TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Christopher Gregor Toomberg 185366IADB

Contentful sisuhaldussüsteemi integreerimine uuele veebiplatvormile ettevõtte Adidas Runtastic näitel

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD
Christos Maris
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Christopher Gregor Toomberg

02.12.2021

Annotatsioon

Käesolevas töös integreeritakse ettevõtte Adidas Runtastic näitel uuele veebiplatvormile Contentful sisuhaldussüsteem. Kliendipoolse osa funktsionaalsuste arendamiseks on valitud raamistik Next.js.

Ettevõtte käesolev veebiplatvorm on aegunud ning seda kirjutatakse ümber moodsate tehnoloogiatega. Selle raames on tarvis platvormile integreerida ka uus sisuhaldussüsteem, mis võimaldaks senisest mugavamat sisu haldamise võimalust ning töövoogu kõikidele osapooltele.

Töö teoreetilises osas antakse ülevaade käesolevast veebiplatvormist, kirjeldatakse sellega kaasnevaid probleeme, eelkõige sisuhaldussüsteemi vaatepunktist, teostatakse tehnoloogiate taustauuring, valitakse analüüsi ja võrdluse tulemusena välja töö lahendamiseks sobivaimad tehnoloogiad ning arhitektuuriline lähenemine, planeeritakse teostus ning lõpuks hinnatakse töö vastavust varasemalt seatud kriteeriumitele.

Töö praktilises osas seadistatakse sisuhaldussüsteem ning arendatakse kliendipoolne rakendus koos muudatuste eelvaatamise funktsionaalsusega.

Töö tulemina valmib infrastruktuur, mille abil on sisuhaldajatel võimalik mugavalt sisu hallata ning oma muudatusi eelvaadelda.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 7 peatükki, 19 joonist, 2 tabelit.

Abstract

Integrating Contentful Content Management System into the New Web Platform on the Example of Adidas Runtastic

The aim of this work is to show how the Contentful content management system has been integrated into the new web platform on the example of Adidas Runtastic. Next.js has been chosen as the framework for implementing client-side functionalities.

The current company's web platform is outdated and is being rewritten with modern technologies. This also means that a new content management system would have to be integrated into the new web platform, that would allow for a more convenient way of content management and workflow both for developers and content managers.

In the theoretical part of the work, an overview of the current web platform will be presented as well as the problems of the current platform from the point of view of content management. Research will be conducted to choose the most suitable technologies and architectural approach in order to solve the task. The implementation will be planned and in the end the result will be assessed in accordance with the criteria set before.

In the practical part of the work, the new content management system will be set up and a client-side application developed with content previewing capabilities.

The result of the work will be an infrastructure that would allow content managers to conveniently manage content and preview changes.

The thesis is in Estonian and contains 30 pages of text, 7 chapters, 19 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakenduse programmeerimisliides
CD	*Continuous Development*, pidev arendus
CDA	*Content Delivery Application,* sisu edastamise rakendus
CDN	*Content Delivery Network,* sisuedastusvõrk
CI	*Continuous Integration,* pidev integratsioon
CMA	*Content Management Application,* sisu haldamise rakendus
CMS	*Content Management System,* sisuhaldussüsteem
DSG	*Deferred Static Generation,* osaline staatiline genereerimine
Fallback	Staatiline veebilehekülg info edastamiseks
GPX	*GPS Exchange Format*, GPS'i vahendusformaat
GraphQL	*Graph Query Language,* graaf-tüüpi pärimiskeel
ISR	*Incremental Static Regeneration,* astmeline staatiline regenereerimine
JSX	*Javascript Syntax Extension,* Javascript'i süntaksi laiendus
Markdown	Lihtteksti vormindamise süntaks
Mobile deeplink	Link, mille abil on kasutaja võimalik suunata otse mobiilirakendusse
OG	*Open Graph Protocol*, *Open Graph* protokoll
Omnichannel	Erinevat tüüpi seadetele sisu edastamiseks mõeldud tsentraalne kanal
Responsive design	Reageeriv disain; disaini printsiip, mille abil on võimalik luua ühe koodibaasi põhjal erinevatele seadmetele sobituvat disaini
REST	*Representational State Transfer,* representatiivse oleku ülekanne; tarkvaraarhitektuuri laad
SaaS	*Software as a Service,* tarkvara kasutamine teenusena
SDK	*Software Development Kit,* tarkvara arendamise komplekt
SEO	*Search Engine Optimization,* otsingumootori optimisatsioon
SPoF	*Single Point of Failure,* süsteemi nõrgim lüli
SSG	*Static State Generation,* staatiline saidigenereerimine
SSR	*Server Side Rendering,* serveripoolne genereerimine
TCX	*Training Center XML,* treeningkeskuse XML
VPN	*Virtual Private Network,* virtuaalne privaatvõrk
Webhook	Liides rakenduste omavaheliseks suhtlemiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval nõuab sisutihedate veebiplatvormide haldamine läbimõeldud lahendusi, mis peavad olema usaldusväärsed ning lihtsasti kasutatavad kõikidele osapooltele. Enamasti tähendab see liigse suhtluse eemaldamist, et tarkvaraarendajad ja sisuhaldajad saaksid kumbki oma tööle keskenduda.

Käesolevas töös demonstreeritakse, kuidas ettevõtte Adidas Runtastic näitel integreeritakse sisuhaldussüsteem uuele veebiplatvormile. Ettevõtte käesolev veebiplatvorm on aegunud ning seda kirjutatakse ümber moodsate tehnoloogiatega. Selle raames on tarvis platvormile integreerida ka uus sisuhaldussüsteem, mis võimaldaks senisest mugavamat sisu haldamise võimalust ning töövoogu kõikidele osapooltele. Senini pole kogu sisu haldamiseks kasutatud ühtegi tsentraalset sisuhaldussüsteemi.

Töö teoreetilises osas antakse ülevaade käesolevast veebiplatvormist, kirjeldatakse sellega kaasnevaid probleeme, eelkõige sisuhaldussüsteemi vaatepunktist, teostatakse tehnoloogiate taustauuring, valitakse analüüsi ja võrdluse tulemusena välja töö lahendamiseks sobivaimad tehnoloogiad ning arhitektuuriline lähenemine, planeeritakse teostus ning lõpuks hinnatakse töö vastavust varasemalt seatud kriteeriumitele.

Töö praktilises osas seadistatakse sisuhaldussüsteem ning arendatakse kliendipoolne rakendus muudatuste eelvaatamise funktsionaalsusega.

Töö tulemina valmib infrastruktuur, mille abil on sisuhaldajatel võimalik mugavalt sisu hallata ning oma muudatusi eelvaadelda.

2 Probleemi püstitus ja eesmärk

Lõputöö eesmärk on moderniseerida veebiplatvorm ning automatiseerida manuaalset tööd: integreerida sisuhaldustarkvara ning kliendipoolne kasutajaliides uuele veebilehele. Tulemusena peab arendaja looma ainult vajalikud mallid ja komponendid — uute lehtede loomine, muutmine ning avaldamine on sisuhaldaja teha.

Peamised valdkonnad, mida veebiplatvormi ümberkirjutamisega soovitakse parandada, on:

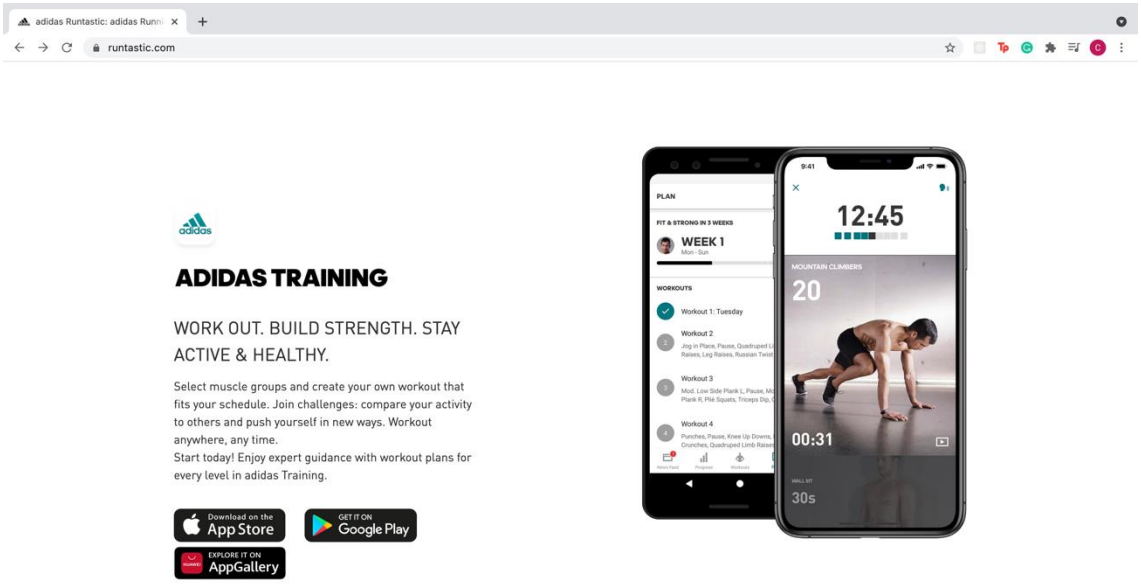
- Uus veebiplatvorm oleks väärtuslik täiendav turunduskanal, mis aitaks kaasa näiteks *premium* liikmelisuse ostudele ning suunaks rohkem külastajaid rakendusi kasutama.
- SEO (*search engine optimisation*) ja sisu jagatavus oleks parandatud.
- Turvalisus – mitmed hetkel kasutusel olevad tehnoloogiad ei toetata ametlikult juba alates 2016. aastast alates, mis tekitab mitmeid turvriske.
- Hallatavus – kogu veebiplatvormi praegune kood on monoliitne rakendus, mis muudab individuaalsete funktsionaalsuste haldamise keeruliseks, kuna on vaja omada sügavat arusaama kogu veebirakendusest.
- Vana veebiplatvormi koodi hulk on kasvanud suureks ning mitmetes kohtades leidub koodi kordust. Puudub konkreetne arusaam vana rakenduse struktuurist.

Ümberkirjutuse raames on tarvis platvormile integreerida ka uus sisuhaldussüsteem, mis võimaldaks senisest mugavamat sisu haldamise võimalust ning töövoogu kõikidele osapooltele.

Hetkel eeldab uute veebilehekülgede loomine ja muutmine manuaalset tööd arendajate poolt, mis ei ole sisult keeruline, küll aga aeganõudev ja tülikas protsess. Arendaja ja haldaja peavad iga väiksemagi muudatuse korral omavahel soovitavas töös kokku leppima ning leidma ühise aja selle muudatuse sisse viimiseks.

