Tallinna Ülikool
Digitehnoloogiate instituut

Staatilise kujunduse lisamine sisuhaldussüsteemis Drupal 8

Bakalaureusetöö

Autor: Siim Viisut

Juhendaja: Inga Petuhhov

Autor: ....,,....,,2017

Juhendaja: ....,,....,,2017

Instituudi direktor: ....,,....,,2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

……..(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Siim Viisut (sünnikuupäev: 04.11.1991)

1. Annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Staatilise kujunduse lisamine sisuhaldussüsteemis Drupal 8“, mille juhendaja on Inga Petuhhov, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas,

..... (allkiri ja kuupäev)

Sisukord

Sissejuhatus

Tänapäeval on suur konkurents, et leida ning hoida kasutajaid enda tarkvara juures. Lisaks tarkvara heale äriloogikale, eeldab see ka head kasutajakogemuse disaini ning rakenduse kujundust. Rakenduse kujundus ning töökindlus mängib olulist rolli kasutajate püüdmisel, hea esmamulje jätmisel ning usalduse tekitamisel.

Käesolevas töös käsitletakse Drupal 8 tarkvara, mis annab veebisaidi loomisel aluseks baasfunktsionaalsuse, et vältida nö. jalgratta leiutamist. Funktsionaalsuse lisamiseks on võimalik kasutada kogukonna poolt arendatud mooduleid ning arendada ise soovitud lisa loogikat. Drupali projekt jaguneb eraldi äriloogika- (ingl. k *back-end*) ning esitluskihiks (ingl. k *front-end*). Äriloogikakihis luuakse kogu peamine lehe loogika ning esitluskiht vastutab andmete töötlemise eest *HTML* märgistuskeeleks, mida veebilehitsejad kasutajale kuvavad. Äriloogikakihis on kasutusel *PHP* ning esitluskihis *HTML*, *Javascript*, *CSS* keeled ning *Twig* mallimootor. Arendustöö mahu jaotus äriloogika- ning esitluskihi vahel sõltub suuresti projekti olemusest. Isiklikule kogemusele põhines arvab autor, et see jaguneb 70% äriloogika- ning 30% esitluskihi arendusele suurtemates projektides ning väiksemates projektides kasvab pigem esitluskihi osakaal.

Drupali põhiste projektide arendamisele spetsialiseerunud arendajad ei pruugi olla kujunduste loomisel nii pädevad, kui nende loomisele pühendunud spetsialistid. Seepärast on mõistlik kasutada kujundust, mis on loodud hea disaineri poolt ning Drupali rakendust, mis on loodud Drupali arendaja poolt. Kuna Drupali tarkvara loob omaselt spetsiifilist *HTML* märgistuskeelt, mida disainer tingimata tundma ei pea, jääb Drupali arendaja ülesandeks siduda disaineri loodud kujundus Drupali esitluskihiga. Sellise meetodiga kasutatakse ära mõlema osapoole tugevusi. Autor on osalenud neljas projektis, kus kasutati väljaspoolt, kolmanda osapoole, loodud kujundust ning sellega vähenes Drupali esitluskihi arendamisele kulunud aeg keskmiselt 40%. Sama meetod sobis erinevate kujunduste puhul ning sammud mida iga disainielemendi, näiteks menüüd, nimekirjad ja pildigaleriid, puhul kasutada tuli olid väga sarnased.

Kujunduse sisumisel oli kaks meetodit, muuta kujunduse lähtekoodi, et see töötaks Drupali platvormil või muuta Drupali poolt genereeritavat veebilehe *HTML* märgistuskeelt, kujunduse jaoks sobivale kujule. Võrdlemisi algusjärgus selgus, et kiirem lahendus on seda teha Drupali poolel, sest Drupali arendajad ei pruugi kursis olla täpselt kujunduse iseärasustega näiteks kasutatud teekide funktsionaasusega. Peale

kujunduse sidumist on võimalik teha täiendavaid kohandusi, vastavalt vajadusele, mis veebisaidi kujunduse omanäolisemaks muudaks.

Käesolevas bakalaureusetöös on autor seadnud eesmärgiks tutvustada meetodeid ning samme, mida on võimalik kasutada, disaineri poolt loodud, staatilise kujunduse sidumiseks Drupal 8 sisuhaldussüsteemiga. Antud informatsiooni abil on võimalik vähendada Drupali esitluskihi loomisele kuluvat aega, kasutades endiselt kvaliteetset disaini.

Töö on jaotatud teoreetiliseks ning praktiliseks osaks. Teoreetilises osas tutvustatakse erinevaid Drupali esitluskihi juurde kuuluvaid komponente, mille abil genereeritakse Drupali veebisaidi *HTML* märgistuskeel. Lisaks käiakse lühidalt üle erinevat tüüpi staatilistest kujundustest, mida on võimalik hankida. Praktilises osas luuakse esmalt uus Drupali teema (ingl. k *theme*) ning arendustöö lihtsustamiseks, lülitatakse sisse *Twig debug*. Seejärel läbitakse kaks autori poolt loodud näidet, mille kaudu tutvustatakse samme kujunduse sidumiseks loodud Drupali teemaga.

Lihtsustatult on tegemist kolme sammuga. Esimeseks kujunduse leidmine. Teiseks staatilise kujunduse *CSS* ning *Javascripti* failide lisamine Drupali teemale. Kolmandaks Drupali teemale malli loomine, mis genereeriks vajaliku *HTML* märgistuskeelega komponendi, kuid staatiline sisu oleks vajadusel asendatud dünaamilise sisuga, mis tuleb Drupali sisuhaldusest.

Tööst arusaamise eelduseks on baasteadmised Drupal 8 sisuhaldustarkvarast ning *HTML*, *Javascript*, *CSS* keeltest ja *Twig* mallimootorist. Töös kirjeldavate sammude läbi proovimiseks on tarvis töötavat Drupal 8 veebisaiti, koos seadistatud Drupal Commerce mooduliga ning tema alammoodulitega. Samuti kasutatakse näidetes käsurealiidest ning *Drush* tarkvara. *Drush* tarkvara abil on võimalik läbi käsurealiidese teha erinevaid Drupali toiminguid kiiremini ning mugavamalt, kui kasutades kasutajaliidest. Näiteks võtab vähem aega Drupali teema aktiveerimine ning Drupali puhvri tühjendamine.

Antud tööd saab kasutada iseseisvalt või varem kirjutatud seminaritöö “E-poe tarkvara Drupal Commerce ülevaade ja seadistamine” järjena. Seminaritöös tutvustatakse täpsemalt Drupal 8 sisuhaldussüsteemi ning e-poe tarkvara Drupal Commerce. Lisaks installeeritakse ning seadistatakse Drupali rakendus. Käesolevas töös kasutatav algne rakendus on seminaritöö tulemusena saadud rakendus.

1. Drupali esitluskihi ning staatiliste kujunduste tutvustus

Antud peatükis tutvustatakse Drupali esitluskihti ning sellega seotud komponente, mille mõistmine on vajalik töö praktilise osa mõismiseks. Antud komponentideks on Drupali regioonid, plokid ning mallid. Esitluskihi tutvustamisel räägitakse lühidalt ka Drupali teemadest. Töö praktilises osas käsitletakse Drupali teema loomist põhjalikumalt.

1.1 Esitluskiht

Drupal 8 tarkvara kasutab esitluskihis *Twig* mallimootorit. Äriloogikakihis valmistatakse ette vajalikud andmed, mis esitluskihis töödeldakse veebilehe *HTML* märgistuskeeleks (ingl. k. *HTML markup*), mida veebilehitseja oskab lugeda ning ekraanile kuvada. *Twig* mallimootor pakub erinevaid funktsioone ning võimalusi, mida saab mallides kasutada andmete töötlemiseks, näiteks tingimuslauseid (ingl. k. *if clause*) ja tsükleid (ingl. k. *loop*).

Äriloogikakihis koostatud andmed peavad järgima teatud kokkulepitud struktuuri, mida esitluskiht oskab automaatselt töödelda. Antud struktuuriga andmeid nimetatakse esitlusmassiivideks (ingl. k. *render array*) ning sellisel kujul hoitakse andmeid, lõpliku *HTML* märgistuskeele genereerimiseni, et pakkuda andmete muutmise võimalust Drupali moodulitele ning teemadele. Lisaks andmetele, mida soovitakse esitleda, sisaldab esitlusmassiiv näiteks informatsiooni aktiivse malli kohta, andmete puhverdamise ning elemendi tüübi kohta (vt koodinäide 1). (Render arrays, 2017)

```
$page = [
  '#type' => 'page',
  '#theme' => 'page',
  '#cache' => [...],
  'content' => [
    'system_main' => [...],
    'another_block' => [...],
    '#sorted' => TRUE,
  ],
  'variable' => 'Lorem ipsum...',
  'sub_render_array' => [
    ...
  ],
];
```

Koodinäide 1. Esitlusmassiivi näide kooditasandil.

