TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Valeria Dmitrijeva 155194IAPB

MULTISEADME VEEBIRAKENDUSE DISAIN, REALISEERIMINE JA TESTIMINE

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad

Tallinn 2018

Autori deklaratsioon

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: Valeria Dmitrijeva

12.05.2018

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on leida universaalne tehnoloogia, mille rakendamine tagab veebirakenduse korrektse kuvamise võimalikult suures arvus seadmetes.

Töö eesmärk täidetakse näidisveebirakendusega, mis on olemasoleva veebirakenduse uus versioon. Tulemuseks on valmis veebilehekülg ja veebirakendus, mis on välja arendatud, kasutades MEAN täis-pinu tehnoloogiat ja Bootstrap 4 raamistikku.

Arenduse tulemusest lähtuvalt määrati tehnika, et rakendada seda püsitatud eemärgi saavutamiseks.

Lõputöö on kirjutatud eesti keeles ning selles on 33 lehekülge teksti, 6 peatükki, 25 joonist ning 1 tabel.

Abstract

Multi-device web-application design, development and testing

The aim of the thesys is to find a universal technology, whose implementation ensures, that the web application is correctly displayed on as many devices as possible.

The purpose of the work is fulfilled by implementing a sample web application, that is a new version of the existing web application. Result of the research is MEAN full-stack web-application designed with the Bootstrap 4 framework.

Based on the results of the development, there was developed the technique, that supports the thesis theme.

The thesis is in Estonian and contains 33 pages of text, 6 chapters, 25 figures, 1 tables.

Lühendite ja mõistete sõnastik

3-tiered architecture	Kolmekihiline arhitektuur [1]
AJAX	Veebiarenduse tehnika Asynchronous JavaScript And XML [2]
Apache	OSi-litsentsiga populaarseim (enam kui pooltel veebisaitidel kasutatav) veebiserveri tarkvara, eeskätt UNIXipõhistele operatsioonisüsteemidele [1]
API	Rakendusliides – reeglid ja vahendid, mida rakendusprogramm kasutab suhtluseks operatsioonisüsteemiga, andmebaasihalduse süsteemiga või muu juhtprogrammiga, samuti sideprotokolliga [1]
Black-box	Objekti (programmi, süsteemi) käitumise testimine musta kasti meetodil, ümbritseva maailma vaatepunktist ning objekti sisemust arvestamata ja selle tundmist nõudmata; strateegia lähtub nõuete spetsifikatsioonist [1]
CRUD	Püsiliku andmestu, eeskätt püsiliku andmebaasi põhioperatsioonid: *Create*: loomine *Read/Retrieve*: lugemine/võtt *Update*: värskendus (ajakohastus) *Delete/Destroy*: kustutus/hävitus [1]
Endpoint	Otspunkt [1] – teenuse URL
Front-end	Eessüsteem – inimkasutajat või kasutavat süsteemi tagaosaga liidestav [1]
Grey-box	Halltestimine – objekti välisest käitumisest ja sisestruktuuri osalisest teadmisest lähtuv käsitlusviis objekti uurimisel, modelleerimisel, testimisel [1]
Happy path	Edukas kasutusjuhu täitumine
HTTP	Veebi aluseks olev hüpermeediumile suunatud standardne rakenduskihi protokoll, mis määrab sõnumite vormingu ja edastuse ning veebiserverite ja brauserite käitumise [1]
JavaScript	HTML-lähtekoodiga interakteeruv objektorienteeritud programmeerimiskeel, mida kasutatakse dünaamilise veebisisu (aktiivsisu) skriptide loomiseks [1]
JSON	JavaScripti alamhulgal põhinev lihtne inimlugemiseks ja kirjutuseks hõlbus andmevahetusvorming [1]

LAMP	Täis-pinu veebiarendustehnooloogia, koosneb Linux-ist, Apache-ist, MySQL-ist, PHP/Perl-ist
Lightweight	Muudest omataolistest suhteliselt lihtsam, väiksem või kiirem programm, protokoll, seade [1]
MEAN	Täis-pinu veebiarendustehnooloogia, koosneb MongoDB-ist, ExpressJS-ist, AngularJS-ist, NodeJS-ist
MVC	Arhitektuuri muster
MySQL	Relatsioonbaasihaldur [1]
Nginx	Veebiserveri tarkvara
PHP	Skriptikeel [1]
Python	Paljusid Interneti protokolle toetav universaalne kõrgkeel [1]
REST	Nüüdisaegse veebi nõuetele sobivat arhitektuuristiili ja ühtset liidest taotlev hajustöötluse, eriti veebiteenuste programmeerimise paradigma [1]
Template	Mall – mingis ettemääratud vormingus ja osaliselt koostatud dokument [1]
URI	Internetis resursi identifitseerimiseks või nimetamiseks kasutatav laiendatav märgistring [1]
White-box	Objekti sisestruktuuri ja -loogika testimine valge kasti meetodil, sh tarkvara puhul arvestades kasutatud programmeerimiskeelt [1]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Järgnev peatükk tutvustab bakalaureuse töö teema.

1.1 Taust ja probleem

Selleks et kasutajate konversioon oleks võimalikult kõrge, peab veebirakendus olema mugav ja korrektne, sõltumata kasutatavast seadmest. Vastasel juhul hakkab kasutaja otsima alternatiivi. Kasutajate seadmed, mis toetavad veebirakendusi, on väga erinevad: väikesed ja suured ekraanid, hiire või puuteplaadiga juhitavad, mobiilsed ja statsionaarsed, püst- või põikformaatiga. Seega peab rakenduse arendamisel arvestama paljude erinevate seadmetega. Arendaja peab valima sobiva tehnika ja vahendid, mis ei ole tavaline, kuna neid on väga palju. Käesolevas töös pakutakse nimetatud probleemi lahenduseks universaalne meetod.

Bakalaureusetöö objektiks on http://innospark-ict.eu/innosparktool/ lehel asuv veebirakendus, mis on plaanis ümber teha. Suurbritaanias tehtud veebiportaali testimise käigus avastati, et see ei ole adapteeritud mitmetele seadmetele ning realiseerimisel oli kasutatud vananenud tehnoloogiat. Seega sobib kõnealune rakendus käesoleva lõputöö teema illustreerimiseks.

Olemasoleval veebilehel olev tutvustus ütleb, et:

- antud veebileht ja rakendus on rahvusvahelise projekti osa;
- projekti eesmärk on kaasata uusi andeid, õpetada Euroopa kodanikele loomingulisi oskusi ja selle kaudu tõsta tootlikkust ja majanduskasvu Euroopa Liidus;
- projekti rahastatakse Erasmus+ programmi vahenditest;
- projektis osalevad TTÜ ja organisatsioonid Suurbritaaniast, Itaaliast, Bulgaariast, Ungarist ning Hispaaniast.

TTÜ-poolsed vastutavad isikud on Jekaterina Tšukrejeva, Jaak Tepandi ja Gunnar Piho. Projekti raames on tehtud veebiportaal, kus asub informatsioon projekti, osalejate ja läbi

viidud uuringute kohta, samuti loovuse test, mis oli koostatud projekti osalejatena. Testi tulemuseks on nõuanded selle kohta, millist loovuse osa tuleb arendada.

1.2 Ülesande püstitus

Bakalaureusetöö eesmärgid:

1. Testida olemaolevat veebirakendust ja selle alusel määrata probleemid ja nõuded, mis toetavad lõputöö teemat.
2. Leida realiseerimiseks tehnoloogia ja vahendid.
3. Arendada olemasoleva rakenduse uut versiooni püstitatud eesmärgi piires.
4. Analüüsida tulemuse sobivust.
5. Lähtudes tulemustest, sõnastada tehnoloogia, mille rakendamine tagab veebirakenduse korrektse kuvamise erinevates seadmetes.

1.3 Metoodika

Käesoleva lõputöö eesmärgi saavutamine hõlmab nii teoreetilist kui praktilist tööd. Teoreetilises osas on analüüsitud kirjalikke materjale ja tehtud järeldused. Praktiline osa on teostatud materjali põhjal tehtud järelduste alusel. Järeldusi on rakendatud näidisrakenduse arendamisel. Probleemid, nõuded ja tulemus on kontrollitud, kasutades visuaalse testimise meetodit, ja sõnastatud läbiviidud arenduse alusel.

2 Olemasoleva veebirakenduse analüüs

Selles peatükis analüüsitakse olemasoleva veebilehe ja veebirakenduse lahendust ning selgitakse välja probleemid.

2.1 Testimine

Veebirakenduse edukus sõltub sujuvast funktsionaalsusest [3]. Madal kvaliteet põhjustab seda, et kasutaja ei külasta enam saiti. Kvaliteeti mõjutavad probleemid on vigased kujutised, CGI-veateated, katkised lingid jne [4]. Veebilehe külastajad ei läbi veebirakenduse kasutamise kursust ja toimivad intuitiivselt [4]. Seega ei pea rakendus nõudma spetsiaalset õppimist, vaid olema selge ja lihtne kasutada. Testimise eesmärk on hinnata veebirakenduse kvaliteeti. Ilma testimiseta on võimatu alati garanteerida, et rakendus vastab kõigile nõuetele [5].

Testimise käigus antakse süsteemile sisend ja kontrollitakse, kas tulemus vastab ootustele. Sisendi all mõistakse kasutaja tegevuse simuleerimist.