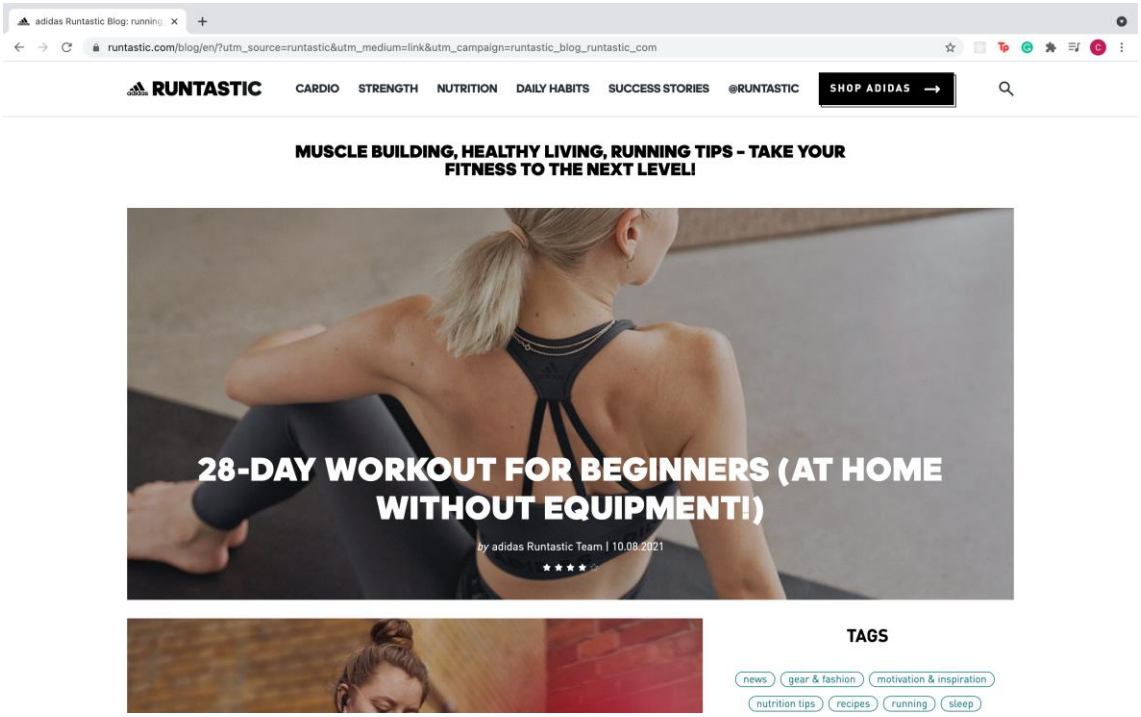
2.1 Ülevaade runtastic.com domeenist

Käesolev veebiplatvorm hõlmab endas mitmeid funktsionaalsusi. Eelkõige on see ettevõtte avaleht ning üks kanal, kuidas kliente suunata kasutama ettevõtte põhitooteid – mobiilirakendusi. Joonisel 1 on esitatud kuvatõmmis ettevõtte avalehest.



Joonis 1. Kuvatõmmis avalehest.

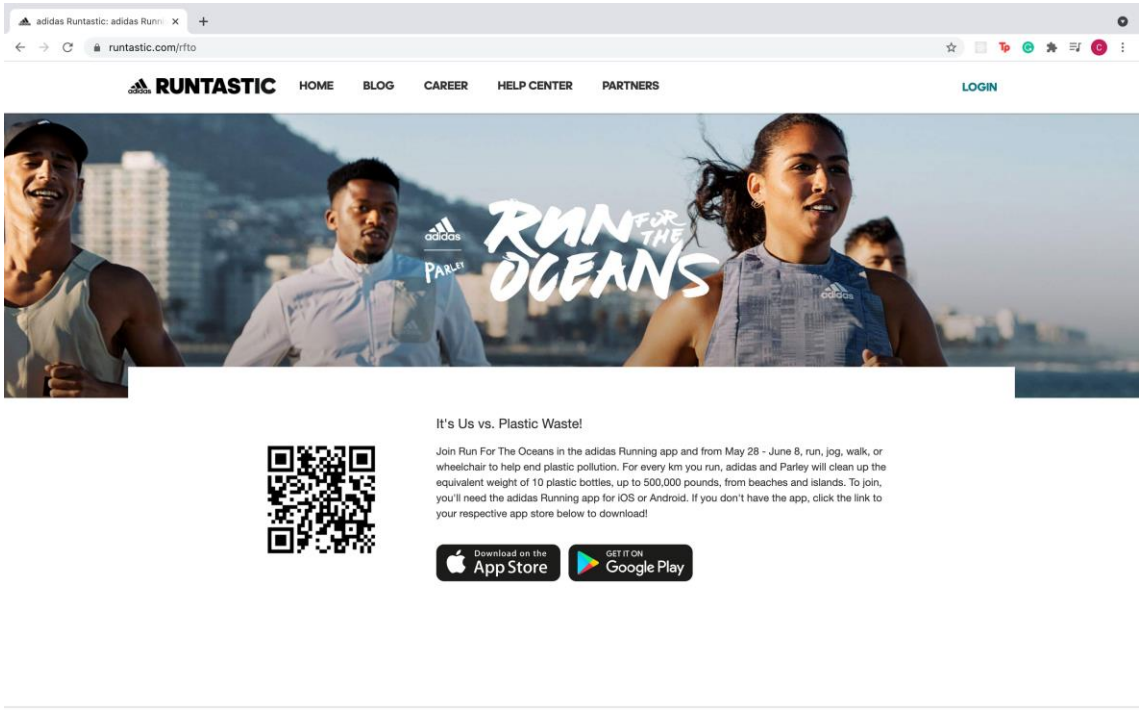
Runtastic.com domeeni all leidub blogi, kus jagatakse asjakohast informatsiooni peamiselt toitumise ja treenimise, kuid ka mentaalse ja füüsilise tervise kohta. Joonisel 2 on esitatud kuvatõmmis blogi avalehest.



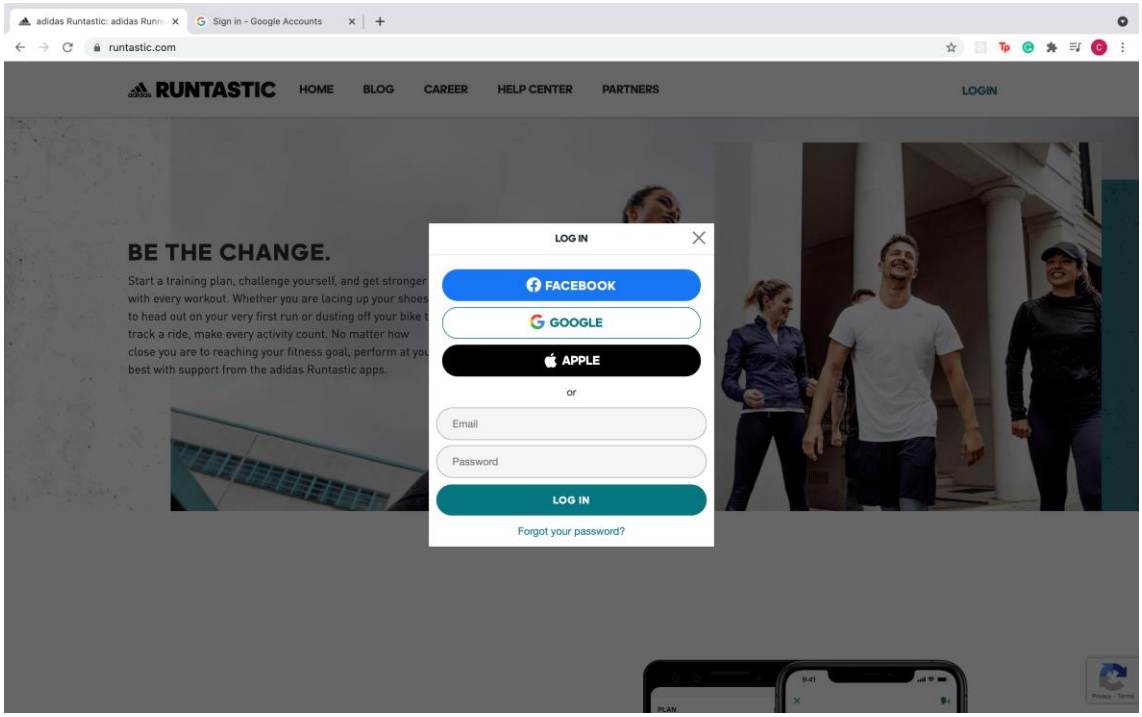
Joonis 2. Kuvatõmmis blogi avalehest.

Veebiplatvormilt leidub erinevate kampaaniate infolehekülgi, näiteks *Run For The Oceans.* Neid lehekülgi nimetatakse mitteametlikult *fallback* lehekülgedeks, mille peamine eesmärk on suunata kasutajaid mobiilirakendustesse kasutades *mobile deeplink* mustrit. Käesoleva töö üks eesmärk ongi eelnevalt kirjeldatud lehekülgede haldamise protsessi uuendamine. Joonisel 3 on esitatud kuvatõmmis ettevõtte *fallback* leheküljest.

Veebiplatvormil on kasutajal võimalik erinevate autentimisviiside (Google, Facebook, Apple) abil sisse logida ning oma põhiandmeid muuta, laadida üles sporditegevusi, kas .gpx või .tcx failina, muuta privaatsusseadeid ja hallata liikmelisuse staatust – näiteks soetada *premium* liikmelisuse tellimus. Joonisel 4 on esitatud kuvatõmmis sisselogimisvõimalustest.

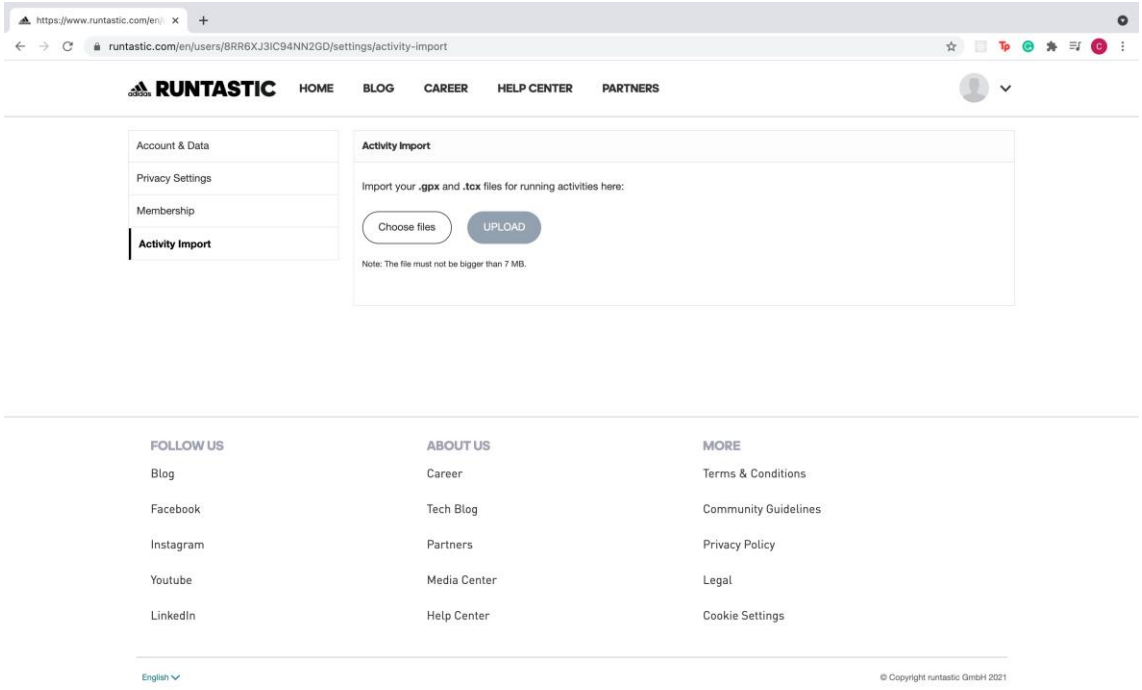


Joonis 3. Kuvatõmmis *fallback* leheküljest.