Veebisaidi kujunduse eest vastutab Drupali sisuhalduses aktiivseks määratud teema. Vaikimisi on peale installeerimist aktiivne *Bartik* nimeline teema, mis tuleb kaasa tuumfunktsionaalsusega (ingl. k. *core*). Eraldi on võimalik määrata aktiivne teema administreerimisliidese jaoks milleks on vaikimisi *Seven* teema. Teemade kood käivitub peale mooduleid ehk selleks ajaks on moodulite poolt andmed töödeldud ning aktiivne teema saab omakorda võimaluse täiendavalt antud andmeid käsitleda. Peale andmete töötlemist kasutatakse mallide olemasolu korral antud teema mallifaile, et genereerida *HTML* märgistuskeel. Juhul kui aktiivne teema ei sisalda kasutatavaid malle, taandub Drupal baasteema (ingl. k. *base theme*) mallidele. Olukorras, kus aktiivsele teemale pole baasteemat määratud, taandutakse *Stable* teema mallidele (Stable theme, 2017).

Lõplik andmete kogum, mille alusel luuakse veebilehe *HTML* märgistuskeel, moodustub paljudest pisematest tükkidest. Antud tükkide loomisega tegelevad erinevad moodulid. Näiteks tulevad plokkide esitlusmassiivid *Block* mooduli, kommentaaride sektsioon *Comment* mooduli ning kasutajate profiilid *Profile* mooduli poolt. Teistele moodulitele ning teemadele antakse samuti võimalus antud andmeid muuta, kasutades haak (ingl. k. *hook*) funktsioone. Näiteks esitlusmassiivide puhul, mille "#type" väärtus on "page", on võimalik antud massiivi andmeid muuta *hook_preprocess_page(&$variables)* funktsiooni abil. *$variables* muutuja sisaldab esitlusmassiivi, mis on kasutusel referentsina ning tänu sellele otse muudetav. Mallide loomisel on vaja teada, mis kujul antud andmed täpselt malli jõuavad, et neid korrektselt kasutada. Näiteks *hook_preprocess_page* funktsiooni puhul asub esitlusmassiiv "*$variables->page"* all ning kui määrata esitlusmassiivi mõne osa "#access" väärtus "FALSE*"* olekusse, ei looda antud elemendi märgistuskeelt (vt koodinäide 2). (Understanding Hooks, 2016)

```php
function hook_preprocess_page(&$variables) {
  $variables['page']['header']['#access'] = FALSE;
}
```

Koodinäide 2. haakfunktsiooni näide, kus “Header” regiooni kuvamine keelatakse ära. Antud elementi ekraanile ei teki.

Teemasid, malle, esitlusmassiive, haakfunktsioone ning teisi kujunduse loomiseks vajalike komponente kasutatakse antud töös, et muuta erinevate komponentide lõpliku *HTML* väljundit.

1.2 Regioonid

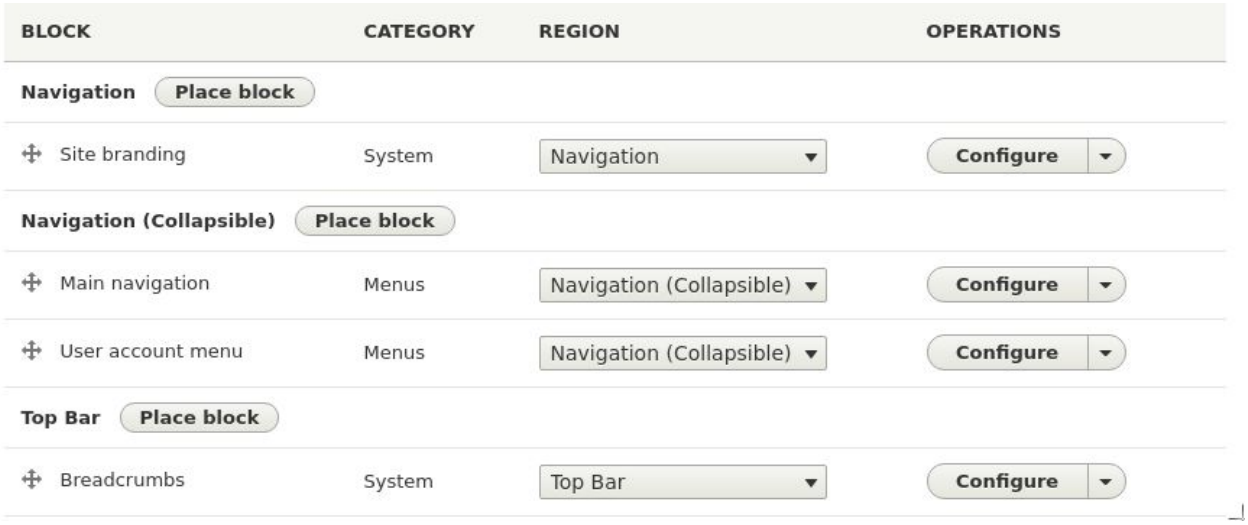
Regioonid on Drupali teema puhul kõige madalamaks komponendiks. Nende abil luuakse veebisaidi üldine kompositsioon. Tihti kasutatavad regioonid on näiteks jalus, päis ning sisuosa. Lisaks jaotatakse sisuosa veel täiendavateks regioonideks, näiteks parem külgriba, peamine sisuosa ning vasak külgriba. Drupali vaikimisi teema *Bartik* regioonide nägemiseks saab külastada veebilehitsejas suhtelist aadressi ***/admin/structure/block/demo/bartik*** (vt joonis 1). Antud lehel kuvatakse visuaalselt kõik regioonid ilma sisusta, et saada kiire ülevaade antud teema kompositsioonist. Veebisaiti külastades on näha, et “*Sidebar first*” regioon sisaldab vaikimisi “*Search*” otsinguvormi ning “*Tools*” menüüd. Regioonid on defineeritud Drupali teema failis ning “page” tüüpi mallides on võimalik neid lehe üldiseks ülesehituseks kasutada. Praktilses osas defineeritakse ning kasutatakse samuti regioone lehe üldise kompositsiooni kohandamiseks. (Adding Regions to a Theme, 2017)

Joonis 1. Lehe /admin/structure/block/demo/mytheme ekraanitõmmis, kus on kuvatud aktiivse teema kasutusel olevad regioonid.

1.3 Plokid

Plokid (*block*) on individuaalsed sisutükid, mis asuvad regioonide sees. Näiteks päises ning jaluses asuvad menüüd, pealkiri ning pealkirja all olev sisu on kõik erinevad plokid. Plokke saab hallata suhtelisel aadressil ***/admin/structure/block***. Plokid on kuvatud regioonidesse jaotatuna, iga regiooni nime kõrval asuva nupu “Aseta plokk” (ingl. k. *Place block*) kaudu on võimalik lisada plokk eeldefineeritud plokkide seast või luua uus. Olemasolevate plokkide regioone ning järjekorda saab vahetada, lohistades

plokk soovitud regiooni sobivasse järjestusse (vt joonis 2).



Joonis 2. Lehe */admin/structure/block* ekraani tõmmis, olemasolevate plokkide regiooni haldamise kohta.

Enamasti on plokid loodud erinevate moodulite poolt, näiteks ostukorvi plokk luuakse *Commerce Cart* mooduli poolt. Samuti on võimalik läbi kasutaja liidese luua lihtsakoelisi plokke, kuid need on enamasti staatilise sisuga ning sobivad pigem näiteks informatiivseteks plokkideks. Lehe peamine sisu asub ”Peamine lehe sisu” (ingl. k. *Main page content*) plokis. Näiteks “My account” lehel on peamiseks sisuks sisselogitud kasutaja andmed. Töö praktilises osas liigutatakse mõned plokid soovitud regioonidesse, et vastavate plokkide sisu kuvataks soovitud regioonidesse. (Working with blocks, 2017)

1.4 Mallid

Suure osa Drupali teema loomisest ning staatilise kujunduse sidumisest moodustab mallide tundmine ning muutmine. Mallides kasutatakse *HTML* märgistuskeelt koos *Twig* raamistiku funktsionaalsusega, et töödelda äriloogikakihist saabunud andmed sobivale *HTML* märgistuskeele kujule. Drupal 8 puhul on mallide failid **.html.twig** laiendiga ning nad peavad asuma mooduli või teema kaustas **templates** kausta sees. Antud kaustas võib soovi korral malle täiendavalt kaustadesse kategoriseerida, et säilitada arusaadavam struktuur (vt joonis 3).

Joonis 3. *Stable* teema mallid asuvad **templates** kaustas ning on täiendavalt kategoriseeritud näiteks **admin** ning **block** kaustadesse.