2.1.1 Testimise metoodika valimine

Kasutaja tegevust võib simuleerida automatiseeritud testidega või valideerida tulemus käsitsi. Automatiseeritud testimise käigus korduvad tegevused täituvad skriptidega, aga algustestid nõuavad siiski käsitsi testimist. Käsitsi testimine on sobiv, kui testija soovib hinnata visuaalsete komponentide paigutust ja kasutajaliidest, samas kui tegevused on ühekordsed [7]. Antud rakenduse puhul sobib käsitsi testimine, kuna plaanitakse hinnata kasutajaliidest, testimine on ühekordne ja siin ei ole skriptide kirjutamist tarvis.

Metoodika rakendamisel tehakse valik *black-box, white-box* ja *grey-box* testimise vahel. Esimesel juhul ei ole programmi kood ja struktuur teada. Teisel juhul on rakenduse struktuur ja kood täielikult kättesaadavad [6]. Mõlema meetodi kombinatsiooni nimetatakse *grey-box* testimiseks, mis tähendab, et kood ja komponentide struktuur on osaliselt avatud. Käesoleva rakenduse puhul rakendatakse *grey-box* meetodit, kuna rakenduse sisemine struktuur ja serveripoolne osa ei ole nähtavad.

Kasutusjuhtude põhiliseks testimiseks defineeritakse kasutaja *happy path* – tema tegevuste edukas stsenaarium [8]. Eduka stsenaariumi järgi saavutab kasutaja eesmärgi [9]. Antud juhul vaadeldakse kasutusjuhtu, kus kasutaja soovib läbida loovuse testi – siseneb pealehele, leiab otselingi testile, teeb testi ära ja saab kätte tulemuse. Eesmärgiks on saada eeldatav tulemus, sõltumata kasutatud seadmetest.

Antud juhul tähendab eeldatav tulemus identset funktsionaalsust ja sisu loetavust võimalikult suures arvus seadmetes. Kuna rakenduse kuvamise eest vastutab peamiselt brauser, testitakse brauseritevahelist ühilduvust ja veebilehe paindlikkust.

Arendajad peavad võrdlema veebilehe kujundust erinevates brauserites käsitsi või poolautomaatsete tööriistade abil, sest veebilehe kujundus võib erineda, sõltuvalt brauseri versioonist [10]. Brauseritevahelise ühilduvuse testimine kontrollib, et rakenduse funktsionaalsus ja sisu säilivad täielikult kõigis testitavates brauserites [11].

Lisaks võib veebileht isegi samas brauseris näha välja erinevalt, sõltuvalt ekraani suurusest ja formaadist. Paindlik veebidisain muudab liidese mugavaks ja suurendab kasutatavust [13].

Paindliku disaini eelised:

- meeldiv kasutajakogemus;
- parem indekseerimine otsingumootorites;
- bäiksemad kulud ühe paindliku disaini arendamisel, võrreldes arendamisega iga seadme jaoks [14].

Kokkuvõttes on testimise

- meetod – *grey-box* käsitsi testimine;
- kasutusjuht – kasutaja soovib läbida loovuse testi;
- eeldatav tulemus – *happy path* sõltumata kasutatud seadmest;
- aspektid – brauseritevaheline ühilduvus ja veebilehe paindlikkus.

Järgnevalt esitatakse määratud kasutusjuhu testimise protsess ja tulemused.

2.1.2 Brauseritevahelise ühilduvuse testimise protsess ja tulemused

Vastavalt käesoleva töö teemale määratakse ära, mida tähendab võimalikult suur seadmete arv: leitakse need brauserid, mis on kõige populaarsemad ja järelikult kasutuses enamikus seadmetes. Selleks võetakse suure portaali http://w3counter.com populaarsemate brauserite statistika aastal 2018; tulemused on kujutatud joonisel 3 [12].

Top 10 Web Browsers		
1	Chrome 65	29.36%
2	Safari 11	11.08%
3	Chrome 66	5.13%
4	Firefox 59	3.70%
5	IE 11	2.81%
6	Chrome 56	2.43%
7	Chrome 49	2.41%
8	UC 11	2.29%
9	Chrome 64	2.20%
10	Safari 10	2.18%

Joonis 1. Kõige populaarsemad brauserid vastavalt w3counter.com reitingule.

Järgmiseks kontrollitakse veebilehe ja sellel asuva veebirakenduse korrektset kuvamist populaarsetes veebilehitsejates: Chrome (49, 56, 64, 65, 66), IE 11, Safari 10, Firefox 59.

Testimine viidi läbi, kasutades veebiteenuse http://crossbrowsertesting.com, mis pakub nii automaattestimise võimalust kui ka brauserite emuleerimist manuaalse režiimi kasutamiseks.

Käsitsi testimise käigus anti veebiteenuse sisendiks kaks ressurssi: pealeht http://innospark-ict.eu ja veebirakendus http://innospark-ict.eu/innosparktool. Väljundiks oli mõlema ressursi hetktõmmiste kogum. Võrreldes saadud kujutisi, on näha, et visuaalselt on nii rakendus kui ka veebileht kõigis brauserites ühesugused. Järelikult on kuvamine kõigis brauserites korrektne.

Peale ekraanitõmmiste võrdlemist toetab testimisportaal brauserite emuleerimist, mille abil viidi läbi manuaal-funktsionaalne testimine; simuleerides brauserite versioone

ühekaupa, oli korratud kasutusjuhu stsenaarium. Tulemuseks oli tõdemus, et kõik vajalikud funktsioonid toimivad ootuspäraselt.

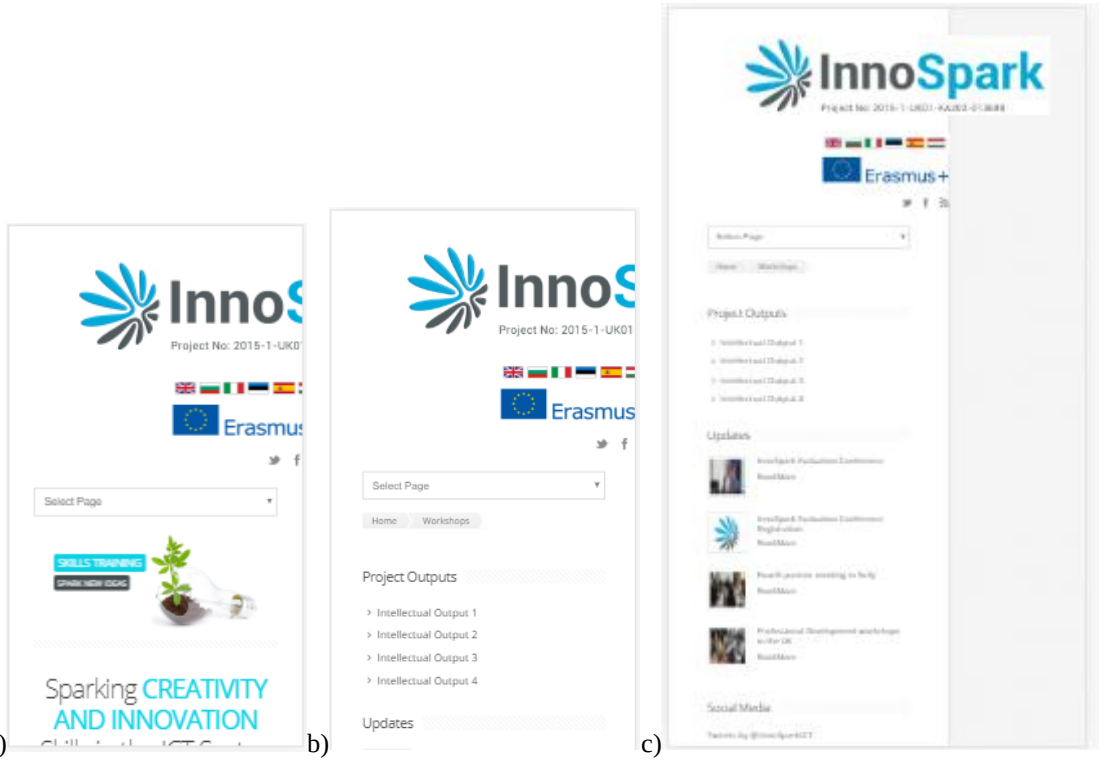
2.1.3 Veebilehe paindlikkuse testimise protsess ja tulemused

Sarnaselt eelmise punktiga leitakse need ekraani suurused, mis on kõige populaarsemad ja järelikult kasutatavad enamikus seadmetes. Selleks võetakse aluseks w3counter.com statistika (Joonis 4).

Top 10 Screen Resolutions		
1	640x360	26.53%
2	1366x768	11.92%
3	1024x768	6.49%
4	1920x1080	6.34%
5	667x375	5.88%
6	1440x900	2.81%
7	736x414	2.69%
8	1280x800	2.46%
9	800x600	2.36%
10	568x320	2.34%

Joonis 2. Kõige populaarsemad ekraani suurused vastavalt w3counter.com reitingule.