Joonis 4. Kuvatõmmis sisselogimisvõimalustest.

Joonisel 5 on esitatud kuvatõmmis kasutajale avalikest sätetest.



Joonis 5. Kuvatõmmis kasutajale avalikest toimingutest.

Hetkel on veebiplatvormi integreeritud ka administraatori paneel, mille abil on ettevõtte töötajatel võimalik hallata kõiki kasutajatega seonduvaid andmeid ja seadeid. Paralleelselt veebiplatvormi ümberkirjutamisega moderniseeritakse ja administraatori paneeli ning tulevikus on see eraldiseisev rakendus.

2.2 Käesoleva platvormi kitsaskohad sisuhaldussüsteemi vaatepunktist

Käesoleva veebiplatvormi suurim puudus on, et siiamaani pole kogu sisu haldamiseks kasutatud ühtegi tsentraalset sisuhaldussüsteemi. Seni on suurem osa tööst toimunud manuaalselt, st et iga väiksemagi muudatuse korral peavad arendaja ja sisuhaldaja omavahel kokku leppima, ühise aja leidma ning lõpuks muudatused sisse viima.

Lisaks leidub ka üldisem probleem, mida oleks mõistlik vältida – erinevat tüüpi sisuhaldussüsteemide paralleelne kasutamine, eriti kui soovitakse kasutada *headless* sisuhaldussüsteemi ja mikroarhitektuurilist lähenemist tarkvara arhitektuuri ühtluse säilitamiseks. Hetkel kasutatakse näiteks veebiplatvormi blogi haldamiseks WordPress'i, mis näib mõistliku lahendusena, kuna blogi haldamine on üks populaarsemaid WordPress'i kasutamise viise [1].

Uue sisuhaldussüsteemi integreerimise käigus on kogu sisu võimalik liigutada tsentraalsele platvormile, sisuhaldusplatvormil kasutajakogemust vastavalt haldaja eelistustele seadistada, mille käigus kaob vajadus lisa süsteemi järele.

Teine probleem, mis kaasneb antud juhul WordPress'i töös hoidmisega, on asjaolu, et paraku pole WordPress arendajate seas eelistatud platvorm [2], kuna see piirab oluliselt tehnoloogiate valikut, mida arendajad kasutada saavad. Sellest tulenevalt võib olla keeruline leida tehnoloogiaga töötatamiseks motiveeritud töötajaid. Kasutades mõnda moodsat sisu haldamise viisi, näiteks *headless* CMS'i (*Content Management System*), saab tööandja vabamalt valida tehnoloogiaid, mida ettevõttes kasutatakse, olles seeläbi atraktiivsem potentsiaalsetele töötajatele [3].

2.3 Planeeritavad muudatused

Uue veebiplatvormi arenduse käigus integreeritakse uude platvormi sisuhaldussüsteem, ja arendatakse kliendipoolne rakendus, millega saab sisu näha ning eelvaadelda. Kasutatakse Contentful raamistikku ning kliendipoolseks tehnoloogiaks on valitud Next.js.

Tulevikus hakkab kogu sisu haldamine erinevate veebiplatvormi osade jaoks toimuma läbi ühe tsentraalse platvormi. See tähendab, et näiteks nii avaleht, blogi ja *fallback* leheküljed pärivad sisu samast kohast. Käesoleva töö eesmärk on integreerida moodne lahendus sisu haldamiseks uuele veebiplatvormile ning seda demonstreerida *fallback* lehekülgede näitel.

Kõige olulisem töö puhul on sisu haldamiseks uue süsteemse lähenemise tutvustamine ning juurutamine.

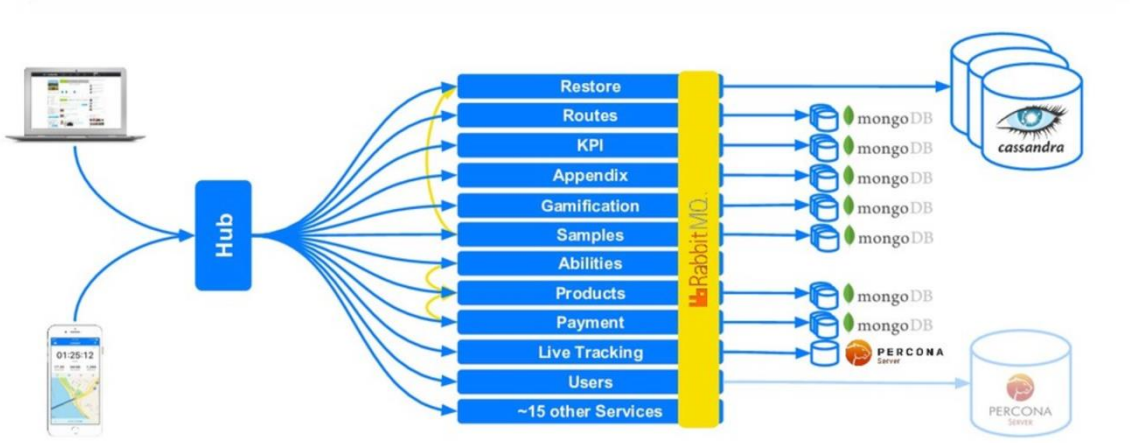
3 Arendatava rakenduse planeerimine

Kuna veebiplatvormi kirjutatakse ümber moodsate tehnoloogiatega, kehtestatakse nõuded, et tulemus oleks tehniliselt võimalikult otstarbekas ning kasutajakogemus oleks meeldiv. Uue mikroteenustel põhineva rakenduse arendamine eeldab seda ideed toetavate tehnoloogiate kasutamist.

Juba hetkel kasutatakse ettevõtte tagarakenduses mikroteenustel põhinevat arhitektuuri. Põhimõte seisneb selles, et on tsentraalne värav, mida nimetatakse Hub'iks, mille kaudu on võimalik ühenduda erinevate teenustega.

Näiteks, kui teha päring mingi kasutaja sporditegevuste saamiseks, läbib see kõigepealt kontrolli, kus verifitseeritakse sertifikaate ning tegeletakse sissetulevate päringute koormuse tasakaalustamisega. Hub'is toimub ka päringute määra piiramine. Samuti autentidakse päringu tegija ning veendutakse, et talle on üldse lubatud ressurssidele ligipääs, enne kui päring lõpuks teenusele edastatakse. Sama põhimõtte alusel toimivad kõik ülejäänud teenused [4].

Joonisel 6 kollasega märgitud RabbitMQ on sõnumibuss, mis võimaldab teenustel omavahel sünkroonselt suhelda.



Joonis 6. Ülevaade ettevõtte *back-end* arhitektuurist [4].

Uue rakenduse arendamisel tuleks lähtuda granulaarsest arhitektuurilisest disainist, mis sobituks juba olemasoleva *back-end* arhitektuuriga.

Hetkel kasutatakse Contentful *headless* CMS'i mobiilirakendustes treeningvideote ja koostööpartnerite andmete pärimiseks. Contentful API'd (*Application Programming Interface*) ei kasutata otse, vaid see on põimitud ühte mikroteenusesse. Vajadus sellise lahenduse järele tuleneb asjaolust, et mobiilirakendused kasutavad toiminguteks enamasti mitut mikroteenust samaaegselt. Seega tekib vajadus selle integreerimiseks kohandatud kujul mõnda teenusesse, et hoida tagarakenduse arhitektuuri ja liideseid ühtlasena. See annab võimaluse kasutada ettevõtte sisest kohandatud API'd ning vajadusel sisuhaldussüsteem mõne teise vastu välja vahetada.

Sarnane lähenemine oleks muidugi ideaalne uue veebiplatvormi CMS'i jaoks, kuid samas peab kaaluma, kas see oleks mõistlik ehk teisisõnu, kas sellise lahenduse järele oleks nõudlus. Kuna veebiplatvorm ei kasuta nii mitmeid mikroteenuseid samaaegselt nagu mobiilirakendused, vaid hetkel päritakse lihtsalt sisu, kas siis avaldatud või eelvaatlemis vormis, siis piisaks hetkel otse *headless* CMS'i kasutamisest.

Lisaks on *headless* CMS'i kasutamine ise juba mikroarhitektuurne lähenemine: kasutatakse väljast ostetud teenust, millega suhtlus käib API kaudu.

3.1 Töö teostamiseks sobivaim CMS tüüp

CMS on tehnoloogia, kuidas mugavalt sisu hallata ning jagada. CMS on lõdvalt defineeritud, seda on lihtsam kirjeldada kasutusjuhu kaudu. Viise, kuidas sisu haldamise protsessi on ajalooliselt käsitletud, on mitmeid. CMS'i tüüpide kirjeldusest järeldub, kuidas oma kasutusjuhtude poolest erinevad traditsiooniline ja *headless* CMS, mis on mõlema tehnoloogilise lähenemise eelised ja puudused ning milline tehnoloogia on käesoleva ülesande lahendamiseks sobivaim.

3.1.1 Traditsiooniline CMS

Ajalooliselt on sisu haldamiseks olnud populaarseim valik avatud lähtekoodiga WordPress. Ligikaudu 40% avalikest internetilehekülgedest toimivad just WordPress'i tehnoloogiat kasutades [1].

Põhjus, miks üldse tekkis vajadus sellise tehnoloogia järele, tuleneb tollasest interneti kui meediavahendi populaarsuse kasvust ning samuti ettevõtete suuruse ja keerukuse kasvust. Näiteks oli ettevõtte osakondadel vaja veebilehe eri osades samaaegselt sisu muuta ning

tihti tegelesid sellega inimesed, kellel polnud tehnilist haridust. Suurele sisu haldamise nõudlusele vastu tulles vähendati arendajatele delegeeritud monotoonset tööd süsteemsete lahenduste loomisega [5].

Traditsiooniline CMS koosneb kahest osast: CMA (*Content Management Application*) ja CDA (*Content Delivery Application*). CMA on CMS'i osa, mille abil on võimalik rollipõhise ligipääsu alusel hallata veebilehe sisu. Lisaks sisaldab CMA funktsionaalsust kujundada veebilehekülgi, ilma et töötaja peaks omama teadmisi HTML'st, CSS'st ja JavaScript'st [5].

CDA on mehhanism, mis kompileerib ja uuendab lõppkasutajale kuvatavat veebilehte sisestatud info alusel [5].

