4.1. Korištenje Gulpa

Da bismo koristili Gulp, moramo imati instaliran *Node.js* [25]. Uz *Node.js* dobijemo koristan package manager pod imenom NPM. S njim možemo instalirati nove pakete upisivanjem naredbe *npm install ime_paketa* u CMD.Ukoliko dodamo *--save--dev* NPM će spremiti ime paketa i njegovu verziju u datoteku package.json u kojoj ćemo imati popis svih korištenih paketa. No, ukoliko želimo instalirati paket „globalno“ na našem računalu dodat ćemo *–g*. Gulp ćemo za početak instalirati globalno.

```
npm install -g gulp
```

Općenito, prilikom instalacije npm paketa, u CMD se ispiše puno teksta, no možemo biti sigurni da je instalacija uspješna ako na kraju dobijemo ovakav, organiziran popis:

```
...
    +-- graceful-fs@3.0.11
    | `-- natives@1.1.0
    +-- mkdirp@0.5.1
    | `-- minimist@0.0.8
    +-- strip-bom@1.0.0
    | +-- first-chunk-stream@1.0.0
    | `-- is-utf8@0.2.1
    +-- through2@0.6.5
    | `-- readable-stream@1.0.34
    |   `-- isarray@0.0.1
    `-- vinyl@0.4.6
      `-- clone@0.2.0
```

Nakon toga moramo ga instalirati u naš projekt. Prvo upišemo *npm init* te upišemo željene podatke o našoj verziji. U izvornom direktoriju našeg projekta upišemo *npm install --save-dev gulp*. Primijetit ćemo da nam se u izvornoj datoteci stvorila mapa node_modules. U njoj je će se spremati datoteke paketa koje smo odabrali. Ta datoteka može postati poprilično velika stoga ju je poželjno spomenuti u .gitignore. Da bi netko koristio našu verziju Gulp skupine paketa, potreban mu je gulpfile.js u kojem su definirani načini na koji obrađujemo naše datoteke i

package.json u kojem se nalazi popis paketa koji koristimo. Taj korisnik samo mora upisati npm install i to će mu napraviti node_modules mapu sa svim paketima spomenutim u package.json i korištenim u gulpfile.js. Svaki paket koji odlučimo koristiti instalirat ćemo s naznakom *--save-dev* što će dodati zapis o njemu u package.json.

4.2. Gulpfile.js

Sad moramo stvoriti našu glavnu datoteku *gulpfile.js*. Prvo ćemo u tu datoteku uključiti pakete koje koristimo kasnije u definiranim zadacima. Pošto smo instalirali Gulp, počet ćemo s njim. Za početak u gulpfile.js upišemo sljedeće:

```
var gulp = require('gulp');
```

Varijablu gulp ćemo koristiti kao osnovu za naše automatizirane zadatke.

4.3. Uglify

Kao primjer klasičnog zadatka spomenut ćemo *uglify*. Uglify je dodatni paket *gulp-a* koji minimizira određene datoteke (u ovom slučaju .js datoteke). Da bismo instalirali Uglify u CMD upišemo npm install --save-dev gulp-uglify. Zatim ga dodamo kao varijablu u gulpfile.js (*var uglify = require('gulp-uglify');*). Zatim napravimo novi zadatak koji ćemo nazvati uglify.

```
// UGLIFY JAVASCRIPT
gulp.task('uglify',function(){
     gulp.src('js/common/**/*.js')
     .pipe(uglify())
     .pipe(gulp.dest('js'));
});
```

Unutar *gulp.src* kao parametar upišemo željenu putanju. Nakon toga pomoću *.pipe* naredbe priključujemo dodatke koje želimo da se izvše. Naredba *uglify* će minimizirati sve što odgovara uvjetu u *.src* (u ovom slučaju, sve *.js* datoteke koje se nalaze u direktoriju *js/common*)

Dakle, napravili smo novi zadatak pod imenom uglify. U primjeru piše da želimo da Gulp uzme sve .js datoteke koje se nalaze u js/common mapi te da ih minimizira i spremi u mapu js. U ovom zadatku poželjno je koristiti drugi paket pod imenom gulp-concat koji spaja sve te datoteke u jednu te ga ubaciti u uglify zadatak. Taj zadatak će se izvršiti ako u naš CMD upišemo gulp uglify.

Međutim, to nije ono što želimo. Mi želimo da se taj zadatak izvrši na početku (kada počinjemo raditi na našem projektu) te svaki put kad napravimo neku promjenu na našim

datotekama. Stoga u Gulpu postoji opcija watch koja gleda na promjene u određenoj datoteci i izvršava definirane zadatke kad se promjene dogode.

Stoga ćemo napraviti novi zadatak pod imenom watch i u njemu definirati navedeno.

```
gulp.task('watch', function() {
  gulp.watch('js/common/**/*.js', ['uglify']);
  gulp.watch(sass/**/*.scss, ['sass']);
  gulp.watch("**/*.php", ['reload-html']);
});
```

Moramo napomenuti da smo, osim uglify-a, definirali još dva zadatka koja će se izvršiti kad se pronađe podudaranje (sass za kompiliranje SASS-a i reload-html za osvježavanje preglednika). Dakle, watch funkcija prihvaća dva argumenta: prvi je izvor, odnosno putanja (isto kao i kod *.src*) a drugi je zadatak ili zadaci koji se pokreću na promjenu u izvoru (izraženi u obliku polja).

Da bismo na kraju sve te zadatke automatizirali stavljamo ih u jedan predefinirani zadatak pod imenom default.

```
gulp.task('default', ['sass','browser-sync', 'watch']);
```

Ovaime smo naveli da čim upišemo gulp u CMD, Gulp će kompilirati SASS, osvježiti preglednik i dalje aktivno gledati na promjene u našem projektu. U našem projektu koristit ćemo još nekoliko dodataka poput *css-nano* za minimiziranje CSS-a, *sourcemaps* za prikazivanje izvorne .scss datoteke u Chrome inspektoru umjesto kompilirane .css datoteke itd. (popis u poglavlju 7.4). Sada već imamo nekakvu osnovu oju ćemo još više optimizirati kasnije za naš projekt.

5. Modernizr

Modernizr[6] je alat s kojim provjeravamo mogućnosti preglednika. U svojoj osnovi je to jedna JavaScript datoteka u kojoj se nalaze funkcije koje smo mi odabrali i koje testiraju podržava li pojedini preglednik te mogućnosti. Rezultate testiranja prikazuje na način da u naš <html> tag dodaje određene klase.

Primjer:

```
<html class="no-cssgradients no-boxshadow">
```

Ovaj primjer provjerava podržava li preglednik gradijente i sjene. Kao primjer za CSS3 odabrat ćemo klasičan atribut sjene (*box-shadow*: dodavanje sjene na element, kao atribute prima udaljenost po X i Y osi od elementa, širinu sjene te boju sjene). Ukoliko smo odlučili da Modernizr testira preglednik za box-shadow atribut te on zaključi da preglednik podržava *box-shadow*, u <*html*> tag dodat će klasu *boxshadow*. U suprotnom dodat će klasu *no-boxshadow* koja će samo podebljati i obojati rubove elementa. Na taj način možemo pregledniku reći što učiniti ako ne podržava tu mogućnost.