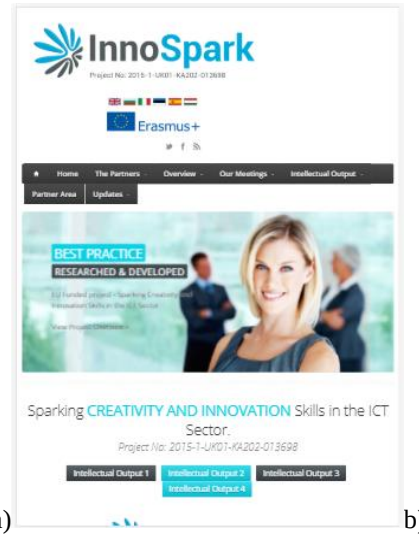
Grey-box meetodi *black*-osa raames võetakse suurused ükshaaval ette ja võrreldakse visuaalsete komponentide paigutust; lõpuks proovitakse täita testitavat kasutusjuhtu ja valideerida funktsionaalsus. Testimine toimub manuaalselt, kasutades Chrome DevToolsi. Erinevate suuruste vaated näitavad, et sisu näeb välja segane ja küljendus ei ole õigesti adapteeritud erinevatele suurustele. Näiteks joonised 3 a) ja b) näitavad, et veebilehe päises asuv pilt ei mahu ekraanile; c) näitab, et sisu laius on palju väiksem kui pildi päises. Seega on probleemiks segane küljendus ja elementide fikseeritud suurused.



a) b) c)

Joonis 3. Veebilehe kujundus a) 640x360 b) 667x375 c) 736x414 suurusega.

Joonisel 4 a) ja b) ei mahu menüü ekraanile ja c) sisu laius erineb päise laiusest. Seega on probleemiks disaini halb kvaliteet.

a) 

b) 

c) 

Joonis 4. Kujundus a) 1024x768 b) 800x600 c) 568x320 suurusega.

Grey-box meetodi *white*-osa raames analüüsitakse koodi ja kontrollitakse paindliku disaini tunnuseid, milleks on:

- meediapäringud;
- paindlik küljendus;
- paindlikud pildid [15].

Koodi läbivaatamine näitab, et küljendus on tehtud, kasutades Skeletoni raamistikku, aga elementide klassid ei ole korralikult määratud (Joonis 5), mille tulemuseks on segane küljendus. Seega on probleemiks raamistiku vale kasutamine.

```
<div class="eighta columns header_left">
    <a href="http://innospark-ict.eu/"><img src="http://innospark-ict.eu/images/innosparklogo.png"  alt="" border="0"></a>
</div>
<div class="eightb columns">
    <div class="header_right">
        <div align="right"><img src="http://innospark-ict.eu/images/topright.png"  alt="" border="0" style="margin-top:20px;"></div>
        <div class="header_soc_search clearfix">


            <a href="http://localhost:8500/InnoSpark/updates/" class="header_soc_rss" id="soc_rss">RSS</a>
            <a href="https://www.facebook.com/InnoSparkICT" target="_blank" class="header_soc_fb" id="soc_fb">Facebook</a>
            <a href="https://twitter.com/InnoSparkICT" target="_blank" class="header_soc_twitter" id="soc_twitter">Twitter</a>

        </div>
    </div>
</div><!-- -->
```

Joonis 5. Vigased Skeletoni raamistiku klassid.

Lisaks on Skeletoni klasside asemel määratud fikseeritud konteinerite kõrgus ja laius, mis ei sobi kõigile ekraanidele. Mõned pildid ja karussell on dünaamilised, aga nende suurus sõltub vigastest konteineritest, osade suurused on fikseeritud (Joonis 6).

```
<img src="images/tool.png" width="456" height="118" alt="" border="0" style="margin-top:15px" ></a>
```

Joonis 6. Fikseeritud pildi suuruse näide.

2.2 Nõuete määramine

Järgnevas peatükis tulevad loetlemisele nõuded.

2.2.1 Teoreetiline taust

Nõude definitsioon koosneb kahest vaatest: kasutaja poolt vaadates on nõue tingimus või võime, mida ta vajab probleemi lahendamiseks või eesmägi saavutamiseks, süsteemi poolt vaadates on nõue tingimus või võime, mida süsteem peab omama, et täita standardi, spetsifikatsiooni või muud ametlikult kehtestatud dokumendi [16].

Nõuete fikseerimisel tuleb määrata probleem, mitte lahendus [17]. Samas arvestatakse kvaliteetsete nõuete põhilisi tunnuseid:

- tarvilik
- teostatav
- korrektne
- sisutihe
- üheselt mõistetav
- täielik
- järjepidev
- kontrollitav
- jälgitav

- disainilahendusest sõltumatu [18].

2.2.2 Funktsionaalsed nõuded

Kõigepealt pannakse paika funktsionaalsed nõuded. Olemasoleva rakenduse funktsionaalsus jääb samaks peale selle, et peaks olema võimalik muuta testi küsimusi.

- Süsteemis peab olema võimalus lisada testi küsimusi.
- Süsteemis peab olema võimalus kustutada testi küsimusi.
- Süsteemis peab olema võimalus muuta testi küsimusi.

2.2.3 Mittefunktsionaalsed nõuded

Lähtudes olemasoleva rakenduse probleemidest, on mittefunktsionaalsed nõuded järgmised:

- Süsteem peab toetama populaarsemaid seadmeid.
- Süsteem peab toetama populaarsemaid brausereid.
- Süsteem peab toetama populaarsemaid ekraani suuruseid.

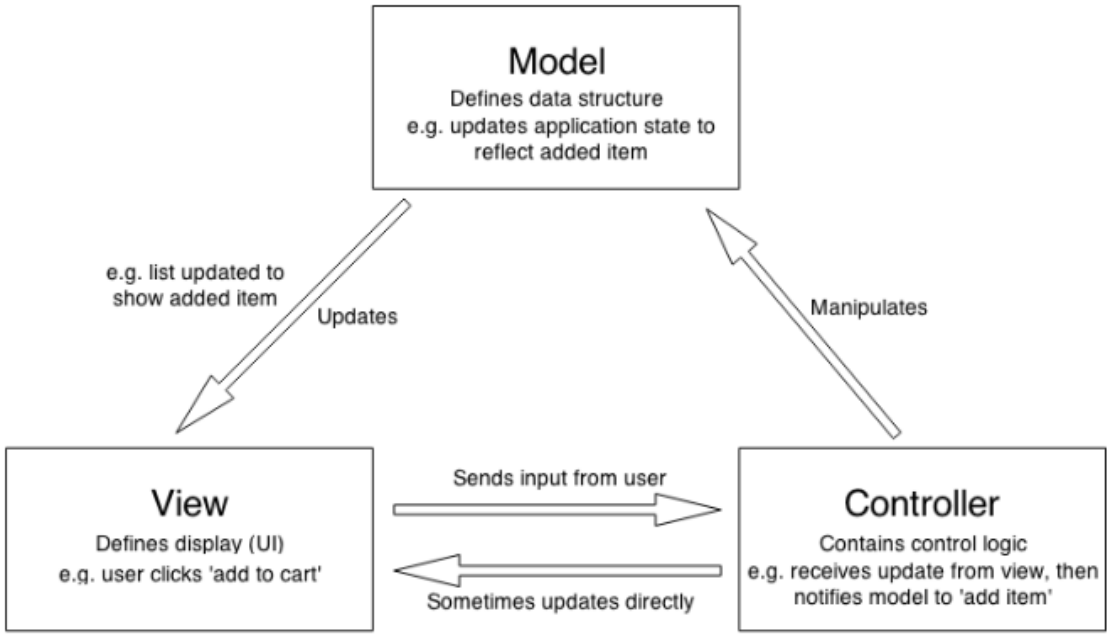
3 Arendus

Järgmisena planeeritakse veebirakenduse struktuur ning valitakse mustrid ja tehnoloogia.

3.1 Olemasoleva lahenduse tehnoloogia

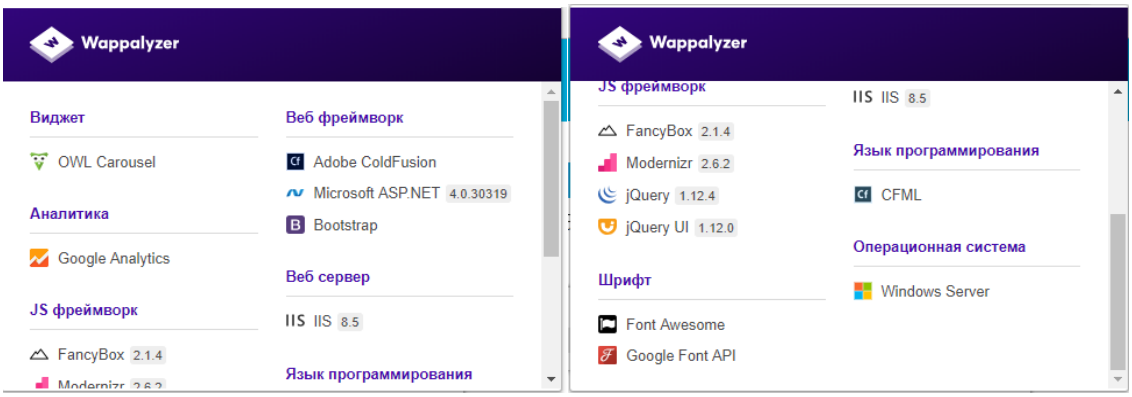
Traditsioonilised veebirakendused sisaldavad kolme kihti (*3-tiered architecture*): veebibrauser (klient), rakenduse server ja andmebaasi server, mis hoiab andmeid ja äriloogika protseduure. Klient vastutab selle eest, kuidas rakendust kasutatakse, ja valideerib sisendid [21]. Kuna serveripoolne lähtekood ei ole avalik, ei ole teada, kas küsimused võetakse andmebaasist või sisemisest andmestruktuurist, aga on teada, et käesoleva veebirakenduse aluseks on kliendi-serveri suhtlemine: kasutaja käivitab rakenduse brauseris ja annab veebisaidile sisendi. Seega on rakenduse tehnoloogia esimeseks komponendiks kliendi-serveri mudel, mis suhtleb andmebaasiga, selleks et oleks kerge andmeid muuta ja lisada ilma lähtekoodi muutmata.