Traditsiooniline CMS sisaldab tervet veebirakendust: nii esi- kui tagarakendust. Näiteks WordPressi administraatori paneelis saab teha kõike veebilehega seonduvat: hallata sisu, kasutajaid, nende õigusi, veebilehe seadeid, domeeni, sertifikaate, installeerida erinevaid pistikprogramme.

Tulemuseks on tarkvara arhitektuuri mõistes suur monoliit. Olenevalt kasutusjuhust võib olla selle kasutamine põhjendatud, näiteks isikliku blogi või lihtsate staatiliste veebilehekülgede jaoks. WordPress'i laadse monoliitse rakenduse puhul on suurim eelis rakenduse toimimiseks vajalik minimaalne seadistamine.

Raamistiku suurima puudusena võib esile tõsta, et traditsioonilisi CMS süsteeme on keerukas isiklikele eelistustele vastavalt seadistada. Kui soovitakse kasutada mikroteenustele rajatud arhitektuuri, tahetakse kasutada vabalt valitud esirakenduse raamistikku ning ei vajata suuremat osa funktsionaalsusest, mida tavaline CMS süsteem pakub, sest võib-olla leidub see mõnes muus mikroteenuses, siis peaks lisaks eelnimetatud süsteemile kaaluma sobivaimaid alternatiive. Lisaks suurendab monoliitse sisuhaldussüsteemi kasutamine pindala, mis on avatud küberrünnakutele, suurendades nii turvariske [6].

Traditsioonilise CMS näol on tegemist laialivalguva terminiga. Seda tüüpi CMS tehnoloogiat kasutades on võimalik arendada mittetehnilisi oskusi omades terveid veebilehekülgi, mille puhul sisu haldamine on vaid üks osa protsessist.

3.1.2 Headless CMS

2010. aastast alates, mil Ethan Marcotti tutvustas terminit *responsive design*, muutus viis, kuidas tänapäeval internetti kasutatakse. Enam ei piisanud vaid tavaarvutitele vorbitud sisust. Sisu edastamise puhul tuli hakata kaaluma *omnichannel* põhimõttel arendatavaid süsteeme – tsentraalne liides, kust kõik erinevad tüüpi seadmed: arvutid, tahvelarvutid, nutitelefonid ja -kellad süsteemsel viisil andmeid pärida saavad [5].

Headless CMS on „peatu“ CMS süsteem. Sellel puudub esirakendus, mis muudab selle sisuliselt iseseisvaks mikroteenuseks, kust on võimalik andmeid pärida [7].

Tegemist on tehnoloogiaga, kust arendajal on võimalik sobivaimal viisil, näiteks REST (*Representational State Transfer*) või GraphQL (*Graph Query Language*), andmeid pärida. Sisuhaldajal on võimalik kasutada mugavat veebipõhist kasutajaliidest, kus saab hallata sisu.

Mis teeb *headless* CMS'i eriti võimsaks on asjaolu, et selline sisuhaldussüsteem on erinevate tehnoloogiate suhtes agnostiline. See tähendab, et süsteem on võimalik põimida kohandatud serverikoodi või valida meelepärane esirakendi raamistik.

Käesolevas töös soovitakse arendada mikroarhitektuuril põhinevat süsteemi ning kasutusjuhtum on väga selgelt defineeritud, mis tähendab, et ei vajata suuremat osa funktsionaalsust, mida traditsiooniline CMS pakub. *Headless* CMS on käesoleva töö jaoks sobivaim CMS tüüp: see on kõige paindlikum tehnoloogia, mis sisaldab täpselt seda, mida vaja, ning mida on alati võimalik laiendada.

3.2 *Jamstack* arhitektuur

Jamstack (JavaScript, API, *Markdown*) on üks arhitektuuriline printsiip, mida järgides on tänapäeval võimalik sisutihedaid veebirakendusi arendada. Põhiideed, millest arhitektuurivoog lähtub, on eraldatus (*decoupling*) ja veebilehekülgede eelgenereerimine (*pre-rendering)* [8].

Jamstack arhitektuuril põhinevad rakendused võib kokku võtta järgmiste punktidega [6]:

- globaalselt hajutatud ning vastupidav intensiivsele internetiliiklusele;
- arendajakogemus baseerub Git-põhisel töövool;

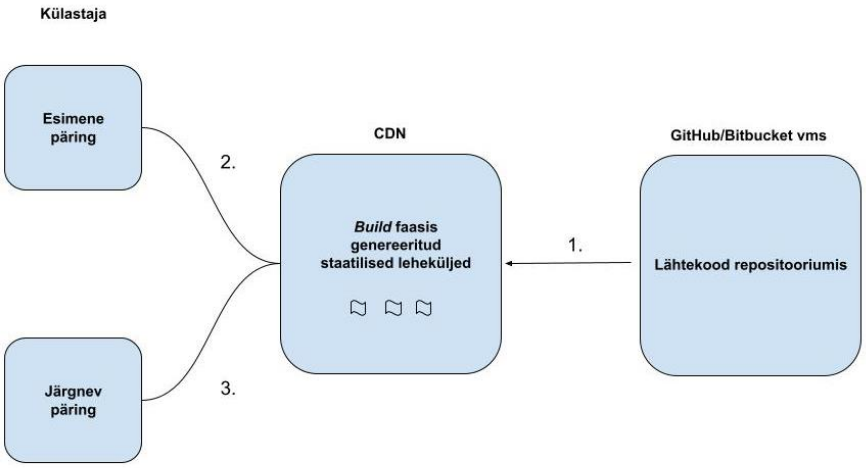
- rakendusel on modulaarne arhitektuur, mis suhtleb teiste teenustega API kaudu;
- sisu on enne avaldamist eelgenereeritud ning optimeeritud.

Tänapäeval on dünaamiliste veebirakenduste sisu kuvamise põhimõte üsna sarnane. Iga kord, kui kasutaja külastab mõnd lehekülge, päritakse andmebaasist andmed, teisendatakse need mõne malli eeskujul HTML'iks, mis külastajale tagasi saadetakse [6].

Selline lähenemine tagab küll värskeima sisu, kuid samuti kaasneb sellega alati liigseid päringuid. *Jamstack*'i üks põhiidee on, et uut sisu ei peaks tootma mitte päringu baasil vaid siis, kui sisu reaalselt muutub [6].

Tavalise *jamstack* arhitektuuril põhineva rakenduse elutsükkel toimib järgnevalt: sisu hoiustatakse sobivas kohas, näiteks *markdown* failidena repositooriumis; käesolevas töös päritakse andmed *headless* CMS'st. Muudatuste avaldamise korral genereeritakse vajaliku sisuga staatilised leheküljed, mis avaldatakse CDN's (*Content Delivery Network*), kus genereeritud sisu asub lõpp-kasutajale füüsiliselt lähimas asukohas, mis tagab veebirakenduse lühikese reageerimisaja [6]. Joonisel 7 on kujutatud *jamstack*-tüüpi rakenduse elutsüklit, kust numbritega on tähistatud olulised sündmused:

1. Automaatne CI/CD (*Continuous Integration/Continuous Development*) töövoog, kus uued leheküljed genereeritakse näiteks CMS'is muudatuse avaldamisel.
2. Esimesel päringul laaditakse eelgenereeritud lehekülg CDN'st alla külastajale füüsiliselt lähimast sõlmpunktist.
3. Järgnevate päringute puhul kuvatakse veebilehekülg puhvrist, et tagada võimalikult kiire reageerimisaeg.



Joonis 7. Ülevaade *jamstack*-tüüpi rakenduse elutsüklist.

Ehkki CDN'i kasutamine, mis on *jamstack* arhitektuuril põhinevate rakenduste üks alustala, pole oma sisult midagi uut, on uudne kiirus ning mugavus, millega andmeid CDN's avaldada [6]. Samamoodi pole *jamstack*'i defineerinud suurkorporatsioon, sellel puuduvad konkreetsed standardid ning selle kasutamine ei eelda nimest tulenevate tehnoloogiate ja lähenemiste kasutamist. Tegemist kogukonna tasandil defineeritud parimate tavade kogumiga [6].

Käesolevas töös loob *jamstack* arhitektuurimustrite järgimine raamistiku sisutiheda rakenduse arendamiseks.

3.3 Funktsionaalsed ja mittefunktsionaalsed nõuded

Rakenduse arendamiseks kirjeldatakse funktsionaalsed ja mittefunktsionaalsed nõuded.

3.3.1 Funktsionaalsed nõuded

Käesoleva rakenduse arendamisel on peamised nõuded esitatud sisuhaldaja vaatepunktist *fallback* lehekülgede haldamiseks.

1. Sisuhaldaja saab sisuhaldussüsteemi abil sisu hallata: sisu lisada, muuta ja kustutada.
2. Sisuhaldaja saab tehtud muudatusi avaldada.
3. Sisuhaldaja saab muudatusi eelvaadelda neid avaldamata.

4. Sisuhaldaja saab eelvaadet jagada isikuga, kellel on eelvaatele ligipääs.

5. Sisuhaldajal on võimalik läbi sisuhaldussüsteemi hallata lokalisatsiooni.

3.3.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalseid nõudeid kirjeldatakse FURPS mudeli abil [9], kus:

- U – *usability*, kasutatavus;
- R – *reliability*, usaldusväärsus;
- P – *performance*, jõudlus;
- S – *supportability*, toevõime;

FURPS mudeli funktsionaalsed nõuded on kirjeldatud eelnevas alapeatükis.

Kasutatavus

Rakendus peab olema arendatud viisil, et sisuhaldajal on võimalik mugavalt sisuhaldussüsteemi kaudu sisu hallata ning seda vajadusel avaldada. Lisaks peab tal olema võimalik sujuvalt avada eelvaade uusimate muudatustega ning eelvaatele suunavat linki jagada.

Usaldusväärsus

Kui rakendust arendada eelnimetatud *jamstack* põhimõtetel, siis peaks üleüldine rakenduse usaldusväärsus ja turvalisus olema käesoleva rakenduse jaoks sobilik.

Usaldusväärsus tähendab, et arendatud süsteem on töökindel ning seda on võimalik ööpäevaringselt kasutada. CDN'i kasutamine tähendab ka, et SPoF'de (*Single Point of Failure*) arvu minimaliseeritakse. Enam ei pea ettevõte toetuma ainult oma serveritele, et sisu hoiustada. Kui mõni sõlmkoht CDN võrgus peaks rikki minema, toetab võrguliiklust mõni muu sõlmkoht ning sisu edastamine pole takistatud [6]. Samuti mõistetakse usaldusväärsuse all seda, et eelvaate lingid genereeritakse alati ennustatava mustri abil, mis tagab, et lingi jagamise puhul näevad kõik osapooled sama staatusega sisu.

Turvalisuse vaatepunktist saab rakendusest olema kaks versiooni: pre-produktsioon ja produktsioon. Pre-produktsiooni versioonis on eelvaate režiim lubatud. Tulevikus on kõik

pre-produktsiooni versiooni aadressid ettevõtte sisese VPN'ga (*Virtual Private Network*) kaitstud. Produktsiooni versiooni näeb lõppkasutaja, see on avalikult kättesaadav ettevõtte domeeni alt ning selles versioonis puudub ligipääs eelvaatele.