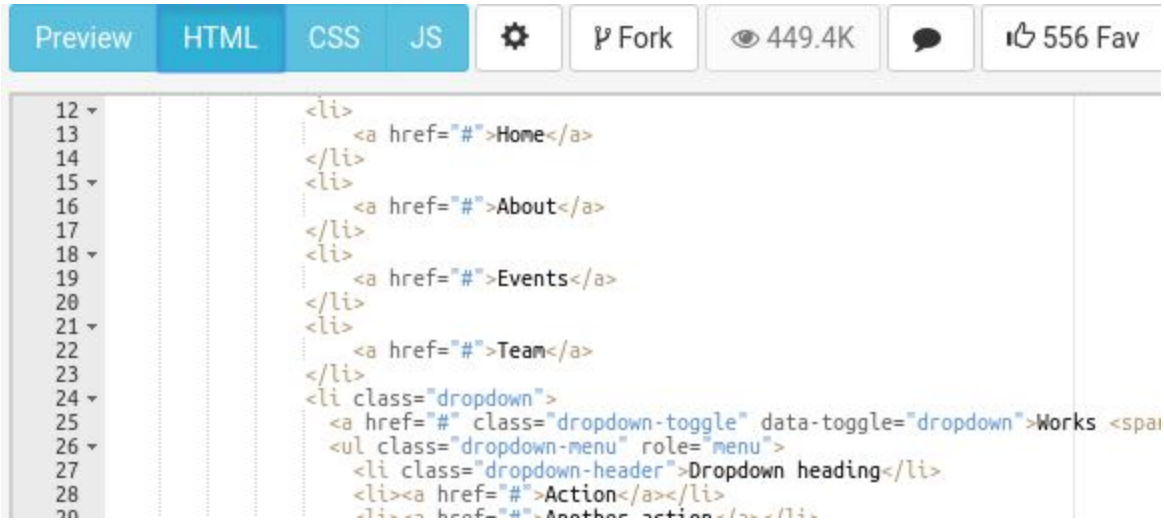
Ühesuguse nimega malle on tihti defineeritud mitmes kujunduses või mõnes moodulis, kuid Drupali süsteemil on priotiteetide järjekord, mille alusel aktiivne mall valitakse. Kõige kõrgema prioriteediga otsitakse malli aktiivsest teemast, seejärel aktiivse teema baasteemast ning lõpuks moodulitest, mooduli kaalude alusel. Sellise prioriteetide järjekorraga on võimalik üle kirjutada, mooduli poolt loodud komponendi mall, kasutades samanimelist malli aktiivses kujunduses. Lisaks võib ühele komponendile sobida mitu malli, millest aktiivne valitakse taas komponendi põhise malli prioriteedi järgi. Näiteks igale regioonile sobib **region.html.twig** nimeline mall, kuid lisaks on võimalik igale regioonile luua spetsiifiline mall kasutades **region--[region].html.twig** nimelist malli, kus "[region]" tuleb asendada regiooni nimega. Kui mõnele mallile on loodud seda tüüpi spetsiifiline mall, siis antud regioon kasutab antud malli ning ülejäänud regioonid kasutavad üldist regiooni malli. Lihtsustatult on võimalik kasutada alternatiivseid mallide nimesid, et kitsendada komponentide ringi, millele antud mall rakendub. Mallide üle kirjutamist ning muutmist käsitletakse sügavamalt töö praktilises osas. (Chaz, C, 2017)

1.5 Staatiline kujundus

Antud töö raames mõeldakse staatilise kujunduse all *HTML, Javascript* ning *CSS* abil kirjutatud kujundusi, mille sisu on staatiline ning kood ei ole tingimata ühilduv Drupali poolt genereeritava lähtekoodiga. Kujundus võib sisaldada vaid ühe komponendi disaini või terve veebisaidi kujundust, alustades disainitud menüüdest ning sisulehtedest ja lõpetades veebivormide ning nuppude disainiga.

Enne kujunduse otsima asumist, tuleks seada paika teatud kriteeriumid, mis kitsendaksid potensiaalsete kujunduste valikut. Tasub läbi mõelda, mis tüüpi veebisaidiga on tegemist, näiteks portfoolio, e-poe või blogiga. Samuti on kasulik otsustada, kas otsitakse tasulist või tasuta kujundust. Lisaks saab valida, kas soovitakse leida tervele lehele ühte disaini või kasutada erinevaid komponentide põhiseid disaini osasid. Näiteks päise, sisulehtede ja jaluse disaine.

Leheülese kujunduse puhul on, peale selle leidmist või ostmist, üldiselt tulemuseks failide kogum, mis sisaldavad antud disaini lähtekoodi ning mida saab allalaadida. Komponendipõhiste disainide puhul võib sattuda olukorda, kus lähtekood pole allatõmmatav, vaid kuvatakse veebilehel välja ning on kopeeritav näiteks eraldi *CSS*, *Javascript* ning *HTML* koodide kaupa (vt joonis 4).



Joonis 4. Näide komponendipõhise kujunduse koodist, mida on võimalik kopeerida *CSS*, *Javascript* ning *HTML* koodi kaupa.

1.5.1 Tasuline kujundus

Disainiagentuuri või mõne teise teenusepakkuja käest on võimalik tellida spetsiaalselt soovitud funktsionaalsuse jaoks loodud kujundus. Sellise kujunduse puhul peab peale, sidumist Drupali teemaga, tegema kõige vähem kohandusi, sest soovitud disain on juba eelnevalt valmistatud spetsiaalselt antud veebisaidi tarvis. Seda tüüpi kujundus on enamasti ka kõige kallim.

Teiseks variandiks on leida kujundus internetist, mis on eelnevalt valmistatud ning peale ostu allalaetav. Internetis on kujunduse leidmiseks võimalik kasutada erinevaid keskkondi, mille kaudu disainerid oma loometööd müüvad. Näiteks populaarne veebisait "*Themeforest.net*" sisaldab pea 40 000 disaini, mis on jaotatud erinevate kategooriate vahel vastavalt veebilehe tüübile, seadmele või isegi sisuhaldussüsteemile (WordPress Themes & Website Templates, kuupäev puudub). Teisteks alternatiivideks on näiteks "*WrapBootstrap.com*" ning "*Free-css.com*". Antud kujunduste miinuseks on see, et neid on võimalik osta ning kasutada kõigil ehk suure tõenäosusega võib sama disaini kasutada ka mõni teine veebisait. Lisaks on peale Drupali teemaga sidumist suure tõenäosusega vaja teha kujundusele täiendavaid täiendusi, sest disain ei pruugi olla loodud spetsiaalselt kliendi veebisaidili. Siiski on võimalik ostetud disain Drupali

teemaga siduda ning vastavalt soovile kujundada omanäolisemaks. Internetist ostetud kujundused on enamasti kordades soodsamad kui spetsiaalselt tellitud disainid.

1.5.2 Tasuta kujundus

Tasuta kujundusi on mugav kasutada, kui soovitakse kiirelt ehitada üldine lehe ülesehitus, käitumine ning hiljem lisada kohandatud disain. Leheüleseid kujundusi on võimalik leida näiteks "*Bootstrapmade.com*" ja "*Tooplate.com*" veebisaitidelt. Samuti on võimalik otsida kujundust erinevatele komponentidele eraldi. Näiteks "*Bootsnip.com*" ning "*bootdey.com*" portaalides on võimalik leida kujundusi toodetele, uudistele, menüüdele ning lehe üldise kompositsiooni jaoks.

Antud töö raames kasutatakse komponentide põhiseid kujundusi, et koodi maht oleks väiksem ning arusaadavam, kuid samad võtted ning tegevused kehtivad ka leheülese disaini puhul. Lisaks on autor kasutanud peamiselt *Bootstrap* raamistikuga kokkusobivaid disainikomponente, et kasutada ära antud raamistiku võimalusi kujunduse loomiseks. Kasutatakse vaid tasuta, avalikult kättesaadavaid komponente, et lugejal oleks võimalik praktilisi näiteid mugavamalt kaasa teha.

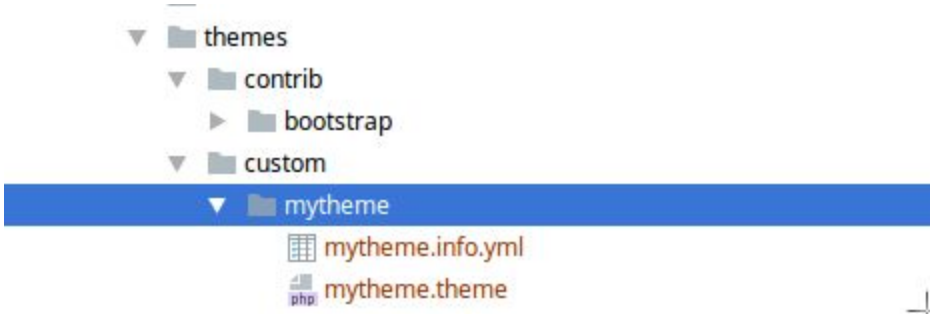
2. Staatilise kujunduse sidumine Drupali teemaga

Praktilses osas käiakse läbi protsess alates uue Drupal 8 teema loomisest kuni staatilise kujunduse sidumiseni loodud teemaga. Lisaks lülitatakse sisse *Twig debug* lahendus, mis lihtsustab mallide leidmise ning muutmise protsessi. Kujunduse sidumisel käsitletakse staatilise kujunduse *CSS* ning *Javascript* failide lisamist Druapli teema külge ning mallide muutmist sobivale kujule, et saavutada soovitud kujundus ka dünaamilise sisuga. Mõnel puhul kasutatakse erinevate tegevuste kiiremaks läbiviimiseks, varasemalt mainitud, *Drush* tarkvara. Autori poolt praktilise tööna valminud lähtekood on leitav autori *Bitbucket* keskkonnast aadressil https://bitbucket.org/viisutprojects/drupal-8-implementing-static-design/overview.