Rakendus tuleb korralikult struktureerida, et oleks lihtne lisada uut funktsionaalsust [20]. Selleks on tihti kasutatav arhitektuurne muster MVC, kus visualiseerimise, andmete ja loogika eest vastutavad mudel (*model*), vaade (*view*) ja kontroller (*controller*). Vaade tagab kasutaja ja rakenduse suhtlemise ja kontroller reageerib süsteemi muutustele asjakohaselt mudelit ja vaadet muutes. Mudel tegeleb andmetega, vastab päringule ja muudab rakenduse olekut (Joonis 7) [22]. MVC mustri eeliseks on komponentide koostöö lihtsustamine ja nende korduvkasutuse võimalus. Samuti muutuvad rakendused paindlikumaks [23]. Antud juhul ei ole vähemalt kliendipoolsed failid kuidagi struktureeritud. Seega on tehnoloogia teiseks komponendiks MVC arhitektuuri rakendamine.



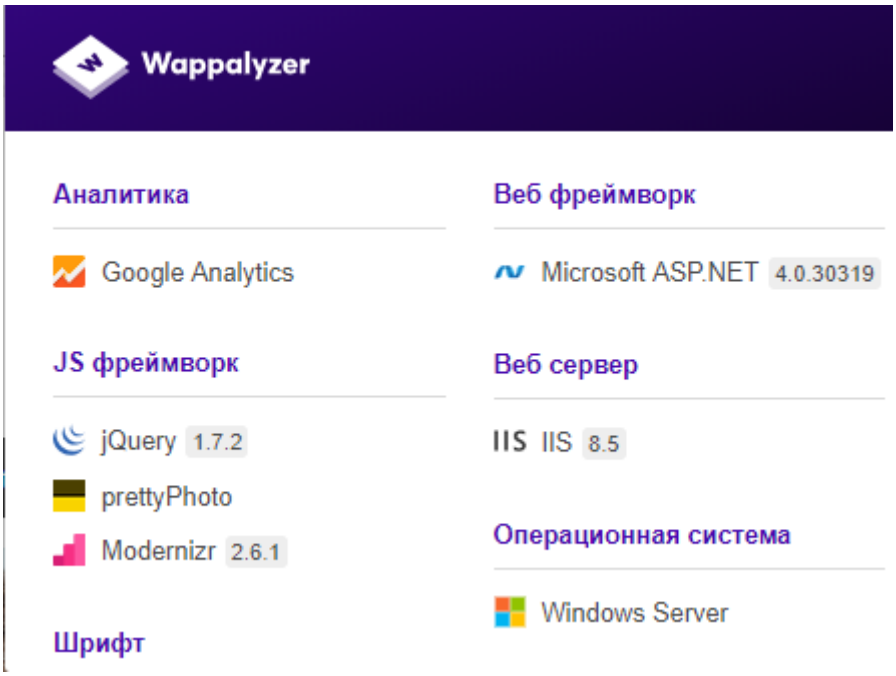
Joonis 7. MVC struktuur [23].

Kuna rakenduse kujundus peab olema responsiivne, tuleb valida vastav *front-end* tehnoloogia. Kasutatavate teekide ja raamistike teadmiseks on kasutatud Wappalyzerit. Olemasolev lahendus koosneb kahest osast: veebiportaal ja veebirakendus, seega on Wappalyzer rakendatud mõlematele. Joonisel 8 on rakenduse analüüsi tulemused.



Joonis 8. Wappalyzeri rakenduse analüüsi tulemused: Vidin, Analüütika, JS-Raamistik, Veebiraamistik, Veebiserver, Programmeerimiskeel, Kiri, Operatsioonisüsteem.

Veebirakenduse *front-end-i* kuuluvad Bootstrap 3, jQuery ja teised CSS ja JavaScript teegid. Veebiportaal kasutab kujunduseks ja animatsiooniks erinevaid CSS ja JavaScripti *front-end* raamistikke, näiteks Skeletoni ja PrettyPhotot (Joonis 9). Bootstrap 3 vastutab responsiivse küljenduse eest, seda dubleerib Skeleton. Seega on tehnoloogia kolmandaks kompondiks responsiivne *front-end* teek.



Joonis 9. Wappalyzeri veebiportaali tulemused: Analüütika, JS-Raamistik, Veebiraamistik, Veebiserver, Operatsioonisüsteem.

Kaasaegse veebiarenduse trendiks on *single page application* disain, kus kogu rakendus koosneb ühest veebilehest, mille sisu muutub vastavalt otspunktile. Kuna liides koosneb erinevatest komponentidest, mida saab sõltumatult muuta, on selline rakendus skaleeruv ja paindlik. Kuna veebileht alla laaditakse korraga ja pärast kuvatakse osade kaupa, on rakendus kiirem. Antud juhul on SPA realiseeritud: pealeht on dünaamiliselt genereeritud Adobe ColdFusion mootoriga. ColdFusion on mõeldud serveripoolse osa arendamiseks. See on ka programmeerimiskeel, mis võimaldab veebirakendusel suhelda *back-end* süsteemiga. ColdFusioni leheküljed koosnevad HTML-ist ja ColdFusioni siltidest. Server genereerib HTML-lehe, mille seejärel brauser kuvab [36]. Rakenduse *back-end* sisaldab ka Microsoft's ASP.NET-i tehnoloogiat, mille eeliseks on see, et programm on eelnevalt kompileeritud serveril, mis kiirendab rakendust [37]. Tehnoloogia järgmine komponent on SPA, et rakendus oleks skaleeruv ja kiire.

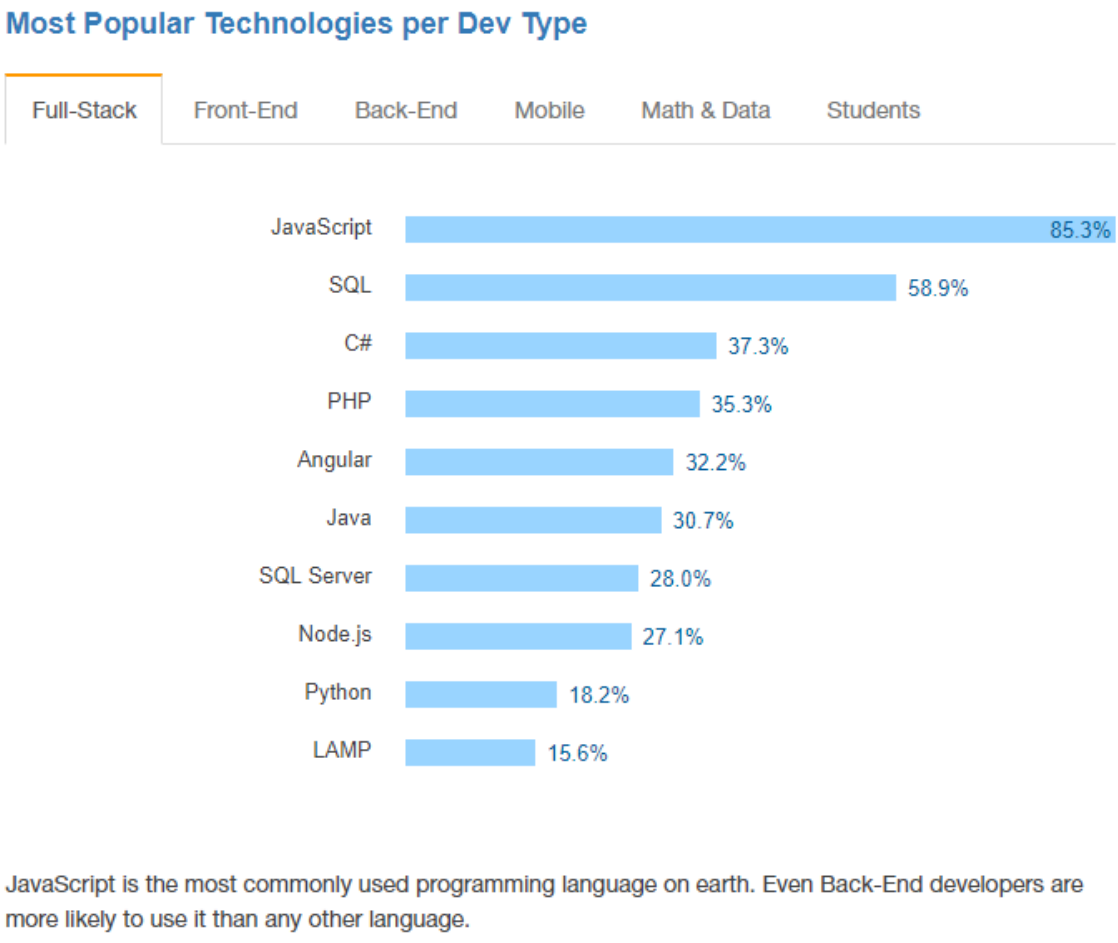
Viimased komponendid on server, kus asub rakendus, ja serveri operatsioonisüsteem. Antud juhul on valitud IIS ja Windows Server.