Ühtlasi tähendab modulaarse arhitektuuri kasutamine ja veebilehtede eelgenereerimine seda, et turvariskidele avatud pindala vähendatakse. Kui võrrelda turvariske, mis kaasnevad monoliitse CMS'i, näiteks WordPress'i kasutamisega, kus kõik funktsionaalsused on põimitud ühte ning, kus rünnakutele avatud pind on kordades suurem, siis lehekülgede eelgenereerimisega saavutatakse ainult lugemisõigustega süsteem, millega juba eos välditakse haavatavust mitmetele riskidele [6].

Jõudlus

Arendatud rakenduses hoiustatakse eelgenereeritud lehekülgi CDN's, mis tagab, et sisu asub lõppkasutajale füüsiliselt võimalikult lähedal. Eelgenereerimise seisneb selles, et vajalikud leheküljed toodetakse ning avaldatakse ainult siis, kui sisu muutub, mitte iga kord, kui külastaja neid pärib. Selline lähenemine vähendab liigseid päringuid ning koormust ettevõtte infrastruktuurile.

Toevõime

Kuna sisuhaldussüsteem on eraldi mikroteenus, siis on selle integreerimine olemasolevasse süsteemi üsna kerge. Samuti on seda lihtsam testida kui mõnda monoliitset rakendust. Arendatav rakendus peab toetama sisu tõlkimist erinevatesse keeltesse. Lokalisatsiooni ja tõlgete haldamiseks peavad kasutajatel olema võimalikult piiritletud õigused, mis siiski võimaldavad ülesannet täita. Näiteks tõlkija peab saama sisestada tõlget ainult enda hallatavasse lokaati.

3.4 *Headless* CMS süsteemi valik

Headless CMS valimisel arvestatakse, kas tehnoloogial on piisavalt funktsionaalsusi, et rakendust arendada. Valitud sisuhaldussüsteem peab võimaldama: sisumudeli loomist, rolli-põhist kasutajate haldamist, lokalisatsiooni haldamise võimalust, meediafailide hoiustamist, tuge *webhook*'dele ja sisu eelvaatamise võimalust.

Lisaks peab sisuhalduskeskkonna kasutuselevõtt olema võimalikult kiire. Võimalusel üritatakse vältida olukorda, kus CMS rakendust peab enne avaldamist kohandama ning

tegelema rakenduse hoiustamisega. Üritatakse leida võimalikult valmis kujul süsteem, mis täidaks kõiki eelnevalt nimetatud nõudeid.

Võrdluseks on valitud kolm *headless* CMS süsteemi: Strapi, Netlify ja Contentful. Leidub arvukalt [10] teisi *headless* süsteeme, mille kõigi omavahel võrdlemine ei oleks teostatav. Autoril ja ettevõttes on kogemust Contentful raamistikuga, mida soovitakse võrrelda kahe populaarse vabavaralise tehnoloogiaga.

3.4.1 Strapi CMS

Strapi on hetkel populaarseim avatud lähtekoodiga *headless* CMS [10]. Strapi sisaldab palju funktsionaalsusi ning võimaldab sisuhaldajale mõeldud administraatori paneeli väga detailselt kohandada.

Strapi leiduvad kõik funktsionaalsused, mida käesoleva rakenduse arendamiseks on vaja ning palju enamat [11]:

- tugi erinevatele andmebaasidele (SQLite, MongoDB, MySQL, Postgres);
- sobiliku andmemudeli loomine graafilise kasutajaliidese abil;
- API, mida on võimalik kohandada ning pärida andmeid kas RESTful või GraphQL protokolli abil;
- raamistik on täielikult JavaScript põhine;
- tugi *webhooki*'dele;
- automaatselt genereeritud dokumentatsioon;
- kasutajate ning nende õiguste haldamine rolli-põhise autentimisskeemi abil;
- lokalisatsiooni haldamine;
- meediafailide hoiustamine ja haldamine.

Järeldub, et raamistik on väga detailselt kohandatav ning arendaja-sõbralik. Tegemist on JavaScript rakendusega, mida võib kasutada muutmata kujul või siis oma vajadustele kohandada.

Tarkvara kasutamine on tasuta, kui rakendust majutada isiklikus serveris, kuid ettevõte pakub ka tasulist automatiseeritud majutusteenust.

Strapi täidab vajalike funktsionaalsuste kriteeriumid ning pakub suurt kohandatavust, kuid samuti arvestatakse, kas kõik funktsionaalsused on vajalikud. Kui kriteeriumiks on *headless* CMS süsteem võimalikult kiiresti üles seada, peaks kaaluma ka alternatiivseid võimalusi.

3.4.2 Netlify CMS

Netlify on samuti avatud lähtekoodiga *headless* CMS rakendus. Tegemist on React raamistikul põhineva rakendusega, mis on ehitatud ümber Git-põhise töövoo. Võrreldes teiste tehnoloogiatega on Netlify CMS'l kõige väiksem hulk funktsionaalsusi.

Netlify's leidub kasutajasõbralik administraatori paneel, on võimalik luua sobivaid sisumudeleid ning hoiustada meediafaile, kuid puudub sisse ehitatud võimalus näiteks rolli-põhiseks kasutajate haldamiseks, mis on käesoleva rakenduse puhul lokalisatsiooni haldamiseks oluline [12].

3.4.3 Contentful CMS

Contentful on võrreldavatest raamistikest ainuke kinnise lähtekoodiga. See tähendab, et rakendust on saadaval ainult SaaS (*Software as a Service*) kujul ning selle kohandamine oma vajadustele on keerulisem, kuid mitte võimatu. Sisuhaldussüsteemi on võimalik kohe veebirakendusena kasutama hakata, ilma et oleks vaja täiendavat seadistamist. Contentful pakub võimalust arendada administraatori paneeli isikustamiseks rakendusi, mida on võimalik Contentful'i platvormi põimida [13]. Käesoleva rakenduse arendamiseks seda funktsionaalsust ei kasutata, kuna Contentful pakutavad sisse ehitatud funktsionaalsused on piisavad.

Lisaks on ettevõttel olemas litsents Contentful'i kasutamiseks. Contentful'i kasutatakse näiteks mobiilirakendustes treeningvideote ja koostööpartnerite andmete haldamiseks. Kuna rakendus on juba ettevõttes kasutusel, on lisaks ametlikule dokumentatsioonile olemas suurem teadmine rakenduse kasutamise kohta.

Hetkel täidab Contentful kõiki eelnevalt kirjeldatud nõudeid [13].

3.4.4 Sobivaima süsteemi valimine

Tehnoloogiate võrdlemiseks koostatakse tabel (Tabel 1).

Tabel 1. Headless sisuhaldussüsteemide võrdlus.

	Strapi	**Netlify CMS**	**Contentful**
Litsents	MIT Expat litsents	MIT litsents	Ei ole vabavara, suletud lähtekoodiga
Sisumudeli haldamine	Toetatud	Toetatud	Toetatud
Rolli-põhine kasutajate haldamine	Toetatud	Ei toetata otse rakendusest, kuid kuna töövoog põhineb Git'il, on seal võimalik kasutajatele õigusi määrata.	Toetatud
Lokalisatsiooni haldamine	Toetatud	Toetatud	Toetatud
Meediafailide hoiustamine	Toetatud	Toetatud	Toetatud
Tugi *webhook*'dele	Toetatud	Pole otseselt toetatud, kuid kuna rakendus baseerub Git versioonihaldussüsteemil, mille kaudu ka muudatused avaldatakse, puudub vajadus eraldi *webhook*'de järele.	Toetatud
Sisu eelvaatamine	Toetatud	Toetatud	Toetatud
Võimalused rakenduse isikupärastamiseks	Võimalus rakendust detailselt kohandada, kuna rakendus on avatud lähtekoodiga	Võimalus rakendust detailselt kohandada, kuna rakendus on avatud lähtekoodiga	Võimalik, kuid mitte eriti mugav, kuna tegemist on suletud lähtekoodiga.
Rakenduse kasutuselevõtt	Võimalus kasutada tasulist majutusteenust või ise lähtekoodi hoiustada, mille puhul on algne seadistamine vajalik.	Lähtekoodi peab ise hoiustama, algne seadistamine on vajalik.	Võimalus kasutada ainult SaaS teenust, mistõttu puudub vajadus rakendust algselt seadistada.

Eelnevat võrdlust arvestades valitakse *headless* CMS tehnoloogiaks Contentful, kuna see täidab kõiki esitatud nõudeid ning selle ülesseadmisaeg on minimaalne, võimaldades

keskenduda kliendipoolse rakenduse arendamisele. Lisaks leidub ettevõttes kogemus tehnoloogiaga töötamisel.

3.5 Kliendipoolse tehnoloogia valik

Kliendipoolse rakenduse all mõeldakse tehnoloogiat, millega uus veebiplatvorm arendatakse. Valitud tehnoloogia peab võimaldama kanda vana veebiplatvormi funktsionaalsused üle uuele platvormile ning sisuhaldussüsteemi aspektist vaadatuna võimaldama sisu eelvaatamist.

Rakenduse arendamiseks valitakse sobiv kliendipoolne tehnoloogia, mis peaks olema staatiline saidigeneraator ehk toetama lehekülgede eelgenereerimist ning ettevõtte siseste nõuete kohaselt peaks põhinema React raamistikul.

Omavahel võrreldakse kahte tehnoloogiat: Next.js ja Gatsby.

Ajalooliselt on Gatsby olnud kindel staatilise saidigeneraatori valik, Next.js'le lisandus eelgenereerimise funktsionaalsus hiljem. Staatilisi saidigeneraatoreid leidub veel, näiteks on väga populaarsed Hugo ja Jekyll, kuid nad on mõeldud pigem spetsiifilisteks kasutujuhtudeks, näiteks blogi, ning seetõttu on nende funktsionaalsus võrreldes Next.js'ga ja Gatsby'ga piiratum. Näiteks eeldab nende kasutuselevõtt enamasti, et sisu hoiustatakse *markdown* failidena lähtekoodiga samas asukohas, mis ei ole sisuhaldajale hea kasutajakogemus.

Hugo ja Jekyll pakuvad malle, et tarkvara oleks võimalik mugav kasutusele võtta. See tähendab, et on kindel süntaks mallide kasutamiseks või, et *markdown* formaadis failid teisendatakse mallimootori abil HTML'ks. Võrreldes seda näiteks React'i JSX (*Javascript Syntax Extension*) süntaksiga seab selline lähenemine käesoleva rakenduse puhul rohkelt piiranguid.

3.5.1 Next.js

Next.js on veebirakenduste arendamiseks mõeldud moodne raamistik, mis põhineb React tehnoloogial. Võrreldes tavalist React'i ja Next.js'i, on viimases palju funktsionaalsusi, mis kergendavad arendaja tööd. Neist on käesoleva töö raames olulisimad SSG (*Static Site Generation*), dünaamilised aadressid ja lokalisatsiooni haldamine [14].