Primjer:

```
.klasa {
box-shadow:0 0 5px #000;
}
.no-boxshadow .klasa {
border:5px solid #000;
}
```

Postoji još jedan način s kojim možemo napraviti *fallback* a to je putem Javascript-a. Jedini SVG koji koristimo u našoj temi je naš logo i on se prikazuje pomoću <img> taga. Modernizr ima test funkciju baš za takve slučajeve pod imenom *svgasimg*. Postavili smo uvjet koji govori da, ukoliko preglednik ne podržava .svg datoteke, da ih zamijeni .png datotekama s istim imenom. Da bi ovo radilo moramo, osim loga u .svg formatu, imati i logo u .png formatu sa istim imenom.

Primjer:

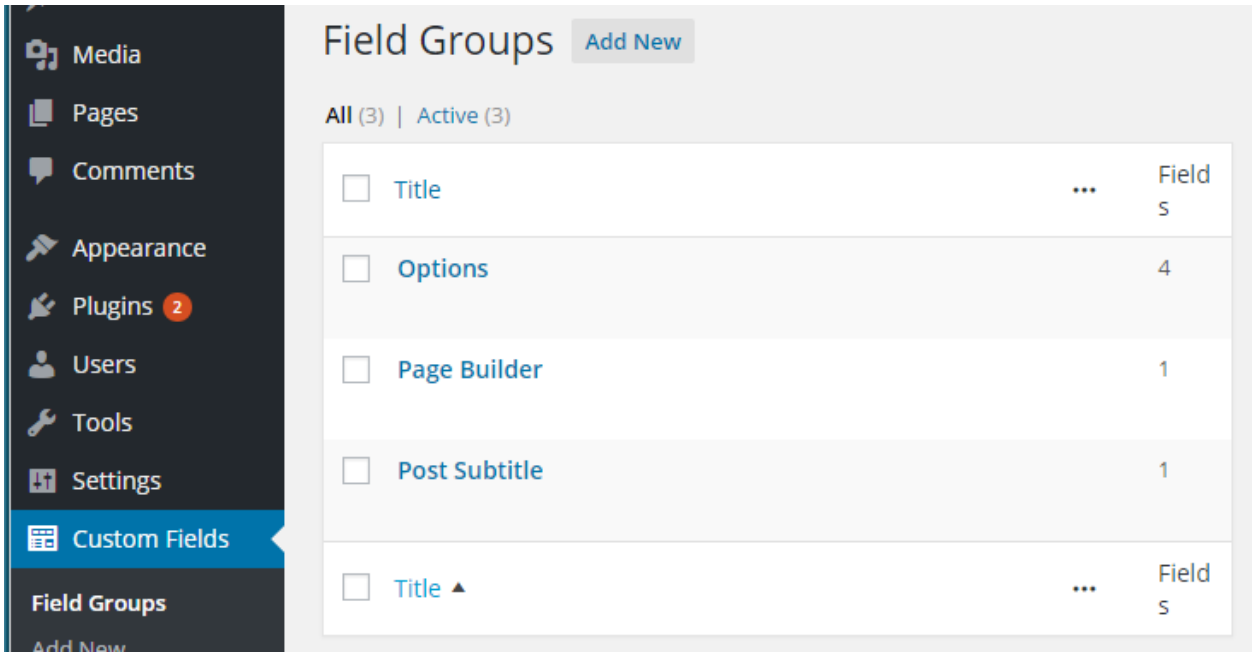
```
if(!Modernizr.svgasimg) {
    $('img[src*="svg"]').attr('src', function () {
        return $(this).attr('src').replace('.svg', '.png');
    });
}
```

Vrijednost *Modernizr.svgasimg* imat će vrijednost *false* ukoliko preglednik ne podržava .svg datoteke. U tom slučaju uzet ćemo sve slike čiji *src* atribut završava sa *.svg* te će zamijeniti vrijednost *.svg* sa *.png*. Upravo ćemo tu skriptu koristiti i u našem projektu.

6. Wordpress dodatak ACF (Advanced Custom Fields)

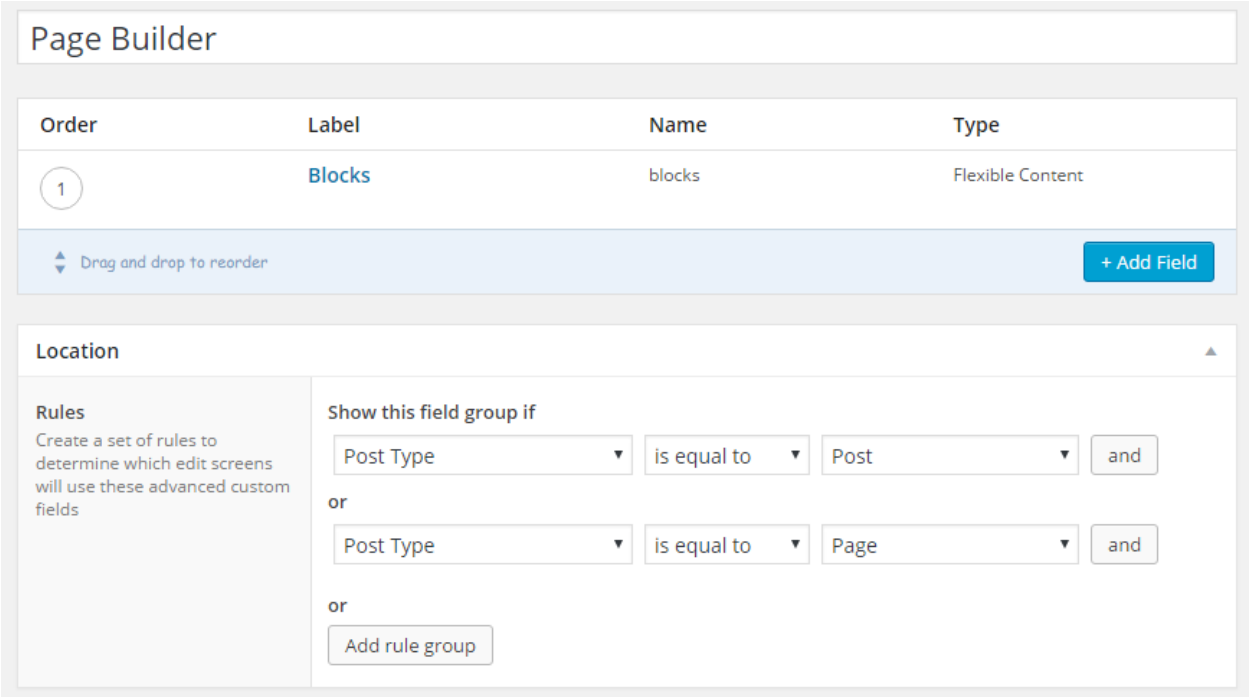
ACF[10] dodatak za Wordpress razvijen od strane Elliota Condona te je ujedno i jedan od najpopularnijih Wordpressovih plugin-ova. Ukratko rečeno, ACF plugin nam omogućava da u adminovom korisničkom sučelju dodajemo nove elemente na stranicu ili post poput klizača, news feed-a ili općenito bilo kojeg custom elementa kojeg smo definirali.