3.2 Tehnoloogia valimine

Järgnevalt võrreldakse olemasoleva rakenduse tehnoloogiat teiste populaarsete tehnoloogiatega. Valiku tegemisel arvestatakse järgmiseid tingimusi:

- Realiseeritakse kliendi-serveri mudel.
- Andmed liiguvad andmebaasi ja kliendi vahel.
- Kasutatakse MVC arhitektuuri.
- Kasutatakse responsiivset *front-end* teeki.
- Realiseeritakse SPA.
- Rakendus paigutatakse serverisse.
- Server toimib operatsioonisüsteemi baasil.

Võrdlemiseks võetakse olemasolevas veebirakenduses kasutatavad ASP.NET, IIS ja Windows Serveri tehnoloogia ja kaks populaarsemat täis-pinu tehnoloogiat – LAMP ja MEAN. Stackoverflow portaali 2017. aasta uuring näitab, et JavaScript, NodeJS, Angular ja terve LAMP on kõige populaarsemad tehnoloogiad (Joonis 10).



Joonis 10. Täis-pinu tehnoloogiate populaarsus stackoverflow.com põhjal.

WISA baseerub Microsoft'il: Windows, IIS, SQL server ja ASP.NET.

LAMP pinu on avatud lähtekoodiga arendusvahendite kombinatsioon: operatsioonisüsteem Linux, veebiserver Apache, andmebaas MySQL,

programmeerimiskeel näiteks PHP, Perl või Python. LAMP on populaarne, kuna see on vaba ja avatud lähtekoodiga, kohandatav ja sobib enamikule Linuxi OS-i distributsioonidele. Siin vastutab Linux võrgule juurdepääsu, andmete salvestamise ja teiste baasfunktsioonide eest. Avatud lähtekoodiga tarkvara eeliseks on see, et selle suur kogukond (*community*) leiab ja fikseerib vead kiiresti [24].

MEAN kombinatsioon koosneb NoSQL MongoDB andmebaasist, Express.js ja AngularJS raamistikust ja veebiserverist Node.js, kus kõik andmed on esitatud JSON-i kujul.

1. Kliendi-serveri mudel
   Kõik kolm võrreldavad realiseerivad kliendi-serveri mudeli.
2. Andmebaas
   WISA – mitte-relatsiooniline SQL.
   LAMP – mitte-relatsiooniline MySQL.
   MEAN – relatsiooniline MongoDB.
3. MVC
   Kõigile kolmele saab rakendada MVC mudelit.
4. Responsiivsus
   Kõik kolm võrreldavat ühilduvad Bootstrap 4 raamistikuga.
5. SPA
   WISA – SPA, ASP.NET klient võtab loogika endale.
   LAMP – server kompileerib dünaamiliselt vaaded, põhineb sünkroonsetel HTTP-päringutel [25]
   MEAN – SPA, kasutatakse asünkroonseid dünaamilisi AJAX kutseid, erinevalt LAMP-ist AngularJS kannab serveri loogika kliendile üle.
6. Server
   WISA – IIS
   LAMP – Apache
   MEAN – NodeJS
7. Operatsioonisüsteem
   WISA – Windows
   LAMP – Windows, Linux
   MEAN – Windows, Linux.

3.3 Valitud tehnoloogia

Järgnevalt tutvustatakse valitud tehnoloogia.

3.3.1 Valitud tehnoloogia eelised

Käesoleva töö veebirakenduse arendamisel on kasutatud MEAN pinu, pidades silmas selle järgmisi eeliseid:

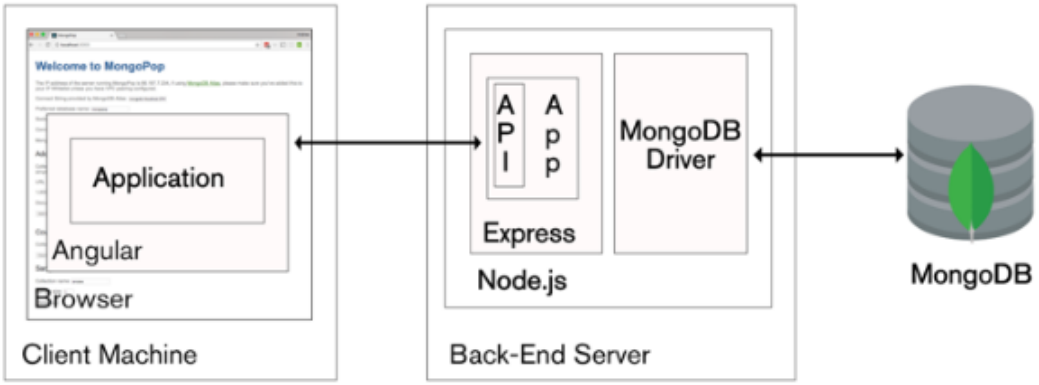
- veebiserverite Node.js ja Express.js kiire ja paindlik arhitektuur: Node on dünaamilise sisu edastamisel palju kiirem kui Apache ja Nginx [26];
- JavaScripti veebirakendus on üle 2,5 korra kiirem kui traditsiooniline PHP rakendus [26];
- asünkroonse kommunikatsiooni võimalus reaalajas [27]: NodeJS on asünkroonsel tegevusel põhinev server, mida saab kasutada *lightweight* ja suure jõudlusega veebiserverikeskkonna loomiseks;
- NodeJS sobib ideaalselt API-de loomiseks: mõne suure ettevõtte (nagu PayPal) API on arendatud Node põhjal [28];
- paindlik, kiire, *lightweight* ja kergesti skaleeritav andmebaas MongoDB;
- puhas ja hooldatav AngularJS *front-end* raamistik;
- serveri- ja kliendipoolse osa arendamine ning andmebaasi pöördumine toimub ühises JavaScript keeles (Joonis 11);
- ühine andmete esitlus JSON kujul;
- JSON kuju on masinloetav ja inimesele lihtsalt mõistetav [26];
- mugavad arenduskeskkonnad;
- mobiilisõbralik [27];
- MongoDB hoiab andmeid võti-väärtus kujul, mistõttu käskude täitmine on palju kiirem kui relatsioonilistel andmebaasidel [29];
- erinevalt staatilistest vormidest ei pea kasutaja „esita“ nuppu vajutama ja ootama – AngularJS reaktiivne kasutajaliides reageerib andmete sisestamise etapil [30].

```
myCollection.insertMany([
    {name: {first: "Andrew", last: "Morgan"},
    {name: {first: "Elvis"}, died: 1977},
    {name: {last: "Mainwaring", title: "Captain"}, born: 1885}
])
.then(
    function(results) {
        resolve(results.insertedCount);
    },
    function(err) {
        console.log("Failed to insert Docs: " + err.message);
        reject(err);
    }
)
```

Joonis 11. JavaScripti objekti salvestamine andmebaasi [30].

3.3.2 Valitud pinu struktuur

Joonis 12 näitab MEAN struktuuri ja tehnoloogiate vahelist suhtlemist. Aluseks on NodeJS põhjal töötav kliendi-serveri suhtlemine, kus AngularJS on klient ja ExpressJS on server. Klient teeb AJAX-i kutse serverile, mis vastab objektile JSON kujul. MongoDB hoiab andmeid ja edastab need ExpressJS-ile.



Joonis 12. MEAN struktuur [30].

ExpressJS vastutab HTTP päringute käsitlemise ehk marsruutimise eest. Marsruutimise funktsioon koosneb teest (*path*) ja HTTP-päringu meetodist (*method*):

```
app.METHOD(PATH, HANDLER),
```

kus *app* – EkspressJS eksemplar ja *handler* on funktsioon, mis käivitub etteantud teel (*route*) [31].

MongoDB on dokumentidele orienteeritud andmebaas, mis kasutab dünaamilist skeemi. See tähendab, et enne andmete sisestamist ei pea defineerima skeemi ja selle struktuur võib olla muudetav ja täidetav uute andmetega [32]. Andmed on binaarse JSON (BSON) kujul [33]:

```
{
    '_id' : 1,
    'name' : { 'first' : 'John', 'last' : 'Backus' },
    'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
    'awards' : [
        {
            'award' : 'W.W. McDowell Award',
            'year' : 1967,
            'by' : 'IEEE Computer Society'
        }, {
            'award' : 'Draper Prize',
            'year' : 1993,
            'by' : 'National Academy of Engineering'
        }
    ]
}
```

AngularJS on *front-end* raamistik, mis põhineb MVC mustril. AngularJS ühendab serverist saadavad andmed sündmuste ja objektidega veebilehel, kasutades malle (*template*) [25].

Pinu keskel asub SPA (*single page application*) muster, mis tähendab, et korraga laaditakse alla kogu veebirakendus (HTML ja JavaScripti failid ja mallid). Seejärel muutuvad veebilehed pidevalt, sõltuvalt kasutaja tegevustest, ilma lehtede allalaadimiseta. Kõik andmed liikuvad läbi REST (Representational State Transfer) teenuse brauseri ja serveri vahel [25].