SSG võimaldab leheküljed enne rakenduse avaldamist *build* faasis valmis genereerida ning CDN's avaldada, et tagada võimalikult kiire reageerimisaeg. Next.js raamistikus on selle funktsionaalsuse kasutamiseks olemas meetodid *getStaticProps* ja *getStaticPaths*. Nende meetoditega on võimalik vastavalt rakenduse *build* faasis pärida vajalikke andmeid ning eelgenereerida aadresse.

Nendesse meetoditesse võib kirjutada otse serveripoolset koodi või päringuid andmebaasist ning sellisel juhul neid lõpprakendusse ei lisata, vaid kasutatakse lihtsalt andmete pärimiseks. Teisalt saab neid meetodeid vajadusel täiendada, et näiteks eelvaatamise puhul meetodit käivitataks mitte ainult rakenduse *build* faasis, mis on käesolevas töös oluline sisu eelvaatamise puhul [15].

Next.js raamistikus on kaustapõhine navigeerimisstruktuur. See tähendab, et iga fail vastab mingile kindlale aadressile. Lisaks võimaldab see dünaamiliste aadresside loomist. Kõik sellised failid asuvad *pages* kaustas. Näiteks luues lehe *pages/about.js*, siis sellele saab navigeerida */abou*t alamaadressilt. Kui luua fail asukohta *pages/posts/[id].js*, siis *[id].js* viitab dünaamilisele osale, mille arendaja saab ise valida; see võib näiteks olla postituse identifikaator, mille järgi saab kindlat postitust pärida.

Dünaamiline osa võib olla ka *catch-all*. Sellisel juhul on faili nimi *[…slug].js*, kus *slug* võib asendada sobiva terminiga. Selline dünaamiline aadress asendab kõik alamaadressid. Näiteks *pages/[…slug].js* vastab aadressidele */foo/bar/1*, */foob/bar*, */foo* jne. Käesolevas töös on dünaamiliste aadresside kasutamine vajalik, et koodi kordust vältides arendada vajalikud veebilehed ning nende eelvaated [16].

3.5.2 Gatsby

Gatsby ja Next.js on omavahel sarnased raamistikud. Gatsby staatilise eelgenereerimise funktsionaalsus on vanem, kuid hetkel on mõlemal raamistikul üsna sarnased funktsionaalsused. Planeeritavat rakendust saaks arendada mõlema raamistiku abil.

Nagu Next.js raamisik, toetab ka Gatsby staatilist eelgenereerimist (SSG), osalist staatilist eelgenereerimist (DSG, *Deferred Static Generation*), serveripoolset genereerimist (SSR, *Server Side Rendering*) ja kliendipoolset genereerimist. Kõiki viise tuleks kasutada sobivaimateks kasutusjuhtudeks, käesoleva veebirakenduse loomisel on olulisim

staatiline eelgenereerimine. Samuti on Gatsby's sarnane kaustapõhine navigeerimisstruktuur [17].

Gatsby raamistikus peab arendaja andmete pärimiseks kasutama raamistiku andmekihti, mis eeldab GraphQL pärimiskeele kasutamist [18]. Andmekihti on võimalik lisada adaptereid, et suhelda ühe või enama andmeallikaga. Käesoleva ülesande lahendamiseks GraphQL keelt vaja ei lähe ja autoril puudub selle tehnoloogia kasutamise kogemus.

Lisaks nõuab staatilise genereerimise kasutuselevõtt Gatsby raamistikus suuremat seadistamist. Next.js raamistikus on võimalik iga komponendi või lehe juurde lisada kindel meetod, mis vajalikku ülesannet täidab.

3.5.3 Sobivaima süsteemi valimine

Tehnoloogiate võrdlemiseks koostatakse tabel (Tabel 2).

Tabel 2. Kliendpoolsete tehnoloogiate võrdlus.

	Next.js	**Gatsby**
Sisu genereerimise võimalused	SSG, ISR (*Incremental Static Generation*), SSR, kliendipoolne genereerimine	SSG, DSG, SSR, kliendipoolne genereerimine
Tugi lokalisatsiooni haldamiseks	Toetatud	Toetatud
Andmete pärimise võimalus	RESTful, GraphQL	GraphQL
Dünaamilised aadressid	Toetatud	Toetatud

Võttes arvesse eelnevat võrdlust, valitakse rakenduse arendamiseks Next.js raamistik.

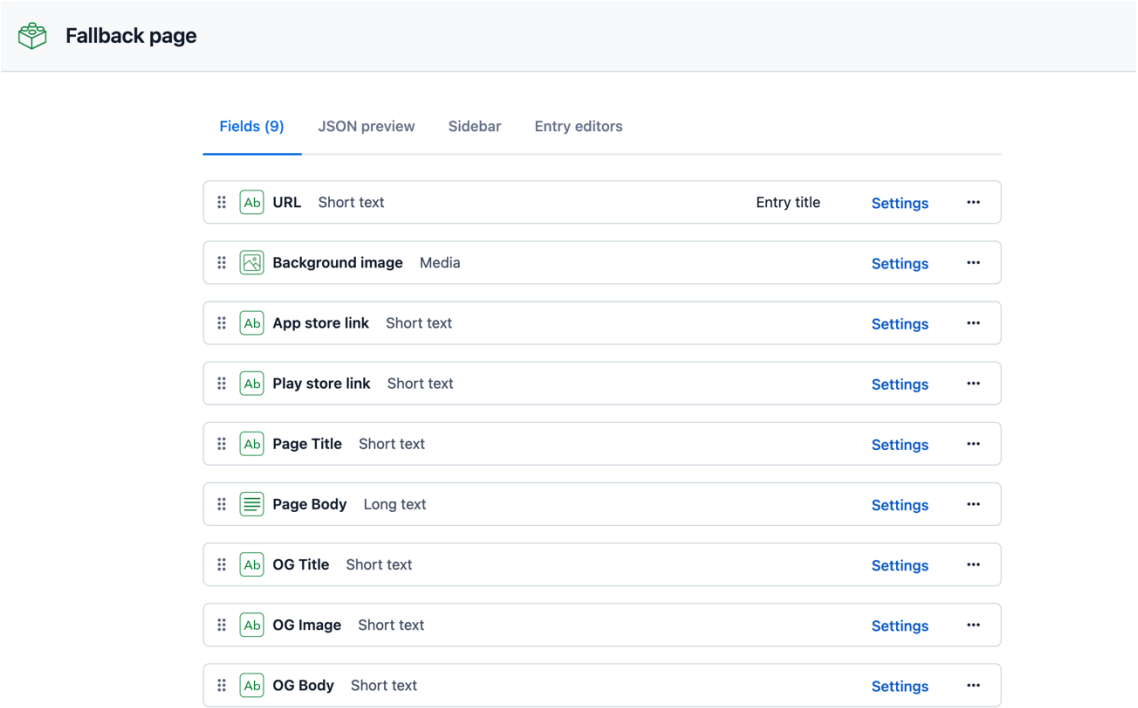
4 Töö teostus

Töö teostamine koosneb kahest etapist: Contentful CMS seadistamine ja kliendipoolse rakenduse arendamine. Sisuhaldussüsteemi seadistamisel luuakse vajalikud andmemudelid, seatakse üles eelvaadete lingid, seadistatakse lokaadid ja *webhook*'id,

Kliendipoolse rakenduse arendamisel luuakse *fallback* lehekülje mall ja funktsionaalsus info pärimiseks avaliku ja eelvaate režiimi jaoks.

4.1 Contentful CMS seadistus

Kõigepealt luuakse Contentful administraatori paneelis vajalik andmemudel *fallback* lehekülgede andmete hoiustamiseks valideerimisreeglitega (Joonis 8). Andmemudel koosneb 9 väljast. Väljades sisaldub info, mida kuvatakse mallis otse külastajale ja ka info, mida otse näha pole, kuid mida kasutatakse SEO jaoks. Selliseid väljad on näiteks OG (*Open Graph*) *Title*, OG *Image* ja OG *Description*. Need võimaldavad esitada veebilehe kohta struktureeritud kujul infot sotsiaalvõrgustikes.

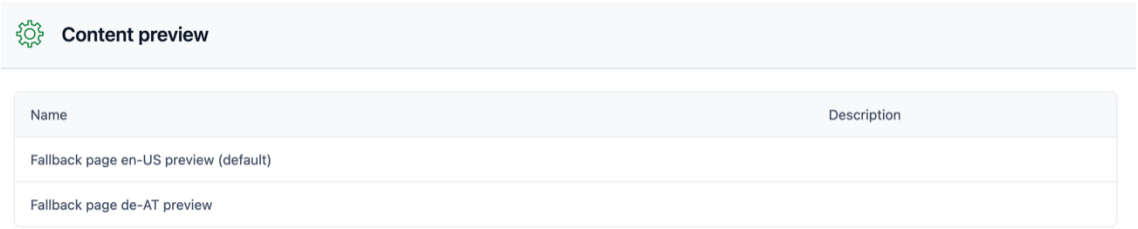


Joonis 8. Andmemudeli loomine.

Lisatakse lokalisatsioon vajalikele väljadele. Hetkel toetatakse kahte lokaati: USA inglise keel, mis on ka vaikelokaat ning Austria saksa keel (Joonis 9). Tõlkijatele määratakse kindlad rollid, mille tulemusena saavad nad hallata ainult lokaate, millele neil on ligipääs.

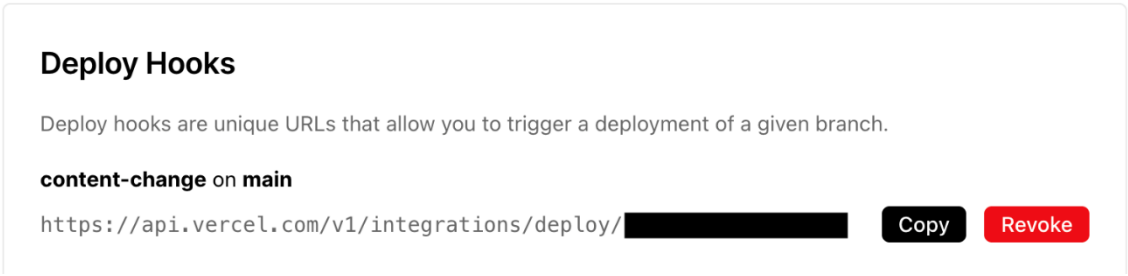
Joonis 9. Ülevaade lokaatidest.

Kuna sisuhaldajatel peab olema võimalus mugavalt tehtud muudatusi eelvaadelda kõikides lokaatides, mida toetatakse, seatakse üles eelvaate lingid mõlema lokaadi jaoks (Joonis 10).



Joonis 10. Ülevaade eelvaadetest.

Käesolevas töös majutatakse arendatud rakendust Vercel teenuses, et demonstreerida, kuidas uue sisu avaldamisega eelgenereeritakse vajalike muudatustega leheküljed, mida hoiustatakse CDN's. Vercel platvormis seatakse üles *webhook*'id (Joonis 11), mille poole päringu tegemisel käivitatakse *build* protsess.



Joonis 11.*Webhook* Vercel rakenduses.