Kada jednom aktiviramo ACF plugin u Wordpress administratorskom sučelju javit će se nova stavka izbornika pod imenom Custom Fields. U njoj definiramo različite skupine custom polja. Za svaku skupinu potrebno je postaviti uvjet pod kojim će nam se ta skupina prikazati. Npr. napravili smo skupinu pod imenom Post Subtitle koja se sastoji samo od jednog tekstualnog polja i rekli smo da se ta skupina prikazuje samo kada uređujemo post. Naša glavna skupina je Page Builder i koristit ćemo ju za dodavanje elemenata na stranice ili postove.



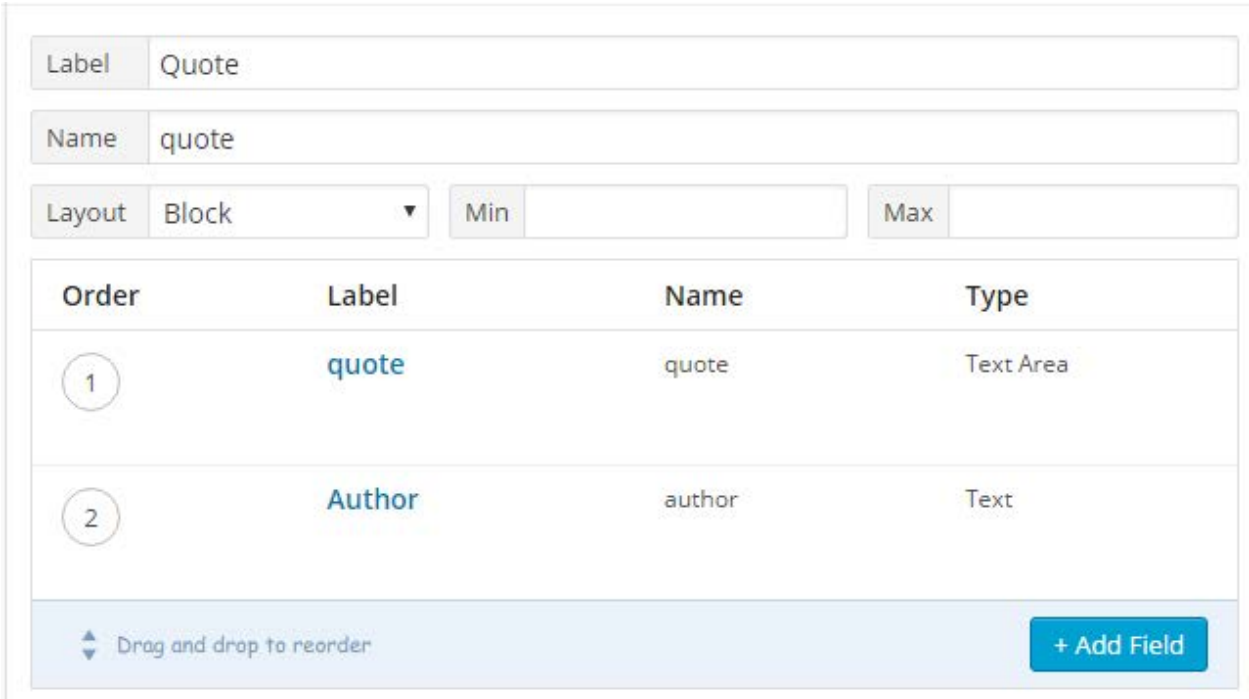
Slika 1. Custom Fields stavka u Wordpress-ovom sučelju

Kao primjer korištenja ACF-a dodat ćemo novi element (citat koji označavamo s <cite>). Za početak trebamo imati na umu što sve trebamo za dodavanje novog citata. Page builder skupina sastoji se od jednog elementa pod imenom Blocks tipa Flexible Content u kojem se nalaze svi definirani elementi.



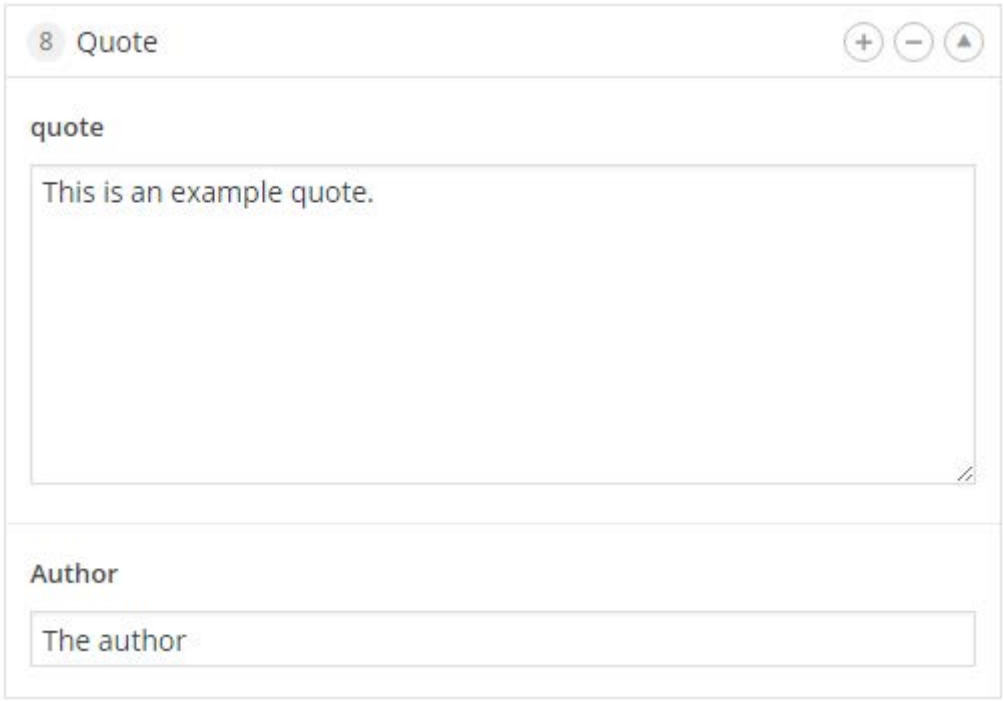
Slika 2. Page Builder grupa, Blocks podgrupa i uvjet prikaza grupe

Koristit ćemo jedan text area za sam citat te običan text field za autora. Stoga smo u *Blocks-u* dodali novi layout pod imenom Quote. U taj layout dodali smo text area pod imenom quote i text field pod imenom author.



Slika 3. Izgled Quote elementa unutar Blocks podgrupe

Kliknemo na Update te otvorimo neku stranicu za uređivanje. Primijetit ćemo da sad možemo dodat novi element Quote te ako ga kliknemo prikazat će nam se jedan text area za citat i jedan text field za autora. Možemo ih ispuniti ali moramo imati na umu da još nismo definirali kako će se ovaj custom sadržaj prikazati na našoj stranici.