2.1. Uue teema loomine

Drupali tuumfunktsionaalsusega kaasa tulevad tuumteemad (ingl.k. *core theme*) annavad kõikidele Drupali veebisaitidele algse kujunduse. Lisaks on võimalik kasutada kogukonna poolt välja töötatud teemasid (ingl.k. *contributed theme*), mis on avalikult kasutatavad. Enamiku projektide jaoks tuleb siiski luua kohandatud teema (ingl.k. *custom theme*), mis arvestab näiteks antud veebisaidi iseloomuga, eesmärgiga ning sihtgrupiga. Uue teema ehitamist on võimalik alustada minimalistliku *Stable* tuumteema peale või kasutada baasteemana mõnda olemasolevat teemat. Baasteemat kasutades näeb uus teema esialgu välja täpselt samasugune nagu valitud baasteema. Antud töö raames loome täiesti uue kujunduse ilma baasteemata, et vältida konflikte baasteema ning leitud staatilise kujunduse vahel.

Drupali süsteem tunneb ära, et uus teema on valmis, kui teatud struktuuri järgivad failid asuvad projektis õiges kaustas. Teemade puhul on selleks kaustaks projekti juurkataloogis asuv **themes** kaust. Hea tava on eraldada kogukonna poolt valmistatud ning projekti jaoks arendatud teemad eraldi, vastavalt **themes/contrib** ning **themes/custom** kaustadesse. Uue teema loomiseks tuleb **themes/custom** kausta luua uus kaust enda soovitud teema nimega. Antud töös on selleks valitud *mytheme*. Äsja loodud kujunduse kausta tuleb lisada kaks faili, **mytheme.info.yml** ning **mytheme.theme** (vt joonis 5).



Joonis 5. Käsitsi loodud teema esmane failipuu.

Antud failinimede puhul tuleb **mytheme** osa muuta vastavalt soovitud kujunduse nimele. Laiendiga **.info.yml** fail sisaldab informatsiooni teema struktuuri ning kirjelduse kohta. Teine **.theme** laiendiga fail käivitatakse iga kord, kui antud teemat kasutav lehekülg laetakse. Antud faili kirjutatakse äriloogikat, näiteks kasutatakse haakfunktsioone. Esialgu võib antud faili sisu tühi olla, kuid faili olemasolu on oluline. Teema ülesehituse **info.yml** faili sisu peab sisaldama teatud kohustuslike atribuute, näiteks nime (ingl.k. *name*) ja tüüpi (ingl.k. *type*). Järgnevas näites on toodud välja vähim vajalik teemakirjeldus. Regioonideks (ingl.k. *regions*) on esialgu lisatud sisuosa "*Content"*, vasak külgriba "*Left Sidebar"* ning väljalülitatud "*Disabled"* regioonid (vt koodinäide 3). (Defining a theme with an .info.yml file, 2017)

```
name: mytheme
type: theme
description: My Awesome theme
core: 8.x

regions:
 left_sidebar: 'Left sidebar'
 content: 'Content'
 disabled: 'Disabled'
```

Koodinäide 3. Teema .info.yml faili sisu, kus on defineeritud näiteks nimi, tüüp, kirjeldus, baasteema ning regioonid.

Peale antud failide loomist on järgnevate käsurea käskudega võimalik uus teema aktiveerida ning määrata vaikimisi teemaks (vt käsurea käsud 1).

```
drush en mytheme -y
drush config-set system.theme default mytheme -y
```

Käsurea käsud 1. Teema aktiveerimiseks ning vaiketeemaks määramise käsud.

Äsja loodud teema aktiveerimise järel näeb veebisait välja võrdlemisi tühi. Mitmed plokid võivad minna väljalülitatud olekusse, sest vaiketeema muutmisel, muutusid ka regioonide nimed. Soovitud plokid saab tagasi sisse lülitada ning sobivasse regiooni määrata plokkide haldamise lehelt (vt joonis 2).

2.2 Twig debug

Twig debug on väga hea tööriist, millega leida informatsiooni aktiivses kasutuses olevad mallide kohta. Lisaks *Twig debug* sisselülitamisele on kasulik välja lülitada mallide puhverdamine, et näha mallide muudatusi ilma puhvrit puhastamata. Antud muudatused tuleb teha **/sites/default/services.yml** failis, muutes kolme rea väärtused vastupidiseks. Tulemuseks jäävad järgneval näitel välja toodud väärtused (vt koodinäide 4).

```
debug: true
auto_reload: true
cache: false
```

Koodinäide 4. **Services.yml** sisu millega lülitatakse sisse *Twig debug* ning lülitatakse välja mallide puhverdamine.

Muudatuste aktiveerimiseks tuleb Drupali puhver puhastada (vt. käsurea käsud 2). Antud muudatustega lülitatakse välja mallide puhverdamine, kuid uute mallide loomisel ning Drupali kujunduses olevates **.yml** failides erinevate muudatuste tegemisel tuleb siiski puhvrit puhastada, et Drupal uue sisu registreeriks.

```
drush cache-rebuild
```

Käsurea käsud 2. Käsk puhvri puhastamiseks.

Twig debug lisab kommentaaridena veebilehe lähtekoodile informatsiooni aktiivsete mallide ning nimetamise viiside kohta, mis antud elemendile sobivad (Debugging Twig templates, 2017). Järgneval joonisel on kuvatud otsingu ploki malli informatsiooni (vt joonis 6). Aktiivselt on kasutusel **block--search.html.twig** nimeline mall, mis asub **themes/contrib/bootstrap/templates/block** kaustas. Seda malli on võimalik üle kirjutada, kasutades sama või kõrgema prioriteediga alternatiive mis on välja toodud “FILE NAME SUGGESTIONS” osas ehk **block--mytheme-search.html.twig** või **block--search-form-block.html.twig** nimedega malle.

Joonis 6. Veebilehe *lähtekoodi näide*, kus on näha rohelise tekstina mallide informatsioon, mille lisab *Twig debug*.

2.3. Staatilise kujunduse lisamine

Järgnevas peatükis käsitletakse tehnikaid, kuidas staatiline kujundus siduda Drupali teemaga nii, et kujunduses olev staatiline kood asendatakse vajadusel dünaamilise sisuga ning kujundusega kaasnevad *CSS* ning *Javascript* failid töötaksid samamoodi nagu staatilises kujunduses. Antud tegevuse tulemuseks on Drupalis kujundatud komponent, mis näeb välja ning toimib samamoodi nagu staatilises kujunduses nähtav lehekülg. Kuid staatilise sisu asemel on kasutusel Drupali sisuhaldusest tulev sisu, näiteks menüülingid ning kasutajate poolt sisestatud sisulehed.

Töö autor on valinud näidetena kasutud staatilised kujundused isikliku eelistuse järgi ning sidunud need Drupali teemaga. Valiku tegemise juures arvestas autor, et disainide puhul oleks nende *HTML* märgistuskeele maht piisavalt väike, et sobiks antud töö sisusse kommenteerida ning vähendada keerukust. Samas jälgis autor, et disain oleks visuaalselt meeldiv ning *CSS* ega *Javascript* koodi mahule piirangut ei seadnud.

Lõpptulemust silmas pidades ei ole vahet, kas esmalt lisatakse Drupali teemale staatilise kujunduse *CSS* ning *Javascript* failid või kirjutatakse vajalikud mallid üle. Autor eelistab alustada failide lisamisega, sest siis on mallide muutmisel näha kohest efekti ning vajadusel malle korrigeerida. Töös käsitletud näidetes lisatakse samuti esmalt kujunduse jaoks vajalikud failid ning seejärel tegeletakse mallidega.

Iga komponendi puhul on vaja süveneda ning otsustada, mis elemendid antud kujunduses peaksid olema dünaamilised ehk tulema Drupali sisuhaldusest ning mis osad võivad olla staatilised ehk lähtekoodi sissekirjutatud. Järgmises peatükis kasutatud staatilises kujunduses on kuvatud külgriba, kus asub lehe logo “Brand” ning mõned menüülingid näiteks “Home” ja “About” (vt joonis 7). Kindlasti on kasulik

antud teema sidumisel asendada menüülingid dünaamiliselt tekkivate menüülinkide vastu, kuid logo osas pole otsustamine enam nii lihtne. Teoorias võiks logo teksti või pildi kirjutada malli staatiliselt ning see oleks täiesti sobilik lahendus. Põhinedes isiklikule kogemusele teab autor, et logo saab asendada dünaamilise sisuga väga vähese vaevaga, seetõttu kasutatakse selleks dünaamilist sisu. Sellise lühikese näite taustal tasub meeles pidada, et Drupali teema loomisel ning staatilise kujunduse sidumisel on ühe tulemuse saavutamiseks tihti mitu erinevat võimalust. Samuti leidub töös kasutatud lahendustele teisi alternatiive.