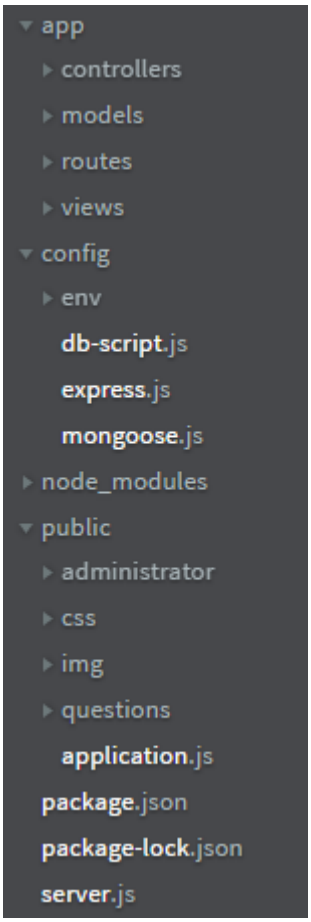
4 Realiseerimine

Selles peatükis on kirjeldatud MEAN täis-pinu veebirakenduse realiseerimise protsessi.

4.1 Rakendusprojekti struktuur

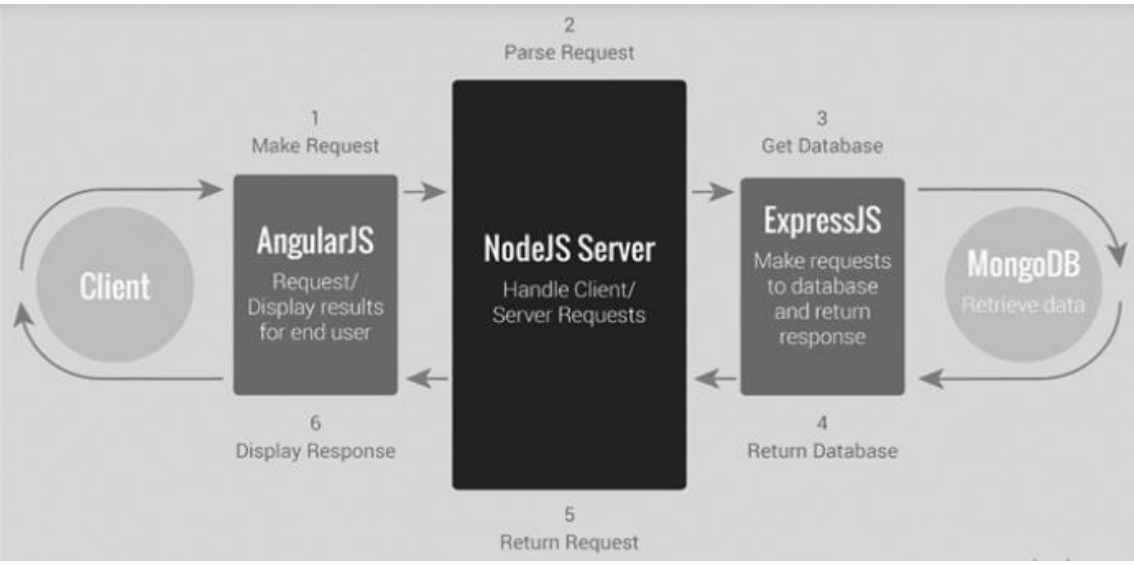
Vastavalt püstitatud nõuetele tuleb realiseerida pealeht ning küsimuste ja tulemuse lehed. Testi küsimusi hoiakse andmebaasis. Selleks, et oleks võimalik küsimusi muuta, realiseeritakse CRUD API administraatori liidese kaudu.

Rakendus on realiseeritud MEAN-i põhjal, seega võib kogu projekti jagada serveri- ja kliendipoolseks osaks. Peale selle kuulub projekti kaust node_modules, mis sisaldab npm mooduleid, mida projekti imporditakse (Joonis 13).



Joonis 13. Projekti üldine struktuur.

Serveripoolseks osaks on ExpressJS rakendus, mis on loodud NodeJS põhjal. Kliendipoolseks osaks on AngularJS rakendus. Viimaseks komponendiks on andmebaas MongoDB. AngularJS võtab päringu vastu ja saadab selle NodeJS serverile, mis annab selle edasi ExpressJS-ile. Viimane küsib andmeid MongoDB andmebaasist, saab vastuse ning annab selle NodeJS serverile. Lõpuks kuvab AngularJS andmed. Kogu protsess on kujutatud Joonisel 14.

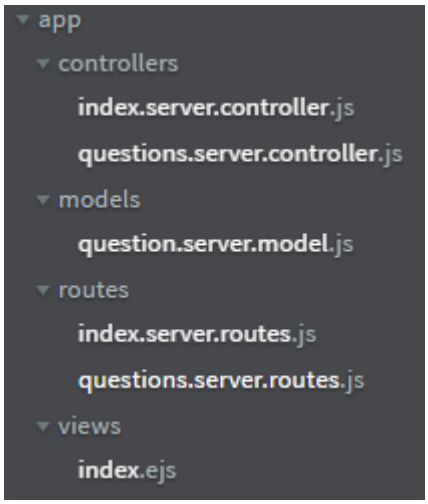


Joonis 14. Rakenduse arhitektuur [34].

4.1.1 Rakenduse serveripoolne struktuur

Rakenduse serveripoolsed failid on paigutatud järgmistesse kaustadesse:

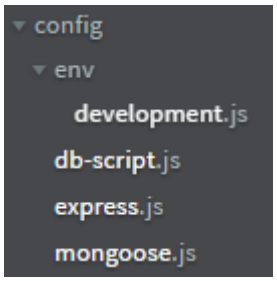
- app – sisaldab ExpressJS serveri rakenduse loogikat ja koosneb järgmistest alamkaustadest vastavalt MVC mustrile (Joonis 15):
  - controller
  - model
  - view
  - routes
- config.



Joonis 15. app kausta sisu.

Põhikaust config (Joonis 16) sisaldab serverirakenduse konfiguratsiooni faili:

- development.js – admebaasi parameetrid
- db-script.js – etteantud info salvestamine andmebaasi skriptiga
- express.js – ExpressJS rakenduse initsialiseerimine ja seadistamine
- mongoose.js – andmebaasiga ühendamine, kasutades mongoose raamistikku.



Joonis 16. config kausta sisu.

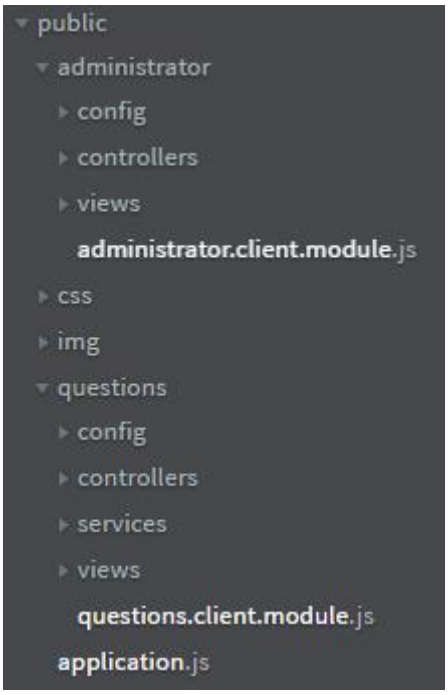
Failis server.js luuakse NodeJS server, mis laeb ExpressJS, kui oma mooduli [22].

4.1.2 Rakenduse kliendipoolne struktuur

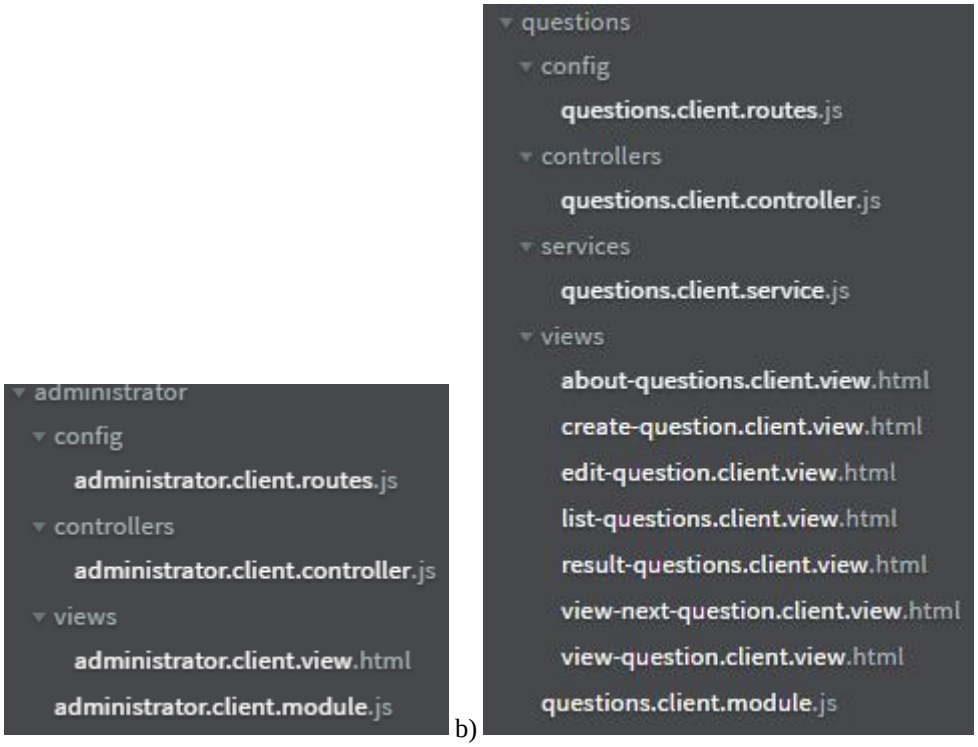
Kliendiks on AngularJS rakendus, mis koosneb moodulitest, mille staatilised failid on grupeeritud kaustadesse MVC järgi (Joonis 17):

- moodulid administrator, question (Joonis 18)
    - config
    - controllers
    - views
    - services

    - failid *.client.modules.js
- css, img – disain ja pildid
- application.js – AngularJS initsialiseerimine.