Slika 4. Izgled Quote elementa u Pages stranici

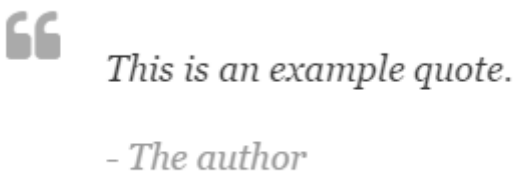
ACF ima definirane PHP funkcije s kojima možemo dohvatiti podatke koje smo definirali u adminovom korisničkom sučelju. Njih ćemo pisati unutar naše index.php datoteke. Funkcija *have_rows* gleda postoje li zapisi u bazi (postoje ukoliko smo dodali quote). Funkcija *get_row_layout* nam daje naziv bloka koji koristimo, te ovisno o nazivu, ispisat ćemo drugi HTML kod koji se nalazi u nekoj od datoteka unutar *page_builder* direktorija.

Način na koji ćemo prikazati te podatke je sljedeći:

```
<?php
   // START: have rows
   if( have_rows('blocks') ):
       while ( have_rows('blocks') ) : the_row();

         //PAGE HEADER
               if( get_row_layout() == 'page_header' ):
               include 'page_builder/page_header.php';

               // ...
```

```
            //GRID
            elseif( get_row_layout() == 'grid' ):
            include 'page_builder/grid.php';

            // Quote
            elseif(get_row_layout()== 'quote'):
            include 'page_builder/quote.php';

            //Slider
            elseif(get_row_layout() == 'slider'):
            include 'page_builder/slider.php';

            //  ...


            //endif (get_row_layout)
          endif;

    // END: have rows loop
    endwhile;

    else:?>

 <div class="container">
 <h1><?php the_title(); ?></h1>
  <?php the_content(); ?>
 </div>


<?php //END: if have rows
endif; ?>
```

Početni uvjet provjerava postoje li konkretni zapisi unutar bloka. Ako ne, generirat će se samo *div* klase *container*, s naslovom stranice i klasičnim sadržajem (ukoliko ga ima). No, ukoliko smo stavili nove zapise, ulazimo u petlju koja prolazi kroz njih. Za svaki zapis provjerava se njegov tip te ovisno o tipu ubacuje određeni kod iz odvojenih datoteka. Ovo je sadržaj datoteke *quote.php* koja će se ispisati ako smo dodali novi element tipa Quote i ispunili prije spomenuti *text area* i *text field*.



Slika 5. Izgled Quote elementa u temi

```
<div class="quote_wrapper">
    <blockquote class="quote">
        <i class="icon-quote-left"></i>
        <?php the_sub_field('quote');?>
        <cite class="cite">- <?php the_sub_field('author');?></cite>
    </blockquote>
</div>
```

Funkcija *the_sub_field*(); prima jedan parametar i to je upravo ime koje smo dali tom elementu u adminovom korisničkom sučelju. Ona vraća vrijednost upravo tog elementa (na primjeru na slici 4 ta vrijednost bi bila „This is an example quote.“).

Moramo još spomenuti posebnu skupinu custom fields-a koja se zove Options[26]. ACF nam omogućava da napravimo posebnu skupinu u kojoj možemo postaviti opcije stranice. Kao opcije ove teme stavili smo ikone koje će se pojaviti u header-u i footer-u(njihov prikaz definirat ćemo kasnije u datotekama header.php i footer.php) linkove na koje vode te ime i opis tvrtke ili organizacije koji piše u footer-u. Options skupinu potrebno je prije registrirati u *functions.php* datoteci kao posebnu skupinu. Princip rada options skupine je sličan kao i kod klasičnih ACF elemenata. Provjeravamo postoji li vrijednost u bazi za taj element, te, ako postoji, generiramo prikladan HTML. Jedina razlika je što u *the_field* funkciju moramo dodati *options* kao drugi parametar.

```
<?php if(get_field('footer_company_name', 'options') ): ?>
       <div class="company_name">
           <?php the_field('footer_company_name', 'options'); ?>
       </div>
 <?php endif; ?>

 <?php if(get_field('footer_address_descr', 'options') ): ?>
       <div class="address">
           <?php the_field('footer_address_descr', 'options'); ?>
       </div>
   <?php endif; ?>
```

Korištenjem ACF-a napokon dobivamo podatke iz baze te ih prikazujemo na željenim mjestima. Sve što nam ostaje jest da dodamo nove elemente u ACF dodatak poput klizača, naslova, CTA linka itd. te da napišemo odgovarajući HTML kod s prikladnim CSS klasama za svaki od njih.

7. Izrada teme

7.1. Razvojno okružje

Na početku smo definirali naše razvojno okruženje pomoću PuPHPet-a[14]. PuPHPet je alat s kojim možemo podesiti naš virtualni stroj za web razvoj. Neke od bitnijih stavki koje možemo podešavati su: tip servera, tip baze podataka, programski jezici u kojima želimo pisati kod itd. Na kraju samo skinemo .rar datoteku u kojoj se nalazi naša vagrant konfiguracija. Sadržaj datoteke raspakiramo u direktorij gdje želimo da bude naš projekt te u CMD-u odemo do tog direktorija i napravimo inicijalni *vagrant up*. Kad smo prvi put upisali *vagrant up*, Vagrant mora skinuti cijeli box, što obično potraje 20-tak minuta. Jednom kad završi, naše razvojno okružje je spremno za rad.

7.2. Wordpress

U PuPHPet-u smo definirali da će naši projekti biti spremljeni unutar direktorija wordpress. Stoga u njemu napravimo novi direktorij koji se u našem slučaju zove zavrsni_rad. Pošto radimo klasičnu Wordpress temu, naš sljedeći zadatak je skinuti posljednju verziju Wordpress-a sa njihove stranice (www.wordpress.org). Konkretno, skinut ćemo arhivu koja u sebi ima wordpress direktorij u kojem se nalazi novi, prazan Wordpress projekt. Sadržaj direktorija wordpress izvezemo u zavrsni_rad. Sljedeći korak je kreiranje baze podataka te spajanje našeg projekta s bazom. Za početak otišli smo u Adminer (slično kao i PHPMyAdmin, definirali smo ga u PuPHPet-u) te tamo kreirali novu bazu wp_mayhem (Mayhem će nam biti ime teme). Nakon što smo to učinili vrijeme je da po prvi put pokrenemo naš projekt. U naš preglednik upišemo wordpress.ia/zavrsni_rad (wordpress.ia smo definirali kao lokalnu domenu u PuPHPet-u a direktorij wordpress u kojem se nalazi zavrsni_rad je tzv. root direktorij). Pošto smo prvi put pokrenuli naš projekt, Wordpress je od nas zatražio da definiramo neke osnovne podatke kao npr. ime našeg site-a, ime baze, admina i lozinku. Jednom kad smo ih upisali, imamo spreman Wordpress projekt.

7.3. Izrada Wordpress teme

Wordpress tema nam daje dizajn stranice. U njoj je definirano gdje će se i kako prikazivati podaci koje dobivamo iz baze. Sve teme možemo pronaći u *wp-content/themes* direktoriju.