2.3.1 Lehe üldine kompositsioon

Üldise lehe kompositsiooni all on mõeldud kõige madalamat kihti ehk millisteks sektsioonideks on lehekülje disain jaotatud. Antud töö raames on otsustas autor otsida kujunduse, kus veebileht oleks jaotatud vasakuks külgribaks ning sisuosaks. Külgriba on avatav/suletav ning sisuosa nihkub külgriba avamisel paremale poole. Eesmärgiks on kuvada logo ning peamenüü külgribal ning sisuosas näiteks pealkiri, sisu ja jalus. Antud lahendust pakub näiteks https://bootsnipp.com/snippets/featured/fancy-sidebar-navigation lehel leitav disain (vt joonis 7). Eelvaates (ingl.k. *Preview*) on võimalik näha ning testida, kuidas antud lahendus töötab ning "HTML", "CSS" ja "JS" nuppude pealt on võimalik näha lähtekoodi, mis kasutusel on.

Joonis 7. Kompositsiooni disain, kus leht on jagatud vasakuks külgribaks ning sisuosaks.

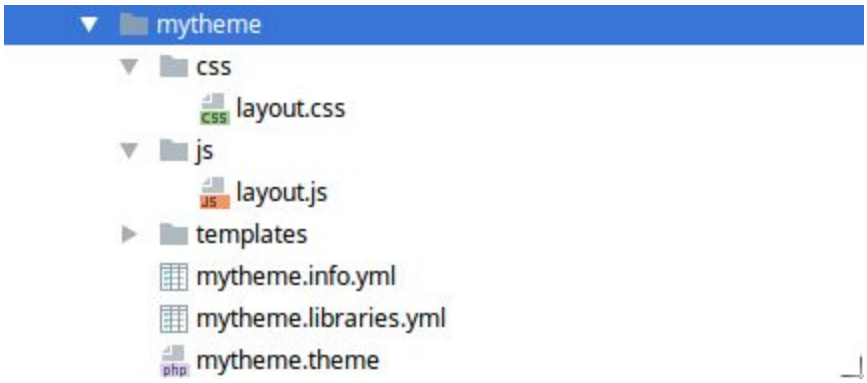
2.3.1.1 *CSS* ja *Javascript* lisamine

Esimese asjana tuleb sisuliselt kopeerida staatilises kujunduses ressurssidena kasutatud *CSS* ning *Javascript* failid Drupali teema kausta. Käesoleva näite puhul on kasutusel vaid üks *CSS* ja üks *Javascript* fail, kuid teiste kujunduste puhul võib neid rohkem olla. *CSS* ja *Javascript* failide jaoks on hea tava luua eraldi kaustad **css** ja **js**, et teema kaust jääks arusaadava struktuuriga. Antud kaustadesse tuleb lisada ressurssidena kasutatud failid, vabalt valitud nimedega. Antud töös nimetatid lisatud failid **layout.css** ning

layout.js (vt. joonis 8). Kui **js** failides on kasutusel *jQuery* on soovitatav antud kood panna näites esitatud koodijupi vahele (vt koodinäide 5). Sellega veendutakse et failis oleks lubatud kasutada *jQuery* koodi "$" kontekstis, mis on väga levinud (JavaScript API overview, 2017).

```
(function ($) {
  // Kood läheb siia vahele
}(jQuery));
```

Koodinäide 5. *Javascripti* kood läheb antud kooditüki vahele.



Joonis 8. *Mytheme* failide struktuur peale **css** ning **js** failide lisamist. Sisaldab ka **libraries.yml** faili.

Järgmisena on vaja defineerida Drupali teema teek (ingl.k. *library*), ehk failide komplekt, mida on võimalik veebilehele vajalikul hetkel laadida. Käesolevas näites moodustavad teegi äsja lisatud **layout.css** ning **layout.js** failid. Esiteks tuleb luua teema kausta uus fail **mytheme.libraries.yml** (vt joonis 7). Antud faili tuleb lisada esimese teegi definitsioon, kus kirjeldatakse teegi nimi ning kaasatud failide asukohad. Lisaks on ressursside hulgas *Bootstrap* failid, mis on staatilise kujunduse poolt nõutud, sest kujunduses kasutatakse *Bootstrap* raamistiku võimalusi. Viimase osana on sõltuvusena viidatud "core/jquery" teegile, mida on vaja, et lisatud **js** failides jääks *jQuery* funktsionaalsus tööle (vt koodinäide 6).

```
global-styling:
  css:
    theme:

https://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css: {
type: external, minified: true }
      css/layout.css : {}
  js:
    https://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js: {
type: external, minified: true }
    js/layout.js : {}
  dependencies:
    - core/jquery
```

Koodinäide 6. **mytheme.libraries.yml** faili sisu.

Kõik teegid, mida soovitakse laadida iga lehe laadimisel, saab lisada **info.yml** faili ning Drupal kaasab nad automaatselt lehele. Teegi nimi peab **info.yml** failis vastama **libraries.yml** failis seatud nimele (vt. koodinäide 7).

```
libraries:
  - mytheme/global-styling
```

Koodinäide 7. **mytheme.info.yml** faili lõppu lisatud osa, mille abil laetakse uus teek.

Peale failide salvestamist ning puhvri puhastamist (vt käsurea käsud 2), laeb Drupal antud teeki kuuluvad failid lehe ressursside hulka. Teema **libraries.yml** failis saab defineerida mitu teeki, kuid kõiki ei pea **info.yml** faili lisama. Ainult need teegid tuleb lisada **info.yml** faili, mida soovitakse igal lehe laadimisel kasutada. Teised teegid võivad olla näiteks komponendi põhised, mida soovitakse laadida ainult lehtedel, kus antud komponent kasutusel. (Adding CSS and JS to a Drupal 8 theme, 2017)

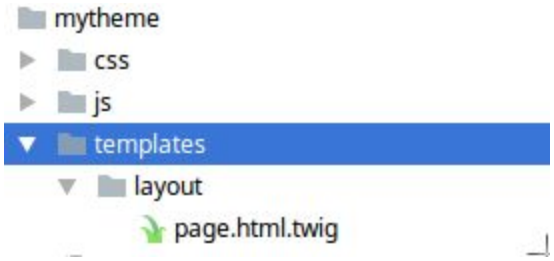
2.3.1.2 Mallide lisamine ning muutmine

Peale ressursside lisamist, tuleb kindlaks määrata mallid, mida on vaja üle kirjutada, et saavutada staatilise kujunduse ressurssidega kokkusobiv veebilehe HTML märgistuskeel. Õigete mallide leidmiseks on mugav kasutada, eelnevalt aktiveeritud, *Twig debug* lahendust. Kuna eesmärgiks on muuta lehe kompositsiooni, tuleb leida mall, mis selle eest hetkel vastutab. Veebilehitsejas lehe lähtekoodi uurides on mallide informatsiooni seas näha, et vaikimisi tuleb lehe kompositsioon “page” tüüpi mallist, sest antud malli haagiks (ingl.k. *Theme hook*) on “page”. Antud malli aitab kindlaks määrata see, et lähtekoodis on otse antud malli sees, üksteise järel, lehe erinevad sektsioonid näiteks päiseriba ning sisu, mis

moodustavad lehe kompositsiooni. Nende sees on omakorda Drupali regioonid, joonisel on näha "content" regiooni malli informatsiooni, lehe malli sees (vt joonis 9).

Joonis 9. *Page* malli informatsioon, mille sees asuvad erinevad Drupali regioonid.

Twig debug informatsioonist on näha, et aktiivne mall asub hetkel **core/themes/stable/templates/layout/page.html.twig** asukohas. Antud malli üle kirjutamiseks peab kopeerima selle faili kohandatud teema kasuta. Hea tava on kasutada sarnast kaustade struktuuri, et oleks mugavam leida ning muuta olemasolevaid malle, sest töö käigus võib neid tekkida suurtes kogustes. Peale malli kopeerimist ning Drupali puhvri puhastust (vt käsurea käsud 2), saab *Twig debug* abil näha, et aktiivseks malliks on muutunud uus fail. Selle näite puhul tuleb hea tava järgides kopeerida fail **themes/custom/mytheme/templates/layout/page.html.twig** asukohta (vt joonis 10) (Working With Twig Templates, 2017).



Joonis 10. *Mytheme* kaustade struktuur peale uue **page.html.twig** kopeerimist.

Viimase, kõige mahukama osana tuleb staatilise kujunduse *HTML* märgistuskeel tuua üle *Twig* mallidesse. Hetkel käsiloleva staatilise kujunduse märgistuskeelt näeb 2.3.1 peatükis mainitud veebilehelt "HTML" nupu alt. Lisaks külgribale ning sisuosale sisaldab see ka staatilisi menüüelemente ning lehe sisuteksti, mida tegelikult on vaja Drupalis dünaamiliselt tekitada. Lisaks näeb kopeeritud lehe mallis ning *Twig debug* abil, et menüü ja logo märgistuskeel genereeritakse teistes mallides. Seetõttu tuleb

samasuguse disaini saavutamiseks muuta ka teisi malle lisaks lehe mallile. Staatilise kujunduse sidumise üks keerulisemaid osasid on staatilise kujunduse ning Drupali *HTML* märgistuskeele võrdlemine. Võrdluse tulemusena saab tuletada, mis mallides saab erinevaid staatilise kujunduse osasid üle kirjutada. Käesoleva näite puhul selgub tegelikult, et kogu staatilise kujunduse sidumiseks tuleb lisaks lehe mallile kirjutada ka üle soovitud menüü mall. Autor eelistab alustada kõige laiemast mallist ning liikuda edasi järjest kitsamate juurde.