Joonisel 12 on kujutatud *webhook*'ide lisamist Contentful rakendusse. URL lahtris asuvale aadressile tehakse päring, kui Contentful veebirakenduses muudatused avaldada.

See päring käivitab Vercel teenuses *build* protsessi, mille tulemusena staatilised leheküljed eelgenereeritakse ning avaldatakse CDN's.

Joonis 12. *Webhook* Contentful rakenduses.

4.2 Kliendipoolse rakenduse arendus

Next.js raamistikus luuakse mall *fallback* lehekülgedele ja nende eelvaadetele. Lehekülg luuakse kujul */pages/[…slug].js*. Lingid sellele leheküljele hakkavad näiteks välja nägema kujul */de-AT/rfto/preview*.

Esimesel positsioonil on lokaat. Kui lokaat puudub, on tegemist vaikelokaadiga. Teisel positsioonil on lehekülje identifikaator, mille abil kindla lehekülje andmeid on võimalik pärida; see on ainuke nõutud väli. Viimasel positsioonil olev väärtus tähistab, kas soovitakse näha lehekülje avalikku versiooni või eelvaadet.

Next.js raamistikku on sisse ehitatud viis, kuidas lehekülgede eelvaadet teostada [19], kuid käesolevas töös ei kasutata seda kahel põhjusel:

1. Oleks tarvis kirjutada tükk serveripoolset koodi, mis autendib päringu arendaja poolt määratud salasõna baasil. See tähendab aga, et salasõna peaks sedasi saatma päringuga krüpteerimata kujul, mis kujutab endas turvariske. Teoreetiliselt oleks tavakülastajal võimalik saada ligipääs eelvaates olevatele andmetele.

2. Sisuhaldajal puuduks võimalus brauseri URL ribal asuvat aadressi otse kopeerida ning jagada, et eelvaadet jagada, kuna peale päringu autentimist suunatakse külastaja eelvaate leheküljele, mille aadress on sama, mis avalikus versioonis. Õigusi eelvaate vaatamiseks kontrollitakse automaatselt seatud küpsiste abil. Sisuhaldajate väljaõpe oleks keerulisem ning tekiks rohkem arusaamatusi.

Fallback lehekülje kood tegeleb päringu tuvastamisega, see tähendab, kas soovitakse pärida avalikku või eelvaate versiooni.

Päringu aadressist tuvastatakse lokaat, identifikaator ja eelvaate märge. Kui tegemist on eelvaatega, lülitatakse sisse Next.js funktsionaalsus ISR, mis võimaldab peale *build* faasi andmete muutumise korral uusi andmeid pärida (Joonis 13) [20].

```
export async function getStaticProps(context) {
    const isPreview = context.params.slug[context.params.slug.length - 1] === 
'preview' && context.params.slug.length > 1;
    const slug = context.params.slug[0];
    const entries = await getPage(isPreview, `${slug}FallbackPage`, 
context.locale);

    if (isPreview) {
        return {
            props: {
                entry: entries.items[0],
                locale: context.locale
            },
            revalidate: 1
        };
    }

    return {
        props: {
            entry: entries.items[0],
            locale: context.locale
        },
    };
}
```

Joonis 13. *getStaticProps* programmikood.

Kasutatakse Contentful JavaScript SDK'd (*Software Development Kit*). Joonisel 14 kujutatakse kahe liidese loomist vastavalt eelvaate ja avalike andmete pärimiseks.

```
import { createClient } from 'contentful';

const client = createClient({
    space: process.env.CF_SPACE_ID,
    accessToken: process.env.CF_DELIVERY_ACCESS_TOKEN,
});

const previewClient = createClient({
    space: process.env.CF_SPACE_ID,
    accessToken: process.env.CF_PREVIEW_ACCESS_TOKEN,
    host: 'preview.contentful.com'
});

const getClient = (preview) => preview ? previewClient : client;

export async function getPage(preview, contentType, locale) {
    return await getClient(preview).getEntries({ content_type: contentType,
locale: locale });
}
```

Joonis 14. Andmete pärimise funktsionaalsus Contentful SDK abil.

Joonisel 15 on kujutatud aadresside eelgenereerimist.

```
export async function getStaticPaths() {
    return {
        paths: [
            {
                params: { slug: ['rfto'] },
                params: { slug: ['en-US', 'rfto'] },
                params: { slug: ['de-AT', 'rfto'] }
            },
        ],
        fallback: 'blocking'
    };
}
```

Joonis 15. Aadresside eelgenereerimine getStaticPaths meetodi abil.

Rakendusse lisatakse märge, mille abil on võimalik eelvaate versiooni kas sisse või välja lülitada.

Kui eelvaate režiim on välja lülitatud (avalikus versioonis), suunatakse kõik eelvaadete aadressidele tulevad päringud ümber avalikult kättesaadavatele aadressidele. Selle jaoks lisatakse Next.js konfiguratsioonifaili next.config.js *redirects* meetod.

Lisatakse toetus pildidomeenidele ja lokalisatsioonile.

Joonisel 16 on kujutatud Next.js konfiguratsioonifaili tehtud muudatusi.

```
module.exports = {
  async redirects() {
    if (!!+process.env.ENABLE_PREVIEW) {
      console.log('Preview enabled');
      return [];
    }
    console.log('Preview disabled');
    return [
      {
        source: '/:slug*/preview',
        destination: '/:slug*',
        permanent: false
      },
    ];
  },
  images: {
    domains: ['images.ctfassets.net', 'd1ki59phkeobjj.cloudfront.net'],
  },
  i18n: {
    locales: ['en-US', 'de-AT'],
    defaultLocale: 'en-US'
  },
  reactStrictMode: true,
}
```

Joonis 16. Next.js konfiguratsioonifaili sisu.

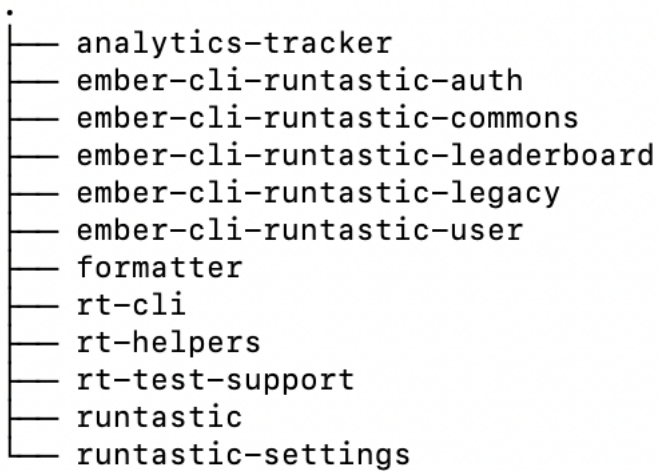
5 Tulemused ja võimalikud edasiarendused

Käesoleva töö raames juurutatakse süsteemne lähenemine, kuidas Contentful *headless* CMS'i abil andmeid hallata ja eelvaadelda, mida demonstreeritakse *fallback* lehekülgede näitel. Arendatud rakendus on kättesaadav aadressilt https://poc-deploy-test.vercel.app/rfto.

Töös mitte kajastatavate sammudena saab lisada, et uut senini arendatud rakendust majutatakse juba ettevõtte enda infrastruktuuris, mis on hetkel ainult VPN'i kaudu kättesaadav. Seda võib lugeda suureks edasiarenguks vana veebiplatvormi väljavahetamise suunas.

Võrreldes vana ja uue rakenduse üldist arhitektuuri järeldub, et uus rakendus on kordades lihtsama arhitektuuriga.

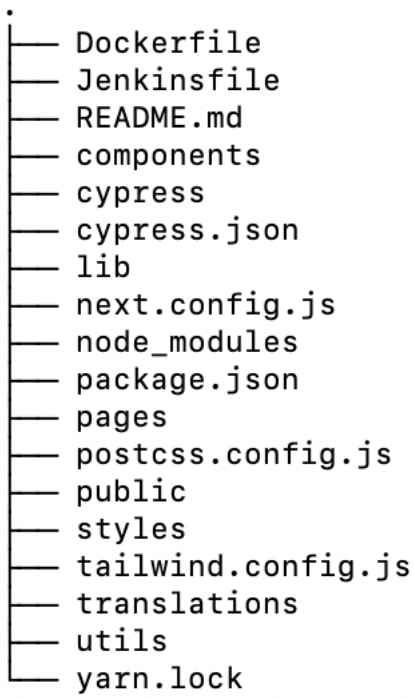
Joonisel 17 kujutatud vana rakenduse arhitektuurist on näha, et veebiplatvorm toimib kolme põhiprojekti koostööl: *ember-cli-runtastic-legacy*, mis on vana veebiplatvormi baasprojekt, *ember-cli-runtastic-auth*, mis vastutab kõikide autentimisteenuste eest ja *runtastic-settings*, projekt, mille abil kasutajal on võimalik oma seadeid hallata. Lisaks kasutatakse vajadusel teiste projektide funktsionaalsusi, kuid eelnimetatud kolme projekti majutatakse veebiplatvormi toimimiseks ettevõtte infrastruktuuris.



Joonis 17. Vana veebiplatvormi ülesehitus.

Uus veebiplatvorm on oma arhitektuurilt minimalistlikum, kuid sisaldab kõike toimimiseks vajalikku: lehekülgi, komponente, teenuseid, meediafaile, üldiseid ja stiili

konfigureerimisfaile, testimist ja majutamiseks vajalikku seadistust. Joonisel 18 on kujutatud uue rakenduse ülesehitust.

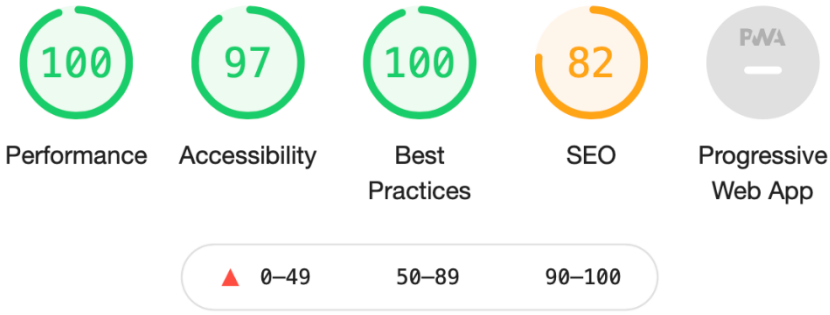


Joonis 18. Uue veebiplatvormi ülesehitus.

Et saada esmane ülevaade lehekülje toimimisest, teostatakse Google Lighthouse analüüs. (Joonis 19). Analüüsist järeldub, et arendatud rakenduse jõudlust, ligipääsetavust ja parimate tavade kasutamist hinnatakse kõrgelt.

Rakendus on arendatud reageeriva disaini printsiipidest lähtuvalt, ligipääsetavuse puhul on arvestatud põhiliste vajadustega, näiteks, et HTML pildielemendil on alati *alt* atribuut ja välditakse CSS'is *display:none* stiilireeglit, mis elemente ekraanilugejate eest peidavad [21]. Rakendus on arendatud Next.js dokumentatsiooni järgides, mis tagab kaitse levinuimate turvaaukude eest.