Joonis 17. public kausta struktuur.



Joonis 18. Moodulite struktuur: a) administraatori moodul, b) Questions moodul.

4.2 Rakenduse realiseerimise põhitegevused

Tabelis 1 on loetletud realisatsiooni põhisammud.

Tabel 1. Realisatsiooni põhisammud.

Samm	Tee	Funktsioon
Server		
1	`config/express.js`	ExpressJS mooduli import ja uue ExpressJS rakenduseksemplari loomine
2	`server.js`	ExpressJS rakenduse start NodeJS serveri põhjal.
3	`config/env/development.js`	Andmebaasi URI salvestamine konfiguratsiooni faili.
4	`config/mongoose.js`	Mongoose mooduli import ja andmebaasiga ühendamine.
5	`server.js`	Mongoose konfiguratsiooni initsialiseerimine.
6	`app/views/index.ejs`	Pealehe malli loomine, mis on kirjutatud, kasutades EJS-id.
7	`app/controllers/ index.server.controller.js`	Andmete edastamine mallile index.ejs. Malli sisu muutub vastavalt kontrollerile.
8	`app/routes/ index.server.routes.js`	Pealehe kontrolleri import ja selle render() meetodi väljakutse.
9	`app/models/ questions.server.model.js`	Mongoose skeemi määramine ja Questionsi mudeli eksportimine.
10	`config/mongoose.js`	Questionsi mudeli registreerimine.
11	`App/controllers/ questions.server.controller.js`	Küsimuste CRUD realiseerimine.
12	`App/routes/ questions.servers.routes.js`	Marsruutide seostamine HTTP meetoditega.
13	`config/express.js`	Marsruutide importimine.
14	`public/application.js`	AngularJS rakenduse initsialiseerimine.
15	`app/views/index.ejs`	<section ng-view> AngularJS dünaamilise sisu konteineri loomine.

Klient		

16	`public/questions/`
	`questions.client.module.js`
	`public/administrator/`
	`administrator.client.module.js`
		AngularJS moodulite loomine.

17	`public/administrator/controllers/` `administrator.client.controller.js`	Praegu ei oma funktsionaalsust.
18	`public/administrator/views/` `administrator.client.view.html`	Administraatori vaate loomine, kus on menüü valikutega luua küsimus või vaadata kõiki küsimusi.
19	`public/questions/views/` `about-questions.client.view.js`	Algusvaate loomine start-nupuga.
20	`public/questions/views/` `create-question.client.view.js`	Vaate loomine, kus on uue küsimuse andmete vorm.
21	`public/questions/views/` `edit-question.client.view.js`	Vaate loomine, mis muudab küsimust.
22	`public/questions/views/` `view-question.client.view.js`	Vaate loomine, mis kuvab küsimuse andmed.
23	`public/questions/views/` `list-questions.client.view.js`	Vaate loomine, mis kuvab kõik küsimused. Küsimuste andmed on võetud kontrollerist.
24	`public/questions/views/` `view-next-question.client.view.js`	Vaate loomine, mis kuvab küsimused ühekaupa.
25	`public/questions/views/` `result-questions.client.view.js`	Vaate loomine, mis kuvab testi tulemuse. Testi tulemus võetakse kontrollerist.
26	`public/questions/controller/` `questions.client.controller.js`	Andmete genereerimine.

27	`public/questions/config/`
	`questions.client.routes.js`
	`public/administrator/config/`
	`administrator.client.routes.js`
		Vaadete ja marsruutide seostamine.

Back-end realisatsiooni käigus installeeriti NodeJS server ja selle baasil loodi ExpressJS rakendus. Seejärel installeeriti ja seadistati MongoDB andmebaas. Andmebaasiga suhtlemine toimub Mongoose raamistikku kasutades. Mongoose on ühendatud andmebaasiga ning andmed vastavad Mongoose skeemile. Kuna tegemist on SPA rakendusega, tehti pealehe mall index.ejs staatiliste HTML-elementidega, kuhu genereeritakse dünaamilisi vaateid. Andmete liikumiseks loodi CRUD mudel ja vastavad

marsruudid. *Front-end* osa realiseerimine teostati, kasutades AngularJS-i, millega loodi vaated ja nende kontrollerid.

4.3 Küsimuste käsitlemine

Ressursside järelepärimiseks luuakse AngularJS $resource teenus:

```
angular.module('questions').factory('Questions', ['$resource',
function ($resource) {
        return $resource('/api/questions/:questionId', {
            questionId: '@code'
        }, {
            update: {
                method: 'PUT'
            }
        });
}]);
```

Meetodid kasutavad $resource teenust REST otspunktidega suhtlemiseks:

- *create* – sisend võetakse vormist välja, seejärel luuakse uue küsimuse ressurss ja saadetakse küsimuse objekti otspunktile.
- *find* – vastuseks *query* meetodile on küsimuste massiiv, mis salvestatakse $scope objekti.
- *findOne* – küsitakse küsimus GET päringuga.
- *update* – muudetakse $scope.question ja salvestatakse see läbi teenuse.
- *delete* – eemaldatakse läbi teenuse [22].

CRUD funktsioonides kasutatakse $resource teenuse meetodit, mille väljakutsumine käivitab $http nimetatud meetodid [35]:

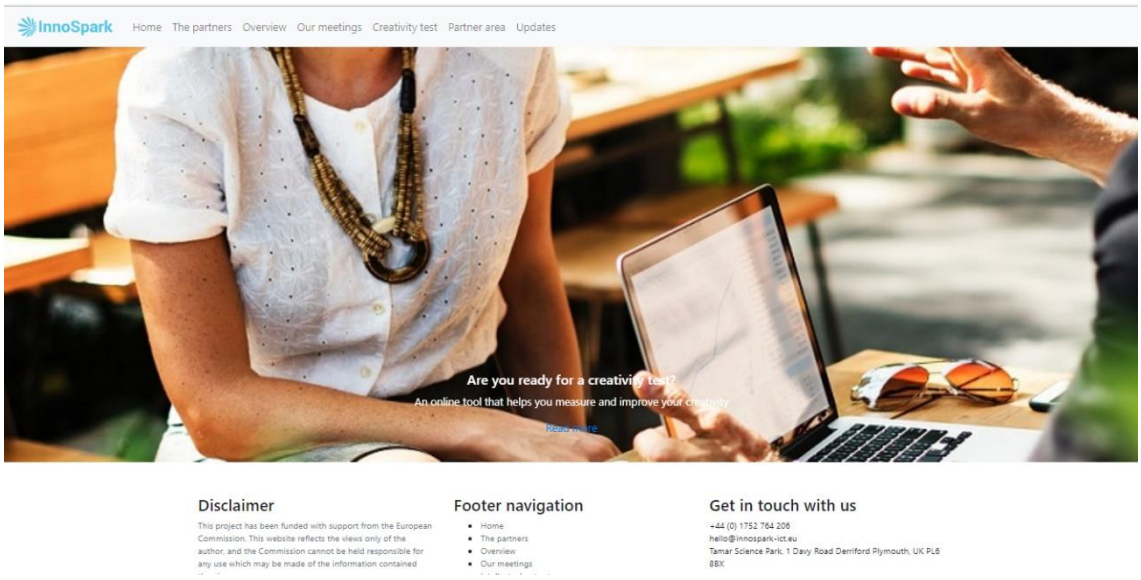
```
{
  'get':    {method: 'GET'},
  'save':   {method: 'POST'},
  'query':  {method: 'GET', isArray: true},
  'remove': {method: 'DELETE'},
  'delete': {method: 'DELETE'}
}
```

4.4 Kujundus

Kuna töö eesmärgiks oli veebirakenduse korrektne kuvamine võimalikult suures arvus seadmetes, valiti kujunduse vormistamiseks responsiivne Bootstrap 4 raamistik.

Joonisel 19 on kujutatud pealehe staatiline vaade, kuhu on paigutatud järgmised elemendid:

- navbar
- carousel
- footer

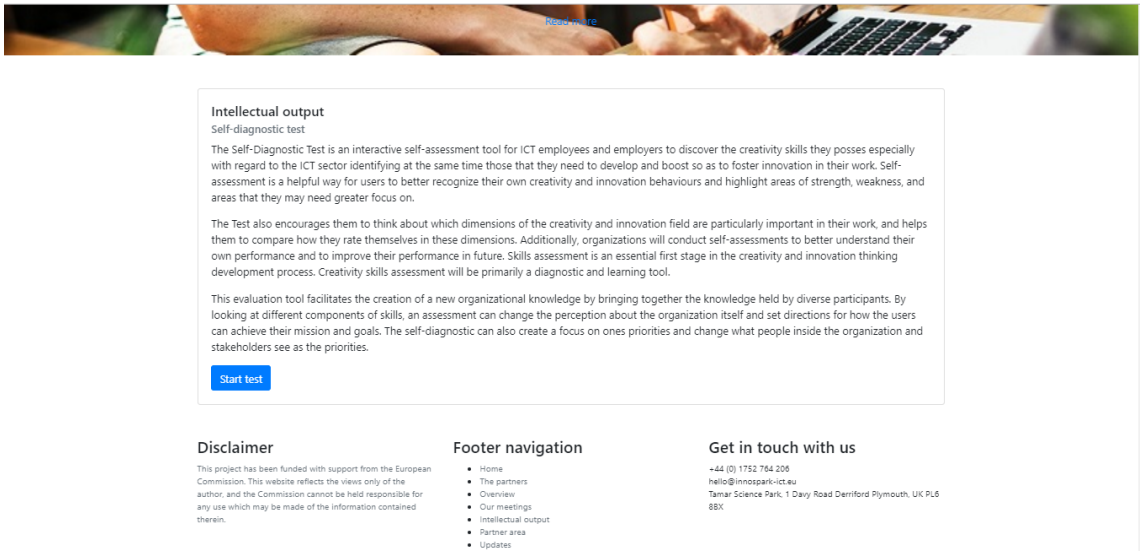


Joonis 19. Pealehe kujundus, kasutades Bootstrapi.