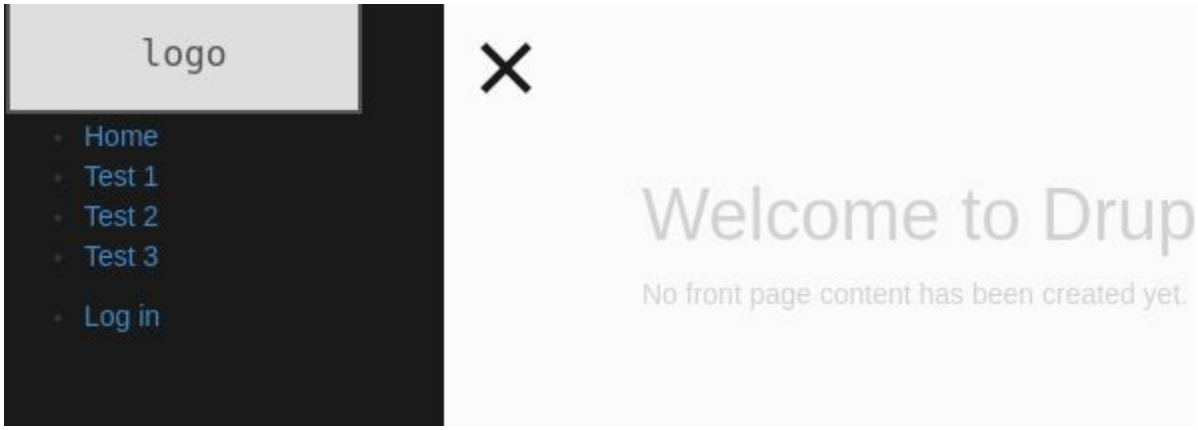
Järgnevas näites on näha tulemus, milleni autor jõudis (vt koodinäide 8). Kui näites olevad *Twig* spetsiifilised read kõrvale jätta, on tegelikult näha, et mall sisaldab praktiliselt üks ühele staatilise kujunduse *HTML* märgistuskeelt, ainult külgriba ning sisuosa staatiline sisu on asendatud Drupali regioonidega. Tegemist on “content” ning “left_sidebar” regioonidega, mis defineeriti uue teema loomisel **mytheme.info.yml** failis (vt koodinäide 3).

```
<div id="wrapper">
  <div class="overlay"></div>

  <!-- Sidebar -->
  {% block navbar %}
    <div class="navbar navbar-inverse navbar-fixed-top"
id="sidebar-wrapper" role="navigation">
      {{ page.left_sidebar }}
    </div>
  {% endblock %}
  <!-- /#sidebar-wrapper -->

  <!-- Page Content -->
  {% block main %}
    <div id="page-content-wrapper">
      <button type="button" class="hamburger is-closed"
data-toggle="offcanvas">
        <span class="hamb-top"></span>
        <span class="hamb-middle"></span>
        <span class="hamb-bottom"></span>
      </button>
      <div class="container">
        <div class="row">
          <div class="col-lg-8 col-lg-offset-2">
            {{ page.content }}
          </div>
        </div>
```

```
      </div>
    </div>
  {% endblock %}
  <!-- /#page-content-wrapper -->
</div>
```

Koodinäide 8. Uue **page.html.twig** sisu, mis tekitab soovitud külgriba ning sisuosa funktsionaalsuse.

Peale malli salvestamist on lehe ülesehitus jaotatud külgribaks ning sisuosaks vastavalt seatud eesmärgile, kuid külgriba sees olev menüü ei vasta veel kujunduses nähtavale (vt joonis 11). Juhul kui külgribal menüüd või logo ei kuvata, peab plokihaldusest kontrollima, kas vastavad plokid on sisse lülitatud ning "Left sidebar" regiooni liigutatud (vt joonis 2).



Joonis 11. Lehekülg on jaotatud külgribaks ning sisuosaks kuid külgriba menüü ei ole õigete *CSS* stiilidega.

Menüü kujunduse parandamiseks peab leidma, mis mall on hetkel antud menüü jaoks kasutusel ning selle üle kirjutama. *Twig debug* abil saab näha, et hetkel on nii peamenüü puhul aktiivseks malliks **core/themes/stable/templates/navigation/menu.html.twig**. Antud malli kasutavad hetkel vaikimisi kõik Drupali menüüd. Taas tuleb antud mall kopeerida uue teema mallide sekka, kuid seekord tasub kasutada spetsiifilisemat malli nime ehk **menu--main.html.twig**, et see rakenduks vaid peamenüüle. Peamine põhjus selleks on see, kui teisi menüüsid soovitakse mujal lehel kasutada, siis nendele jääks kehtima endine disain ning uut kujundust kasutaks vaid peamenüü. Sarnaselt lehe mallile tuleb võrrelda staatilises kujunduses ning Drupali veebilehel olevat *HTML* märgistuskeelt ning leida erinevused, mille tõttu *CSS* stiilid Drupalis ei rakendu. Tuleb kontrollida elementide unikaalseid identifikaatoreid ning klasse, mille alusel *CSS* reegleid luuakse. Võte, mida autor on kasutanud keerulisemate *Twig* mallide puhul, on kopeerida staatilise kujunduse tükk *Twig* malli sisse, et staatiline tükk tekiks Drupalisse dünaamilise tüki kõrvale. See võimaldab lähtekoodis kergesti võrrelda dünaamilist ning staatilist osa ning leida kiiremini erinevused, mis probleeme võivad tekitada. Sedasi võrreldes oli kerge märgata, et vaikimisi oli Drupali

mallis puudu menüüelemendil “nav” ning “sidebar-nav” klassid (vt koodinäide 9). Peale antud klasside lisamist menüü mallis, rakenduvad vajalikud *CSS* stiilid antud menüüle.

```
{% import _self as menus %}

{{ menus.menu_links(items, attributes, 0) }}

{% macro menu_links(items, attributes, menu_level) %}
  {% import _self as menus %}
  {% if items %}
    {% if menu_level == 0 %}
      <ul{{ attributes.addClass('nav', 'sidebar-nav') }}>
    {% else %}
      <ul class="dropdown-menu">
    {% endif %}
    {% for item in items %}
      {% if item.below %}
        <li{{ item.attributes.addClass('dropdown') }}>
          <a href="{{ item.url }}" class="dropdown-toggle" data-target="#"
data-toggle="dropdown">{{ item.title }} <span class="caret"></span></a>
          {{ menus.menu_links(item.below, attributes, menu_level + 1) }}
      {% else %}
        <li>
          {{ link(item.title, item.url) }}
      {% endif %}
        </li>
    {% endfor %}
    </ul>
  {% endif %}
{% endmacro %}
```

Koodinäide 9. Uue menüü malli sisu, millega töötab soovitud küljeriba sisu kujundus.

2.3.2 Toote detailvaade

Teise näitena on käsitletud rohkem e-poe spetsiifilisemat kujundust ehk toote detailvaadet. Näites kasutatud kujundus on leitav https://bootsnipp.com/snippets/featured/product-detail-page-ecommerce lehelt (vt joonis 12).

Joonis 12. Toote detailvaate staatiline disain.

2.3.2.1 *CSS* ja *Javascript* lisamine

Autor eelistab alustada taas staatilise kujunduse *CSS* ja *Javascript* lähtekoodi ületoomisega Drupali teemasse. Sarnaselt eelnevale näitele kujunduses on kasutusel üks **css** ning üks **js** fail, mis tuuakse Drupali teemasse **mytheme/css/product-detail.css** ning **mytheme/css/product-detail.js** failidesse. Järgmisena tuleb defineerida kujunduse **libraries.yml** failis uus teek, mille abil laeb Drupal ressursid veebilehele (vt. koodinäide 10).

```
product-detail:
  version: 1.0
  css:
    theme:
      css/product-detail.css : {}
  js:
    js/product-detail.js : {}
```

Koodinäide 10. **mytheme.libraries.yml** faili lisatud “product-detail” teek, millega laetakse **product-detail.css** ning **product-detail.js** fail.

Seekord ei ole vaja laadida antud teeki iga lehe laadimisel, vaid ainult toote detailvaate kasutamisel. Ehk **info.yml** faili pole uut kirjet vaja lisada, vaid selle asemel tuleb **mytheme.theme** faili lisada haakfunktsioon, mille abil kaasatakse antud teek ainult juhul, kui leht sisaldab toote detailvaadet (vt. koodinäide 11).

```
/**
 * Implements hook_preprocess_theme()
 */
function mytheme_preprocess_commerce_product(&$variables) {
  // Check if product is viewed as detail view.
  if ($variables['elements']['#view_mode'] == 'full') {
    // Load related resources if detail view.
    $variables['#attached']['library'][] = 'mytheme/product-detail';
  }
}
```

Koodinäide 11. **mytheme.theme** faili lisatud haakfunktsioon, millega äsja defineeritud teek laetakse koos tootega.