Wordpress ima neke svoje teme nazvane po godinama kad su napravljene (twentyfourteen/fifteen/sixteen). Te teme možemo slobodno obrisati te napraviti novu mapu pod imenom Mayhem. U novoj mapi napravit ćemo naše prve datoteke: index.php i style.css. U style.css ćemo samo u komentar upisati sljedeće:

```
/* Theme Name: Mayhem */
```

Na taj način smo dali ime temi. Osim Theme Name možemo dodati i stavke poput: Author, Description, Tags itd. Style.css datoteku više nećemo dirati, tj. ovu smo napravili samo zato jer Wordpress traži tu datoteku da izvuče podatke o temi (baš kao Theme Name). U našem Wordpress sučelju odemo na Appearance > Themes te tamo odaberemo našu temu. Sada smo spremni početi raditi na strukturi i dizajnu naše teme.

7.4. Prvi koraci

Prvo smo kreirali direktorije s kojima ćemo organizirati naš projekt. U mapi sass ćemo imati sve naše .scss datoteke. Glavna datoteka bit će nam style.scss a sve ostale koje ćemo importati u style.scss nalazit će se u common direktoriju. Isto tako ćemo učiniti i za .js datoteke. Napravit ćemo i css direktorij u kojeg ćemo posredstvom Gulp-a kompilirati naš SASS. Mapa icons sadržavat će ikone koje smo odabrali i skinuli s Icomoon-a, web aplikacije s kojom možemo birati između velikog broja ikona, te generirati našu vlastitu datoteku s fontovima. Mapa *images* sadržavat će slike.

Sad moramo konfigurirati *Gulp* da automatiziramo naš proces. U gulpfile-u smo definirali sljedeće zadatke: reload-html (na poziv ponovno učita našu stranicu), uglify(spaja jquery datoteku importanu s bowerom i sve datoteke koje se nalaze u js/common mapi u jednu te ih minificira), sass (kompilira SASS, dodaje *prefix-e* od preglednika, minificira css, prati izvorne .scss datoteke u inspektoru i ponovno učita stranicu), browser-sync (sinkronizira stranicu s preglednikom) i watch(poseban zadatak koji prati zadane direktorije te vrši određeni zadatak kad se dogodi promjena u nekom od njih). Na kraju tu je najbitniji zadatak default koji se izvrši kad upišemo gulp u naš cmd. On će odmah na početku izvršiti spomenute zadatke te gledati na daljnje promjene. Datoteka functions.php sadržavat će funkcionalnosti vezane uz Wordpress (registracija glavnog menija, definiranje ACF *options* stranice i naslovne slike za postove).

```
gulp.task('default', ['sass','uglify','browser-sync', 'reload-html',
'watch']);
```

Moramo napomenuti da smo uz sve to napravili Git [27] repozitorij i izvršili inicijalni commit. Isto tako, instalirali smo ACF dodatak [10] u naš Wordpress projekt kako bismo ga mogli koristiti. Git repozitorij omogućit će nam veću kontrolu nad verzijama našeg projekta, spremanje trenutne verzije te vraćanje na prijašnje ukoliko pogriješimo u nekom trenutku.

7.5. Organizacija stranice

Pošto se header i footer ponavljaju kroz cijeli site, poželjno ih je staviti u posebne datoteke. Wordpress je konfiguriran da potraži datoteke s imenom header.php i footer.php te da na njih gleda na taj način. Za početak ćemo kreirati te datoteke te u naš index.php na početku upisati sljedeće:

```
<?php get_header(); ?>
```

A na kraju:

```
<?php get_footer(); ?>
```

Te dvije funkcije pozivaju neš header i footer u index.php. Podsjećamo da ćemo koristiti ACF plugin da napunimo stranicu sa sadržajem tako da će prostor između te dvije funkcije za sad ostati prazan. I u header-php i u footer.php pozivat ćemo ACF-ove funkcije (options) i standardne Wordpress-ove funkcije.

U header.php upisat ćemo početak našeg dokumenta. Struktura će za sad izgledati ovako:

```
<!DOCTYPE HTML>
<html>
<head>
   <title><?php bloginfo('name'); ?></title>
     <!--<meta> tagovi-->
     <!--primjer:-->
 <meta name="viewport" content="width = device-width, initial-scale =
1.0">
     <!--<link> tagovi-->
     <!--primjer:-->
  <link rel="stylesheet" href="<?php bloginfo('template_directory');
?>/css/min/style.css">
</head>
<body <?php body_class(); ?>>
```

Funkcija *bloginfo* nam vraća postavke koje mi definiramo u adminovom sučelju. Bloginfo('name'); nam vraća naslov stranice a bloginfo('template_directory'); nam vraća adresu izvorne datoteke teme koje smo odabrali u sučelju. Funkcija *body_class*(); dodaje klase body tagu poput imena teme, id-a stranice ili posta na kojem se nalazimo. Koristimo ju dalje u stiliziranju stranice s CSS-om.

S druge strane, footer.php izgledat će ovako:

```
<footer>
<!--kod za footer-->
</footer>

 <script
src="<?php bloginfo('template_directory');?>/js/scripts.min.js">
</script>

<?php wp_footer(); ?>
</body>
</html>
```

Funkcija wp_footer(); dodaje se prije završetka *body* oznake i dodaje Wordpress-ov kod potreban za funkcioniranje stranice. Mnogi dodaci generalno koriste ovu funkciju za referenciranje njihovih JavaScript datoteka.

7.6. JavaScript

Naša tema imat će relativno malo JavaScript-a u sebi. Uvest ćemo skriptu za Owl Carousel [28] (klizač) koja se osim jedne .js datoteke za koju je potreban jQuery sastoji i od dvije css datoteke (koje smo konvertirali u scss i importali u style.scss). Glavnom divu koji će sebi sadržavati stavke klizača dodat ćemo klasu .slider te ćemo s jQuery-em inicijalizirati klizač i postaviti sljedeće postavke:

```
$(document).ready(function() {
    $(".slider").owlCarousel({
        singleItem:true, //slider prikazuje samo jednu stavku
        pagination:true, // vidljiva paginacija
        slideSpeed : 1000, // brzina mijenjanja slajdeva
        autoPlay:3000, // automatska izmjena slajdeva svake 3 sekunde
        stopOnHover: true //zaustavljanje autoPlay-a na hover
    });
});
```

Owl Carousel ima i brojne druge postavke. [29]

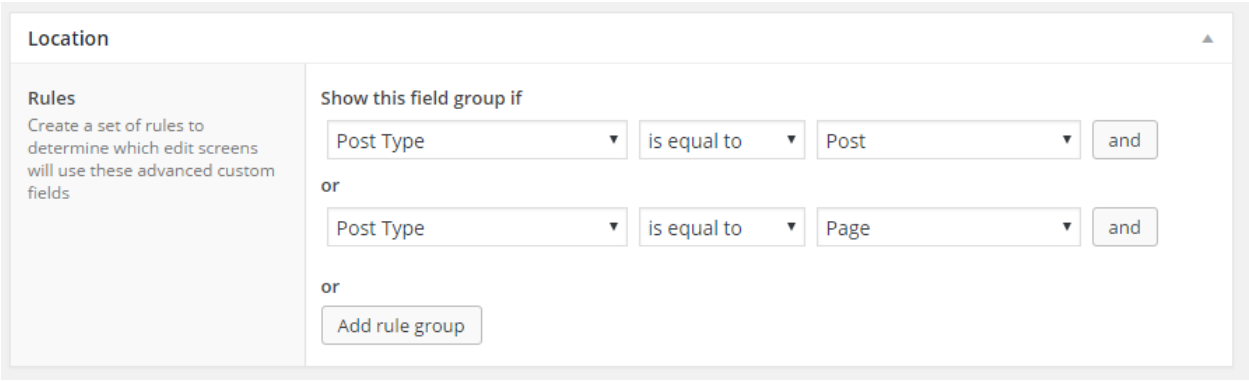
Osim skripte za slider koristit ćemo i prije spomenuti Modernizr, te njegovu mogućnost kreiranja fallback-a za SVG-ove kao što je opisano u poglavlju 5.