Analüüsist järeldub ka, et SEO parandamiseks soovitatakse näiteks *robots.txt* faili ning *meta tag*'de lisamist. Arendatud rakenduse puhul kasutatakse OG protokolli *meta tag*'e, mis võimaldavad lehekülgi sotsiaalvõrgustikes struktureeritud kujul esitada.



Joonis 19. Ülevaade Google Lighthouse raportist.

Tulevikus hakkavad kõik uue veebiplatvormi osad sisu pärima tsentraalsest kanalist, hetkel demonstreeritakse sellise lähenemise potentsiaali *fallback* lehekülgede põhjal. Võimalike edasiarendustena tõstetakse esile:

1. Ettevõtte sisese VPN'i kasutamine, et pre-produktsiooni versiooni rakenduse eelvaate lehed oleksid ainult ettevõtte siseselt kättesaadavad.

2. Eraldi *backend* koodi kirjutamine, mis võimaldaks CMS veel sujuvamalt käesolevasse infrastruktuuri põimida ning vajadusel välja vahetada.

Ehkki sisuhaldussüsteemi integreerimine on valminud rätseptööna Adidas Runtastic ettevõttele, saab arendatud rakenduses kasutatud põhimõtteid kasutada ära ka teiste analoogsete projektide puhul.

Sisuhaldussüsteemina on kasutatud *headless* CMS tehnoloogiat, mis on eraldiseisev mikroteenus ning erinevate tehnoloogiate suhtes agnostiline, võimaldades seda kasutada paindlikult kohandatud kasutusjuhtudeks. Uue veebiplatvormi sisu haldamise funktsionaalsus on arendatud *jamstack* põhimõtetel, muutes selle turvaliseks, optimeerituks ja resistentseks intensiivse võrguliikluse suhtes.

Kui mõnes teises firmas on vajalik integreerida sisu haldamise lahendus koos eelvaatlemise funktsionaalsusega, on võimalik kasutada käesolevas töös leiduvat lahendust, kus, kasutades ühte koodibaasi, on arendatud sisu avalikud ja eelvaate režiimi mallid. Tulevikus on sellise rakenduse haldamine koodi hulga tõttu lihtsam. Lisaks on eelvaadetele võimalik omistada eraldi aadressid, muutes need sisuhaldajate omavahelise suhtluse tarbeks kergesti jagatavateks. Eelvaate aadresse on võimalik kaitsta, kasutades näiteks firmasisest VPN'i, tagades avaldamata sisu privaatsuse.

6 Kokkuvõte

Käesoleva töö eesmärgiks on uue veebiplatvormi arendus moodsate tehnoloogiatega ning sinna uue sisuhaldussüsteemi integreerimine.

Töö raames integreeritakse sisuhaldussüsteem Contentful uuele veebiplatvormile ning selle kasutamist demonstreeritakse *fallback* lehekülgede põhjal. Kliendipoolseks tehnoloogiaks valitakse Next.js.

Töös antakse ülevaade käesolevast veebiplatvormist ning tuvastatakse sisu haldamisega seotud probleemid. Seejärel liigutakse edasi arendatava rakenduse planeerimise juurde. Valitakse välja, millist tüüpi CMS oleks käesolevaks kasutusjuhuks sobivaim. Lisaks valitakse välja arhitektuuriline lähenemine, mis sobituks ettevõtte käesoleva arhitektuuriga ning oleks asjakohane ülesande lahendamiseks. Kasutatakse mikroarhitektuurilist lähenemist ja *jamstack* arhitektuuri.

Analüüsi osas valitakse võrdluse tulemusena välja sobivaim *headless* CMS ja kliendipoolne tehnoloogia. Omavahel võrreldakse kolme *headless* sisuhaldussüsteemi: Strapi, Netlify CMS ja Contentful. Neist esimesed kaks on vabavaralised tehnoloogiad, mida soovitakse võrrelda suletud lähtekoodiga Contentful'ga, millega on autoril varasemalt kogemust.

Kliendipoolse tehnoloogia valimisel võrreldakse omavahel kahte React tehnoloogial põhinevat JavaScript raamistikku: Gatsby ja Next.js, millel mõlemal on staatilise saidigeneraatori omadused.

Töös eesmärgid on veebiplatvormi moderniseerimine ning sisuhaldaja ja arendaja töö lihtsustamine. Töös algselt sõnastatud eesmärgid on täidetud: uue veebiplatvormi arendamiseks on kasutusele võetud moodne raamistik ja *fallback* lehekülgede haldamise protsessi on lihtsustatud. Uus raamistik on turvalisem, pakub sisu haldamiseks optimeeritud lahendusi ning on vana platvormiga võrreldes lihtsama ülesehitusega. Nüüd peab arendaja looma vaid vajalikud mallid ning sisuhaldaja saab sisu haldamisega tegeleda Contentful keskkonnas.

Arendatud rakendus on valminud vastavalt ettevõtte nõuetele, kuid üldist funktsionaalsust saab ära kasutada ka teiste sarnaste ülesannete lahendamiseks. *Headles* CMS tehnoloogia

on eraldiseisev ning teiste tehnoloogiate suhtes agnostiline lähenemine, mis muudab selle väga paindlikuks tehnoloogiaks. Avalikud ja eelvaate mallid on loodud kasutades ühte koodibaasi, mis lihtsustab veebirakenduse haldamist.

Edasiarendustena peaks tulevikus rakenduse pre-produktsiooni versioon olema turvatud ettevõtte sisese VPN'ga ja Contentful API'ga suhtlemiseks vajalik kood võib olla põimitud eraldi mikroteenusesse, selle asemel, et seda otse kasutada.

Hetkel ei ole uus veebiplatvorm kasutusel primaarse platvormina, kuid sinna lisandub pidevalt uusi funktsionaalsusi, mis viitab peatse platvormi vahetuse suunas. Uut lähenemist väärtustatakse ning nähakse selle eeliseid vana ees.

7 Kasutatud kirjandus

[1] M. Osman, „Wild and Interesting WordPress Statistics and Facts (2021),“ Kinsta, 21 9 2021. [Võrgumaterjal]. Loetud aadressil: https://kinsta.com/blog/wordpress-statistics/. [Kasutatud 21 10 2021].

[2] Jamstack, „Jamming into the Mainstream: Jamstack Community Survey 2021,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://jamstack.org/survey/2021/#demographics. [Kasutatud 22 10 2021].

[3] Contentful, „Why your CMS is driving away top talent,“ [Võrgumaterjal]. Loetud aadressil: https://assets.ctfassets.net/fo9twyrwpveg/619p0PxxBKEQcMSi8kKSoc/3eb42d3d540ef3f20bf635697cf86aa2/WhitePaper_Losing_Talent__1_.pdf . [Kasutatud 22 10 2021].

[4] S. Lasselsberger, „Evolution of the Runtastic Backend, 2018,“ 4 5 2018. [Võrgumaterjal]. Loetud aadressil: https://www.youtube.com/watch?v=iEFTwtFa07Q. [Kasutatud 23 10 2021].

[5] B. Heslop, „History of Content Management Systems and Rise of Headless CMS,“ Contentstack, 8 12 2018. [Võrgumaterjal]. Loetud aadressil: https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms/. [Kasutatud 23 10 2021].

[6] M. Biilmann ja P. Hawksworth, „Modern Web Development on the Jamstack,“ 2019. [Võrgumaterjal]. Loetud aadressil: https://www.netlify.com/pdf/oreilly-modern-web-development-on-the-jamstack.pdf. [Kasutatud 23 10 2021].

[7] J. Fraser, „Headless, decoupled and Contentful: A non-technical explanation for the confused,“ Contentful, 4 2 2019. [Võrgumaterjal]. Loetud aadressil: https://www.contentful.com/blog/2019/02/04/difference-between-headless-decoupled-contentful/. [Kasutatud 24 10 2021].

[8] Jamstack, „What is Jamstack?,“ Jamstack, [Võrgumaterjal]. Loetud aadressil: https://jamstack.org/what-is-jamstack/. [Kasutatud 22 10 2021].

[9] P. Eeles, „Capturing Architectural Requirements,“ IBM, 15 11 2005. [Võrgumaterjal]. Loetud aadressil: https://web.archive.org/web/20201112020231/http://www.ibm.com/developerworks/rational/library/4706.html%22%20%5Cl%20%22N100A7. [Kasutatud 23 10 2021].

[10] Jamstack, „Headless CMS, A List of Content Management Systems for Jamstack Sites,“ Jamstack, [Võrgumaterjal]. Loetud aadressil: https://jamstack.org/headless-cms/. [Kasutatud 24 10 2021].

[11] Strapi, „Features,“ Strapi, [Võrgumaterjal]. Loetud aadressil: https://strapi.io/features. [Kasutatud 24 10 2021].

[12] Netlify CMS, „Overview,“ Netlify, [Võrgumaterjal]. Loetud aadressil: https://www.netlifycms.org/docs/intro/. [Kasutatud 22 10 2021].

[13] Contentful, „Features,“ Contenful, [Võrgumaterjal]. Loetud aadressil: https://www.contentful.com/features/. [Kasutatud 21 10 2021].

[14] Vercel, „Next.js: The React Framework,“ [Võrgumaterjal]. Loetud aadressil: https://nextjs.org/learn/basics/create-nextjs-app. [Kasutatud 30 10 2021].

[15] Vercel, „ Data Fetching,“ [Võrgumaterjal]. Loetud aadressil: https://nextjs.org/docs/basic-features/data-fetching. [Kasutatud 30 10 2021].

[16] Vercel, „Routing,“ [Võrgumaterjal]. Loetud aadressil: https://nextjs.org/docs/routing/introduction. [Kasutatud 30 10 2021].

[17] Gatsby, „Rendering Options,“ [Võrgumaterjal]. Loetud aadressil: https://www.gatsbyjs.com/docs/conceptual/rendering-options/. [Kasutatud 2 11 2021].

[18] Gatsby, „Part 4: Query for Data with GraphQL,“ [Võrgumaterjal]. Loetud aadressil: https://www.gatsbyjs.com/docs/tutorial/part-4/. [Kasutatud 2 11 2021].

[19] Vercel, „Preview Mode,“ [Võrgumaterjal]. Loetud aadressil: https://nextjs.org/docs/advanced-features/preview-mode. [Kasutatud 28 11 2021].

[20] Vercel, „Incremental Static Regeneration,“ [Võrgumaterjal]. Loetud aadressil: https://vercel.com/docs/concepts/next.js/incremental-static-regeneration. [Kasutatud 4 11 2021].

[21] Mozilla Developer Network, „display,“ Mozilla Developer Network, 13 8 2021. [Võrgumaterjal]. Loetud aadressil: https://developer.mozilla.org/en-US/docs/Web/CSS/display. [Kasutatud 2 12 2021].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Christopher Gregor Toomberg

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Contentful sisuhaldussüsteemi integreerimine uuele veebiplatvormile ettevõtte Adidas Runtastic näitel“, mille juhendaja on Kristiina Hakk PhD.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

02.12.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.