Kasutatud on "hook_preprocess_theme" haakfunktsiooni, kus "hook" osa on asendatud teema nimega ning "theme" tähistab soovitud komponenti, mille külge haakida soovitakse. Autor soovitab haakfunktsiooni "theme" väärtuse leidmiseks kasutada *Twig debug* abi, et leida veebilehel sobiv element üles ning kasutada antud elemendi juurde kuuluvat "Theme hook" väärtust, mis toodete puhul on "commerce_product" (Understanding Hooks, 2016). Peale antud muudatuste tegemist ning puhvri puhastust (vt käsurea käsud 2), laetakse teeki lisatud failid kui külastatav leht sisaldab toote detailvaadet.

2.3.2.2 Mallide lisamine ning muutmine

Twig debug abil näeb, et toote väljundi loomise eest vastutab mall, mille asukohaks on **modules/contrib/commerce/modules/product/templates/commerce-product.html.twig**. Antud fail tuleb kopeerida uue teema kausta. Leitud kujunduse *HTML* märgistuskeele seast otsustas autor eemaldada komponendid, mis on Drupali toote detailvaate jaoks ülearused, näiteks kommentaaride arv ja koguse muutmise osa. Koodi lihtsustamise jaoks, jättis autor alles vaid pealkirja, hinna, värvivaliku, ostukorvi lisamise ning toote tutvustuse osa. Taas saab sisuliselt kopeerida staatilise kujundus HTML koodi uude malli ning asendada soovitud sisu *Twig* muutujatega mis lisavad veebilehele dünaamilise sisu (vt. koodinäide 12). (Chaz, 2017)

```
<article{{ attributes.addClass('product-detail') }}>
  <div class="row">
    <div class="col-xs-4 item-photo">
      {{ product.field_product_images }}
    </div>
    <div class="col-xs-5" style="border:0px solid gray">
      <h3>{{ product.title }}</h3>

      <h6 class="title-price"><small>{{ 'Price'|trans }}</small></h6>
```

```
      <h3 style="margin-top:0px;">{{ product.variation_price }}</h3>

      <div class="section variations">
        {{ product.variations }}
      </div>
    </div>

    <div class="col-xs-9">
      <div style="width:100%;border-top:1px solid silver">
        {{ product.body }}
      </div>
    </div>
  </div>
</article>
```

Koodinäide 12. **Commerce-product.html.twig** sisu peale kopeerimist ning muutmist.

Peale malli muutmist on enamus kujundusest visuaalselt kujundusele sarnane, kuid värvivalimiku ja ostukorvi lisamise nupu ehk variatsioonide vormi osa on kujundusest erinev (vt joonis 13). Sellises olukorras on vaja otsustada, kas hakata muutma erinevaid malle, et Drupali *HTML* märgistuskeel samasuguseks saada või muuta staatilise kujundusega kaasa tulnud *CSS* reegleid nii, et nad rakendusksid variatsoonide vormi elementidele.

Joonis 13. Drupali toote detailvaates ei ole *CSS* kood rakendunud variatsioonide vormile.

Antud olukorras otsustas autor muuta lisatud kujunduse ressursse, sest märkas *Twig debug* abil, et variatsioonide vormi väljundi muutmiseks õigele kujule, on vaja üle kirjutada mitmeid erinevaid malle. Värvi valikute ning ostukorvi lisamise nupu jaoks tuleb antud juhul muuta teeki lisatud **css** faili. Autori

arvates on mugav on taaskasutada kujunduses olevaid stiile, kuid rakendada need olemasoleva Drupali märgistuskeele peale. Mõneti on tegemist katse-eksituse meetodiga, sest tuleb proovida, millisele elemendile täpselt *CSS* stiile rakendada, et saavutada soovitud tulemus. Samuti tuleb varjata ülearused elemendid, mis Drupali poolt on genereeritud ning vajalikud osad muuta kujundusega samasuguseks. Järgnev näide sisaldab **product-detail.css** faili lõppu juurde lisatud ridasid, mille abil rakenduvad *CSS* stiilid antud vormile (vt. koodinäide 13).

```
.product-detail .variations {
  padding-bottom: 20px;
}
.product-detail .variations legend {
  font-size: 75%;
  font-weight: normal;
  line-height: 1;
  color: #777;
  text-transform: uppercase;
  margin: 0;
  border: none;
}
.product-detail .variations .field--item {
  width: 100%;
}
.product-detail .variations .form-group {
  margin-bottom: 5px;
}
.product-detail .variations .form-required:after {
  display: none;
}
.product-detail .variations .input-group-addon,
.product-detail .variations .form-radio {
  display: none;
}
.product-detail .variations .radio {
  display: inline-block;
  margin: 0;
}
.product-detail .variations label {
  color: #faebd700;
  cursor: pointer;
  margin-right: 5px;
  height: 20px;
```

```
  font-size: 10px;
  padding: 2px;
  border: 1px solid gray;
  border-radius: 2px;
  width: 25px;
}
.product-detail .variations label.color-red {
  background-color: red;
}
.product-detail .variations label.color-black {
  background-color: black;
}
.product-detail .variations label.color-blue {
  background-color: blue;
}
```

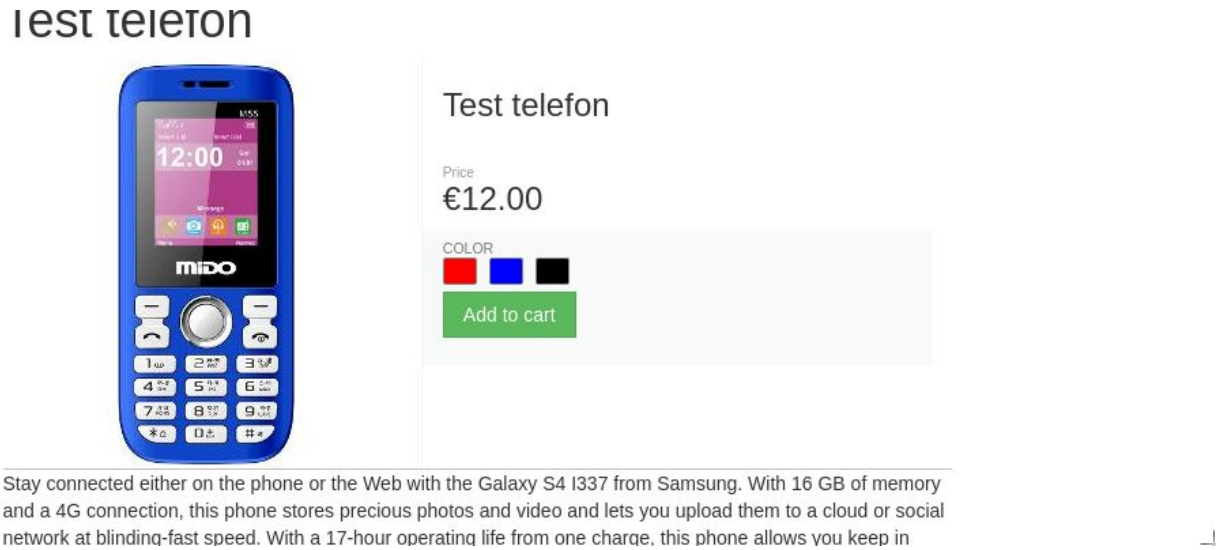
Koodinäide 13. **product-detail.css** faili lisatud read.

Peale *CSS* muudatusi on kujundus muutunud sarnasemaks, kuid nupule ei rakendu *Bootstrap* poolt tulev kujundus, sest nupu elemendilt on puudu "btn" ja "btn-success" klassid, mis staatilises kujunduses on nupu küljes. Lisaks nupule, leidis autor, et värvivalikute elementidele on vaja lisada vastavat värvi iseloomustav klass, et *CSS* reeglitega neid dünaamiliselt stiilida. Eelmainitud klasside lisamiseks tuleb on autori arvates kõige lihtsam luua uued haakfunktsioonid (vt koodinäide 14).

```
function mytheme_preprocess_form_element_label(&$variables) {
  // Add color as a class to the label for styling purposes.
  if (strpos($variables['element']['#id'], 'attributes-attribute-color')
!== false) {
    // If color is set as Red, then added class will be color-red. That
class can be targeted with CSS
    $variables['attributes']['class'][] = 'color-' .
strtolower($variables['title']['#markup']);
  }
}

function mytheme_preprocess_input__submit(&$variables) {
  // Add classes to submit button so they would use Bootstrap styles.
  $variables['attributes']['class'][] = 'btn';
  $variables['attributes']['class'][] = 'btn-success';
}
```

Koodinäide 14. **mytheme.theme** faili lisatud haakfunktsioonid, millega lisatakse värvivalikutele ning ostukorvi nupule soovitud klassid.

Esimese haakfunktsiooniga lisatakse iga värvivaliku võimaluse juurde vastav klass, näiteks väärtuse “blue” puhul lisatakse klass “color-blue”. Selle abil on võimalik kujundada iga värvivalikut nagu seda on tehtud varasema *CSS* koodinäidise lõpus (vt koodinäide 13). “color-blue”, “color-black” ning “color-red” valikud värvitakse vastavat värvi, et veebisaidi kasutaja näeks toote tooni (vt joonis 13).



Joonis 15. Toote detailvaade peale *CSS* muutmist ning haakfunktsioonide loogika loomist.