7.7. Index.php - Definiranje ACF komponenti

Kao što smo već spomenuli u poglavlju 6, ACF plugin nam omogućava da u adminovom sučelju možemo samo dodavati komponente koje želimo te na taj način dodavati sadržaj na našu stranicu. Naš zadatak je konfigurirati stranicu tako da se učita lista komponenti i njihovih

odabranih postavki (npr. kod citata, tekst i autor) te da se generira odgovarajući HTML kod. Moramo napomenuti da HTML kod koji će se generirati mora u sebi sadržavati odgovarajuće klase i identifikatore jer ćemo uz pomoću njih stilizirati pojedinu komponentu. Komponente koje ćemo definirati u temi su sljedeće: *Heading* (veliki naslov), CTA (*Call To Action*), *Grid* (responzivna mreža), *Quote* (citat), *WYSIWYG Field* (Skraćeno od *What You See Is What You Get* - standardno polje za unos s editorom), Klizač, *Carousel* (customizirani klizač), *News Feed* (popis zadnjih postova) i *Page Header* (slika s naslovom).

Za početak odemo u *Custom Fields* u adminovom sučelju i dodamo novu grupu polja koju ćemo nazvati *Page Builder* (Slika 2). Sve naše komponente spremit ćemo u jednu veliku komponentu tipa *Flexible Content* koja će se zvati *Blocks* (također na slici 2). *Page Builder* grupu polja moći ćemo koristiti i za stranice i za postove. Stoga ćemo ispod postaviti uvjet da se ova grupa polja prikazuje pod sljedećim uvjetima:



Slika 6. Uvjeti pod kojim se prikazuje grupa Page Builder

Grupa *Page Builder* prikazat će i na stranicama i na postovima.

7.8. Izrada klizača

Kao primjer komponente odabrat ćemo klizač (*slider*). Naš klizač mora imati opciju da se dodaju novi slajdovi te ćemo za svaki od tih slajdova morati definirati sljedeće stavke: pozadinsku sliku, naslov, opis, tekst CTA linka ispod opisa, adresu linka na koji on vodi i background position opciju za namještanje pozadinske slike. U naš *Blocks* flexibilni sadržaj dodat ćemo novi *layout* i nazvati ga *Slider*. Treba nam opcija da dodajemo nove slajdove. Stoga ćemo u naš *Slider layout* dodati novu komponentu koja će se zvati *Slide*, a njezin tip bit će *Repeater*. *Repeater* je tip komponente koji se može ponavljati, tj. omogućava nam da dodajemo više istih komponenti. Sad napokon možemo dodati glavne stavke pojedinog slajda. Trebamo obratiti pozornost na imena pojedinih komponenti jer ćemo ih koristiti u kodu. Sad napokon možemo otići u *Pages* sekciju u adminovom sučelju, dodati novu stranicu te dodati naš klizač u

nju. U sljedećem koraku ćemo definirati kako ćemo informacije dobivene iz adminovog sučelja prenijeti u HTML kod.

Između zaglavlja i podnožja nalazit će se naš sadržaj definiran ACF-om. Pošto smo u adminovom sučelju dodali novi klizač, vrijeme je da ga i prikažemo. Naš Fleksibilni sadržaj nazvali smo *Blocks* te mu možemo pristupiti u kodu pod imenom *blocks*. Napravit ćemo petlju koja će proći kroz sadržaj koji smo dodali na stranicu.

```
<?php
 // START: have rows
 if( have_rows('blocks') ):
     while ( have_rows('blocks') ) : the_row();

     // END: have rows loop
     endwhile;

   //END: if have rows
 endif; ?>
```

Unutar petlje ubacit ćemo uvjete koji gledaju tip komponente, te ako je odgovarajući tip izvršiti određeni kod. Naš kod sada izgleda ovako:

```
<?php
 // START: have rows
 if( have_rows('blocks') ):
     while ( have_rows('blocks') ) : the_row();

          //Slider
             if(get_row_layout() == 'slider'):
             include 'page_builder/slider.php';

     // END: have rows loop
     endwhile;

   //END: if have rows
 endif; ?>
```

Da ponovimo: početni uvjet provjerava ima li elemenata u skupu *blocks*. Ukoliko ima, petlja prolazi kroz svaki od njih. U tijelu petlje provjerava se tip elementa. U ovom slučaju imamo samo klizač (*slider*). Ukoliko smo ga dodali u bazu preko Wordpressovog sučelja, ispisat će HTML kod koji se nalazi unutar datoteke *slider.php* u direktoriju *page_builder*.

Sad je vrijeme da kreiramo novu mapu koja će se zvati *page_builder* i unutar nje datoteku *slider.php*. Datoteka *slider.php* će u sebi sadržavati petlju koja će prolaziti kroz svaki slajd i uzimati stavke svakog slajda. Stavkama pristupamo funkcijom *get_sub_field('ime_stavke').*

```
<?php if(have_rows('slide')):?>
<div class="slider">
    <?php
    while(have_rows('slide')): the_row();?>
        <?php $attachment_id = get_sub_field('image');
        $bgImageSize = "thumb_1920x1080";
        $bgImage=wp_get_attachment_image_src($attachment_id,
$bgImageSize);
        ?>
        <div class="slider_item" style="background-image: url('<?php
echo $bgImage[0]; ?>');
        <?php if(get_sub_field('background_position')): ?>
            background-position:<?php
the_sub_field('background_position'); ?>;
        <?php endif; ?>)">



 <?php
        if(get_sub_field('title')|| get_sub_field('description')):
                ?>
                <div class="title">
                    <?php if(get_sub_field('title')): ?>
                        <h1><?php the_sub_field('title') ?></h1>
                    <?php endif; ?>
                    <?php if(get_sub_field('description')): ?>
                        <p><?php the_sub_field('description') ?></p>
                    <?php endif; ?>
                </div>
            <?php endif; ?>
            <?php if(get_sub_field('cta_label')): ?>
                <p class="slider_CTA">
                   <a href="<?php the_sub_field('cta_url'); ?>">
                     <?php the_sub_field('cta_label'); ?>
                   </a>
                </p>
            <?php endif; ?>

        </div>
    <?php endwhile; ?>
</div>
<?php endif; ?>
```

Naš kod za klizač prvo provjerava jesmo li uopće dodali koji slajd(*if (have_rows('slide'))*). Ako jesmo stvara div s klasom .*slider*. Prisjetimo se da smo u prije s jQuery-em rekli da svaki div koji ima klasu .slider pretvorimo u OwlCarousel. Zatim pokrećemo petlju koja ide kroz svaki slajd. Nadalje, dohvaćamo našu odabranu sliku te ju prilagođavamo za prikaz. Zatim stvaramo novi div s zadanom pozadinskom slikom. Ostatak koda provjeravamo jesmo li upisali nešto u određena polja, te ako jesmo ispisuje se prilagođeni HTML kod s prikladnim klasama. Jedino što nam ostaje je stilizirati naš klizač. SASS kod za naš klizač izgleda ovako:

```
// Slider
  .slider_item {
    height: 0;
    max-height: 50vh;
    min-height: 300px;
    text-align:center;
    position:relative;
    .title {
      width:80%;
      position:absolute;
      top: 50%;
      left: 50%;
      margin-right: -50%;
      transform: translate(-50%, -50%);
      h1 {
        color:$slider_h1_color;
        text-transform: uppercase;
        background:$slider_h1_p_background;
        font-weight:600;
        font-size:30px;
        margin:0;
      }
      p {
        background:$slider_h1_p_background;
        color:$slider_p_color;
        font-size:18px;
        padding:10px;
        margin:0;
        margin-top:10px;
      }
    }

    .slider_CTA {
      position:absolute;
      bottom:10px;
      text-align:center;
      width:100%;
      a {
        min-width:300px;
        display:inline-block;
        padding:10px 20px;
        text-decoration:none;
        font-size:20px;
        transition:300ms;
        text-transform: uppercase;
        color:$slider_CTA_color;
        background:$slider_CTA_background;
        &:hover {
          background:$slider_CTA_background_resp;
          color:$slider_CTA_color_resp;
        }}}}
```

Na isti način napravili smo i ostale komponente s iznimkom *News Feed-a* koji je uzeo zadnja četiri posta te je za svaki prikazivao minijaturnu verziju slike posta, naslov i podnaslov posta.

Podnaslov posta nije „po defaultu“ definiran u Wordpressu stoga ćemo ga naknadno dodati s ACF-om.

Prikaz postova:

```
<?php foreach( $posts as $post ): setup_postdata( $post ) ?>
         <article class="col-xs-12 col-md-3">
             <a href="<?php the_permalink(); ?>">
                <?php if ( has_post_thumbnail() ) {
                   the_post_thumbnail('small-thumbnail');
                 }?>
                <?php if(get_field('subtitle')):?>
                   <h4><?php the_field('subtitle');?></h4>
                <?php endif; ?>
             </a>
             <a href="<?php the_permalink(); ?>">
                <h3><?php the_title(); ?></h3>
             </a>
         </article>
<?php endforeach; ?>
```

7.9. Header.php

Za početak ćemo samo ubaciti naš logo.

```
 <a href="<?php bloginfo('wpurl'); ?>">
 <img src="<?php bloginfo('template_directory'); ?>/images/logo.svg“>
</a>
```

Funkcija *bloginfo('wpurl')* vratit će početnu adresu a *bloginfo('template_directory')* izvornu adresu odabrane teme. Podsjećamo, da smo Modernizr-om riješili *fallback* za SVG datoteke, pa ukoliko preglednik ne prepozna naš logo, zamijenit će ga PNG verzijom.

7.10. Meni

Naš *header.php* za sad u sebi sadrži samo osnovne elemente (*html, title, meta* i *body*). Pošto svaka naša stranica i post mora imati glavni meni u sebi, njega ćemo definirati u ovoj datoteci. Našim menijem možemo upravljati u adminovom sučelju. Za to je zaslužna funkcija *wp_nav_menu(array('theme_location' => 'ime_menija')* koja će iz baze izvući podatke koje mi konfiguriramo te napraviti klasičnu listu. Tu listu, naravno, možemo stilizirati po želji. Prije nego što možemo koristiti naš meni, moramo ga registrirati u *functions.php* pomoću funkcije *register_nav_menu*.

```
function registriraj_menu() {
  register_nav_menu('header-menu', 'Header menu');
}
add_action( 'init', 'registriraj_menu' );
```

7.11. Options

Naša tema će imati mogućnost da pokraj loga možemo postaviti ikone/linkove koji vode na druge stranice poput društvenih mreža i slično. ACF dodatak nam omogućuje da možemo dodati custom postavke za našu stranicu što je idealno za ovaj slučaj. Na početku ćemo registrirati našu *Options* stranicu u *functions.php* koja će se zvati Global koristeći ACF-ovu funkciju *register_options_page*.

```
register_options_page('Global');
```

U adminovom sučelju sada imamo novu stavku *Options*. ACF options[26] funkcionira jednako kao i klasične ACF stavke. U *Custom Fields* u adminovom sučelju napravimo novu grupu. Glavni element bit će tipa *Repeater* jer želimo imati mogućnost dodavanja više linkova. Elementi *repeater-a* bit će sljedeći: *dropdown* meni u kojem će se nalaziti popis ikona (svaka ikona imat će svoj naziv pomoću kojeg ćemo ispisati tu ikonu – *the_sub_field('icon')*) i tekstualni okvir u kojeg ćemo upisati link na koji ta ikona vodi. Kod za *Options* u *header.php-u* izgleda ovako:

```
  <?php while( have_rows('header_contact_icons','options') ):
the_row(); ?>

 <a href="<?php the_sub_field('url'); ?>" target="_blank">
 <i class="icon-<?php the_sub_field('icon'); ?>"></i>
</a>
 <?php endwhile; ?>
```

Imamo jednu petlju koja prolazi kroz stavke *Repeater-a* i za svaki ispisuje link s ikonom.

7.12. Footer.php

Za *footer.php* smo napravili još jednu takvu listu ikona i polje sa imenom tvrtke ili organizacije te njezinom adresom. Kod provjerava postoji li element *footer_company_name* u *Options-u* te, ukoliko postoji, kreira * * element sa klasom *company_name* i ispisuje vrijednos elementa unutar njega. Slična stvar događa se i sa linkom u *footeru*. Petlja prolazi kroz sve elemente te za svaki ispisuje *<a>* tag sa prikladnim linkom i ikonom.

```
 <?php if(get_field('footer_company_name', 'options') ): ?>
        <div class="company_name">
            <?php the_field('footer_company_name', 'options'); ?>
        </div>
  <?php endif; ?>
    <?php if(get_field('footer_address_descr', 'options') ): ?>
        <div class="address">
            <?php the_field('footer_address_descr', 'options'); ?>
        </div>
    <?php endif; ?>
  <?php if( have_rows('header_contact_icons', 'options') ): ?>
    <div class="footer_contact">
    <?php while( have_rows('footer_contact_icons', 'options') ):
the_row(); ?>
        <a href="<?php the_sub_field('url'); ?>" target="_blank">  <i
class="icon-<?php the_sub_field('icon'); ?>"></i></a>
    <?php endwhile; ?>
    </div>
```

7.13. Single.php

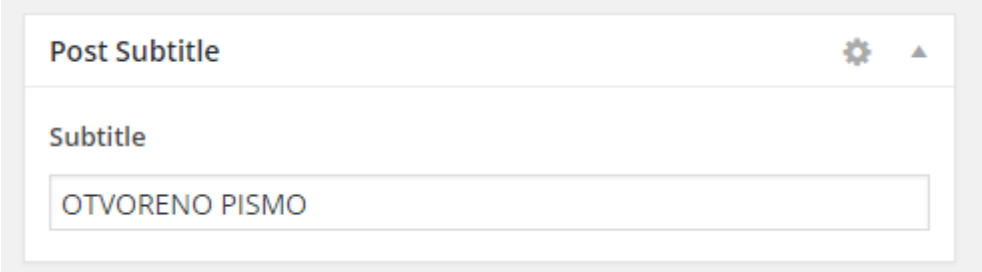
Kao što wordpress traži imena datoteke poput *header.php, index.php, footer.php* i slično, tako traži i datoteku single.php. U toj datoteci ćemo definirati kako će izgledati naš post. Postovi se razlikuju od stranica po tome što oni imaju vremensko značenje, tj. neka novost bi bila post dok bi općenite informacije poput "O nama" bile smještene u meni stranice..