2.4 Staatilise kujunduse sidumise kokkuvõte

Näidete tulemusena annab loodud Drupali teema lehele üldise kompositsiooni, külgriba ja toote detailvaate kujunduse. Autori arvab isiklikule kogemusele põhinedes, et kahe näite puhul kasutatud meetodeid saab rakendada enamus staatiliste kujunduste lisamiseks, olenemata suurusest. Kui kasutatud näidetele otsida ning lisada veel mõned kujundused, saab võrdlemisi kiiresti tulemuseks tervenisti kujundatud veebisaidi.

Pea iga kujunduse lisamisel tekib olukord, kus ühe tulemuseni jõudmiseks on mitu võimalust. Autor soovitab sellisel juhul analüüsida võimalike lahendusi ning valida tee, mis tundub kõige aegasäästvam. Näiteks teises näites, otsustas autor asuda muutma kujunduse poolt saadud *CSS* ressursse ja kasutada haakfunktsioone, mitmete erinevate mallide üle kirjutamise asemel, Sest mallide leidmiseks, muutmiseks ja testimiseks kuluks, autori arvates, rohkem aega. Lihtsamate lahenduste puhul on võimalik need ka lihtsalt läbi proovida ning katsetada, milline kõige paremini antud juhul töötab.

Internetist leitavad kujundused pole enamasti loodud, pidades silmas konkreetselt selle kujunduse kasutajate nõudeid. Antud kujundused sisaldavad tihti komponente, mida kasutaja enda lehele ei soovi lisada. Toote detailvaate näites jättis autor samuti Drupali mallist välja need staatilise kujunduse osad,

mida antud vaates, autori arvates, vaja ei olnud. Staatiliste kujunduse algsesse lahendusse ei ole vaja pidama jääda, vaid muuta antud lahendust enda nõuetele vastavaks. Lisaks võib ka staatilistes kujundustes esineda vigu, mille parandamiseks tuleb nende lähtekoodi muuta.

Olukorras, kus vajaminevad andmed ei ole vaikimisi Drupali poolt kättesaadavad, on võimalik need ise ettevalmistada ning haakfunktsioonide abil esitluskihile edastada. Esitluskihis on võimalik *Twig* mallides antud andmeid kasutada. Toote detailvaate näites lisas autor haakfunktsioonide abil vajalikud klassid, et neid oleks võimalik eraldi *CSS* abil disainida.

Mallide otsimist lihtsustab *Twig debug* olulisel määral. Lisaks saab selle abil määratleda suurusjärgu, kui palju malle, mõne lahenduse sidumiseks Drupali teemaga, tuleb üle kirjutada. Samuti lihtsustab *Twig debug* poolt esitatud mallide informatsioon haakfunktsioonide loomist (vt koodinäide 14).

Lihtsustatult on esimeseks vajalikuks sammuks hankida staatilise kujunduse juurde kuuluvad ressursid ning laadida need vajalikul hetkel Drupali lehele. Teiseks tuleb staatilise kujunduse *HTML* märgistuskeel tõsta *Twig* mallide sisse nii, et staatilise sisu asemel kuvataks dünaamiline sisu ning kujundusega kaasas olevad ressursid töötaksid sama moodi nagu staatilise kujunduse puhul. Iga kujundus võib olla teistest pisut erinev, kuid antud kahe sammu läbimisega saab staatilised kujundused tööle Drupali tarkvara peal.

Kokkuvõte

Käesoleva töö eesmärgiks seadis autor Drupal 8 esitluskihi ning erinevate meetodite tutvustamise, mille abil saab staatilist *HTML* kujundust siduda äriloogikaga. Näitena loodi uus teema, mille baasteemaks kasutati *Bootstrap* teemat. Loodud teemas kirjutati üle lehe kujundus ning toote detailvaade. Lugejatel on võimalus järgida samu näiteid ning neid läbi proovida. Kirjeldatud võtteid saab kasutada erinevate kujunduste puhul ning nende abil saab implementeerida staatilist kujundust erinevate suurustega Drupal 8 rakendustega. Töös kasutatud näidis kujundused on avalikult kättesaadavad.

Autor valis töös kasutatud näited, sest nende sidumiseks Drupali teemaga tuli kasutada pisut erinevaid võtteid. Näidetes käsitletud võtted saab kasutada ka täiendavate kujunduste lisamisel, olenemata kujunduse mahust. Autor on antud juhtnööre kasutanud, internetist ostetud tasulise kujunduse puhul. Peale vajalike komponentide integreerimist, kohandati täiendavalt soovitud elementide kujundust. Antud meetodiga säästeti kujunduse arendamisele kuluvat aega pea poole võrra, võrreldes algusest lõpuni arendamisega.

Drupal 8 seoses laiemalt on endiselt suur eestikeelse materjali puudus ning erinevaid teemasid, millest on võimalik kirjutada, on teisigi. Ainuüksi esitlusmassiivide puhul on erinevaid kasutusvõimalusi ning funktsionaalsust piisavalt, et sellest eraldi töö kirjutada. Antud töö kõrvale luua uurimustöö, mis võrdleb erinevate populaarsete sisuhaldustarkvarade esitluskihtisid ning võimalusi, kuidas staatilisi kujundusi nende platvormidega siduda.

Autor soovitab lisaks väljatoodud näidetele proovida ning implementeerida täiendavaid kujundusi, et vilumus antud võtete kasutamisel kasvaks. Peale mõningast kasutamist muutub kujunduste lisamine järjest kiiremaks ning võimaldab katsetada erinevaid kujundusi, et leida sobiv lahendus kiiremini.

Summary

Implementing static design in Drupal 8 CMS

Bachelor's thesis

The main goal of this bachelor's thesis was to introduce Drupal 8 theme layer and methods for implementing static HTML design with Drupal's business logic. New theme based on *Bootstrap* subtheme was built as an example. The website's page layout and product's detailview were overwritten in the process. Readers can follow the same steps and examples. Described methods can be used on different designs for Drupal 8 applications of all sizes. The example designs are freely available online.

Thesis can be used indipendently or as a follow-up to previously written paper "E-poe tarkvara Drupal Commerce ülevaade ja seadistamine". Drupal 8 and e-commerce module Drupal Commerce are introduced more in depth in the aforementioned paper. Drupal 8 application which was used on current bachelor's thesis was installed and configured in the paper as well.

In author's opinion the amount of examples covered in the thesis was suffiecient for users to understand the steps for implementing a design. The same steps can be used for other designs as well. Author has used the same guidelines on a design which was bought online. Customizations were made after the implementation of required components' designs. It took approximately half of the time while developing using the methods covered in the thesis compared to building the theme from the ground up.

In addition to faster theming, the author tries to relieve the lack of resources of information in estonian. Main resource of information about Drupal 8 is currently the Drupal's own drupal.org website. Drupal's website is only in english and other resources are mainly blogs and forums. Information in estonian doesn't essentialy exist.

Drupal 8 is still lacking information in estonian and there are many topics to cover. Writing only about render arrays is enough for another thesis.

Author recommends trying implementing additional designs to gain more experience with the methods. The time it takes to implement a design will get better with practice which allows trying different designs and get to the final solution quicker.

Kasutatud kirjandus

Adding Regions to a Theme (2017) Drupal.org. loetud 01.12.2017 aadressil:
https://www.drupal.org/docs/8/theming-drupal-8/adding-regions-to-a-theme

Adding CSS and JS to a Drupal 8 theme (2017) Drupal.org. loetud 03.12.2017 aadressil:
https://www.drupal.org/docs/8/theming-drupal-8/adding-stylesheets-css-and-javascript-js-to-a-drupal-8-theme

Chaz, C. (2017) Drupal 8 Theming with Twig. Birmingham, UK: Packt Publishing.

Debugging Twig templates (2017) Drupal.org. loetud 06.12.2017 aadressil:
https://www.drupal.org/docs/8/theming/twig/debugging-twig-templates

Defining a theme with an .info.yml file (2017) Drupal.org. loetud 01.12.2017 aadressil:
https://www.drupal.org/docs/8/theming-drupal-8/defining-a-theme-with-an-infoyml-file

JavaScript API overview (2017) Drupal.org. loetud 03.12.2017 aadressil:
https://www.drupal.org/docs/8/api/javascript-api/javascript-api-overview

Render arrays (2017) Drupal.org. loetud 01.12.2017 aadressil:
https://www.drupal.org/docs/8/api/render-api/render-arrays

Stable theme (2017) Drupal.org. loetud 01.12.2017 aadressil:
https://www.drupal.org/docs/8/core/themes/stable-theme

Understanding Hooks (2016) Drupal.org. loetud 06.12.2017 aadressil:
https://www.drupal.org/docs/8/creating-custom-modules/understanding-hooks

WordPress Themes & Website Templates (kuupäev puudub) ThemeForest. loetud 28.11.2017 aadressil:
https://themeforest.net

Working with blocks (2017) Drupal.org. loetud 01.12.2017 aadressil:
https://www.drupal.org/docs/8/core/modules/block/overview

Working With Twig Templates (2017) Drupal.org. loetud 06.12.2017 aadressil:
https://www.drupal.org/docs/8/theming/twig/working-with-twig-templates