Korisnik će moći koristiti ACF plugin i za pisanje postova, stoga će single.php imati sličnu strukturu poput index.php-a. Glavna razlika je u tome što, u našem slučaju, svaki post ima mogućnost prikazivanja glavne slike i podnaslova. Ukoliko smo definirali glavnu sliku i podnaslov za neki post, oni će se prikazivati na početku.

```
<div class="container">
    <div class="classic_wysiwyg">
        <?php if ( has_post_thumbnail() ) {
            the_post_thumbnail('post-featured');
        }  ?>
```

```
        <p class="post_date_time"><?php the_date();?>, <?php
the_time(); ?></p>
        <h1><?php the_title(); ?></h1>
        <h3><?php if(get_field('subtitle')): the_field('subtitle');
endif; ?></h3>

    </div></div>
```

Moramo napomenuti da smo s ACF-om napravili novu grupu koja se prikazuje samo na post-ovima. Ta grupa se sastoji samo od jednog tekstualnog polja u koji možemo upisati naš podnaslov. U kodu smo ispisali naš podnaslov funkcijom *the_field('subtitile').*



Slika 7. Subtitle tekstualno polje u Posts stranici

Slika 8. Subtitle tekstualno polje ispod naslova u temi

8. Zaključak

Kao što postoje mnogi alati u programiranju koji nam pomažu da napravimo konačan proizvod (poput Unity-a za videoigre) u kojima su predefinirane neke mogućnosti koje se općenito često koriste u toj grani (u slučaju s Unity-om i videoigrama možemo spomenuti koliziju objekata) tako postoje i alati koji nam pomažu u izradi web stranica, aplikacija, siteova i slično. U slučaju s web developmentom to je NPM kojeg dosad nisam previše konkretno objasnio. NPM[30] (*Node Package Manager*) je kolekcija gotovih JavaScript modula. Njega smo koristili kad smo birali pakete koje želimo koristiti u našoj *Gulp* konfiguraciji. Osim *Gulp* paketa, s njim možemo skinuti štogod nam treba, bez obzira radimo li aplikaciju s Reactom ili nešto sasvim treće. Korištenje gotovog, ispravno napisanog koda koji je dobro objašnjen u njegovoj dokumentaciji se dokazalo kao izrazito koristno i praktično. Nadam se da sam uspio prikazati koliko je bitno znanje programiranja kako bi danas-sutra mogli raditi ovakav posao u struci te da običan HTML i CSS nažalost nisu dovoljni.

Naglasak ovog rada bio je na načinu na koji se tema radi, stoga nisam previše pažnje obraćao na dizajnerske detalje. Postoje mnoge teme na koje se utrošilo puno više vremena te koje su postale toliko popularne da su developeri počeli raditi dodatke samo za tu temu koristeći tehnologije koje sam pokazao u ovom radu. Na taj način su ju proširili te je ona postala univerzalnija. Na kraju krajeva, moja tema je samo primjer dok u praksi postoje milijuni dobro dizajniranih i funkcionalnih tema.

Za kraj ću ostaviti linkove na kojima možete vidjeti gotovu temu i kod s kojim je izrađena:
Kod: https://github.com/WhitePointX/Mayhem
Tema: http://www.domagojzganec.com.hr/private/Mayhem/

U Varaždinu, _____

_____

9. Literatura

[1] https://www.vagrantup.com/, dostupno 12.07.2016.
[2] http://www.infoworld.com/article/3026209/linux/why-linux-is-still-better-than-windows-10.html,dostupno 21.9.2016.
[3] http://www.makeuseof.com/tag/6-superb-reasons-use-linux-programming/, dostupno 21.9.2016
[4] http://sass-lang.com/documentation/file.SASS_REFERENCE.html, dostupno 12.07.2016.
[5] https://www.w3.org/Style/CSS/ , dostupno 21.9.2016.
[6] https://modernizr.com/, dostupno 12.07.2016.
[7] https://developer.mozilla.org/en-US/docs/Web/SVG , dostupno 21.9.2016.
[8] https://cssanimation.rocks/transition-vs-animation/ , dostupno 21.9.2016.
[9] http://www.w3schools.com/html/html5_video.asp , dostupno 21.9.2016.
[10] https://www.advancedcustomfields.com/, dostupno 12.07.2016.
[11] http://www.yac.mx/hr/pc-tech-tips/hardware/How_to_Create_and_Use_Virtual_Machines.html, dostupno 21.9. 2016.
[12] https://www.virtualbox.org/, dostupno 12.07.2016.
[13] www.vagrantcloud.com, dostupno 12.07.2016.
[14] https://puphpet.com/, dostupno 12.7.2016.
[15] http://www.ubuntu.com/ , dostupno 21.9.2016.
[16] https://www.debian.org/index.en.html , dostupno 21.9.2016.
[17] http://www.popwebdesign.net/popart_blog/2014/12/sta-je-sass/, dostupno 12.07.2016.
[18] http://getbootstrap.com/ , dostupno 21.9.2016.
[19] http://www.w3schools.com/cssref/sel_class.asp , dostupno 21.9.2016.
[20] http://bourbon.io/, dostupno 12.07.2016.
[21] http://neat.bourbon.io/ dostupno 12.07.2016.
[22] https://bower.io/ dostupno 12.07.2016.
[23] http://learnlayout.com/clearfix.html , dostupno 21.9.2016.
[24] http://gulpjs.com/ dostupno 12.07.2016.
[25] https://nodejs.org/en/ , dostupno 21.9.2016.
[26] https://www.advancedcustomfields.com/add-ons/options-page/, dostupno 12.07.2016.
[27] https://git-scm.com/, dostupno 12.07.2016.
[28] http://www.owlgraphic.com/owlcarousel/, dostupno 12.07.2016.
[29] http://owlgraphic.com/owlcarousel/#customizing , dostupno 21.9.2016.
[30] https://www.npmjs.com/ dostupno 12.07.2016., dostupno 21.9.2016

10. Popis slika

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Domagoj Žgomec *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/~~diplomskog~~ *(obrisati nepotrebno)* rada pod naslovom Izrada teme za Wordpress CMS korištenjem SASSa i Gulpa *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Žgomec Domagoj
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Domagoj Žgomec *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom Izrada teme za Wordpress CMS korištenjem SASSa i Gulpa *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Žgomec Domagoj
(vlastoručni potpis)