Joonisel 20 on kujutatud testi alusvaade järgmiste elementidega:

- card

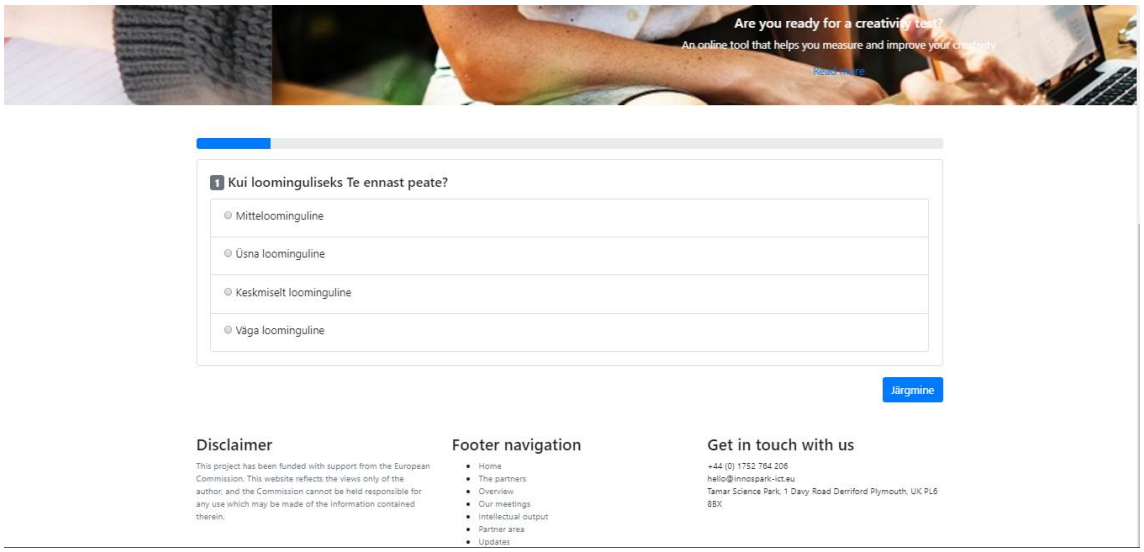
btn.



Joonis 20. Testi alusvaade.

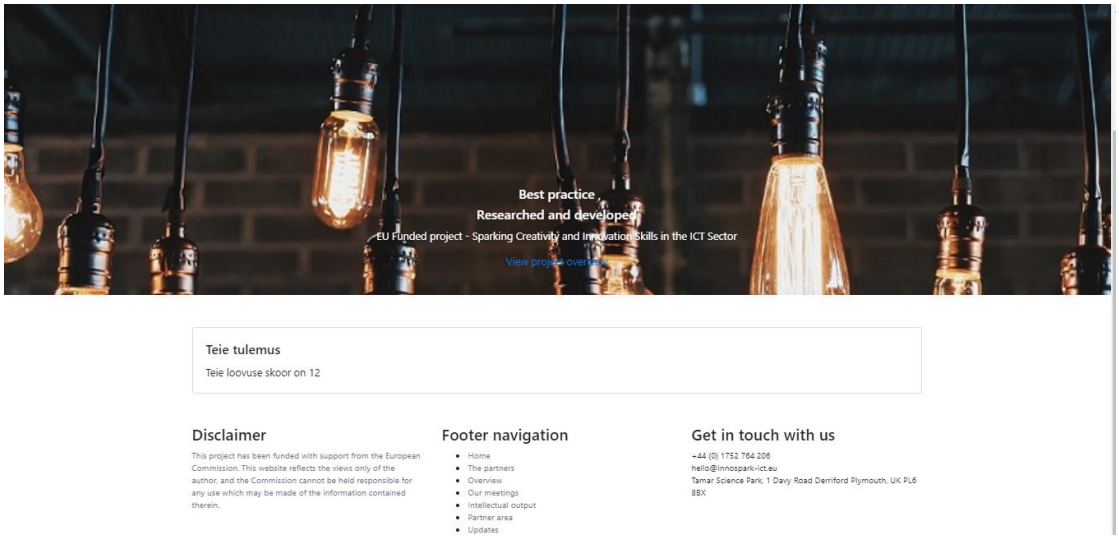
Joonisel 21 on kujutatud testi vaadet komponentidega:

- progress-bar
- card
- btn.



Joonis 21. Testi vaade.

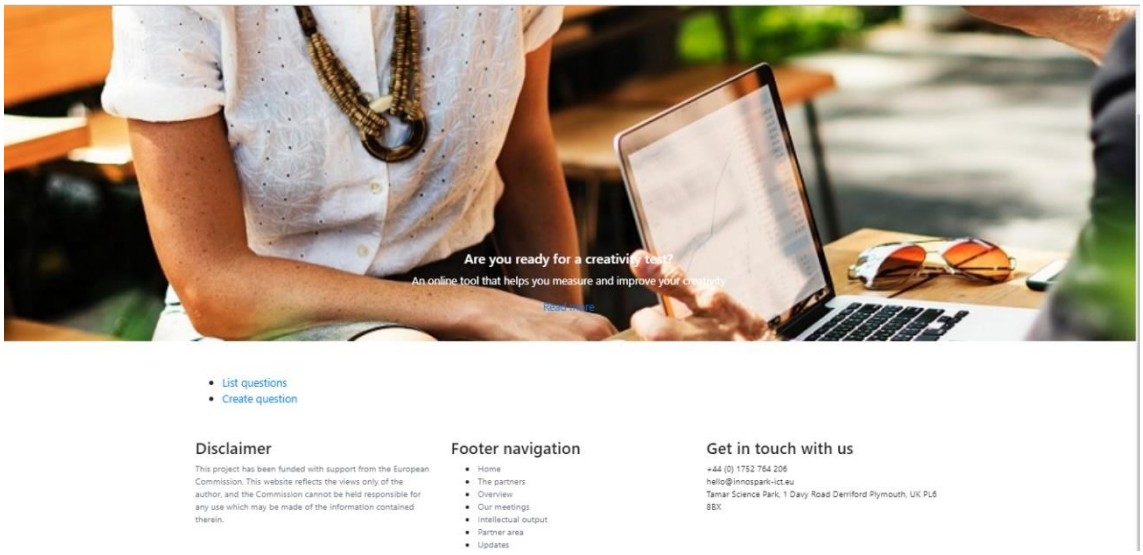
Joonisel 22 on kujutatud tulemuse vaade card-elemendiga.



Joonis 22. Tulemuse vaade.

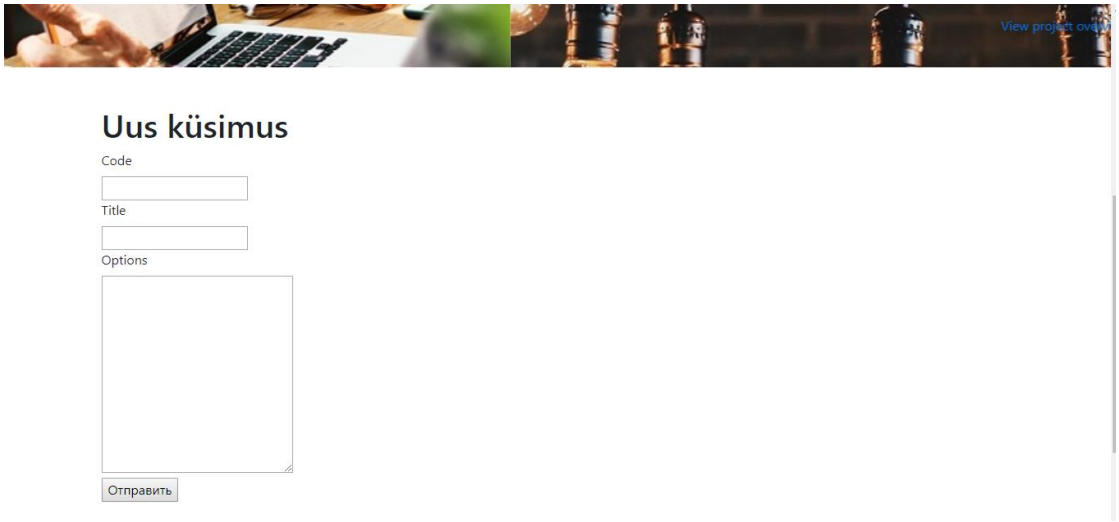
Joonisel 23 on kujutatud administraatori vaade, kus menüüst saab valida kas vaadata kõiki küsimusi või lisada uus. Kui küsimusi ei ole, siis menüü asemel süsteem küsib lisata uue

küsimuse.



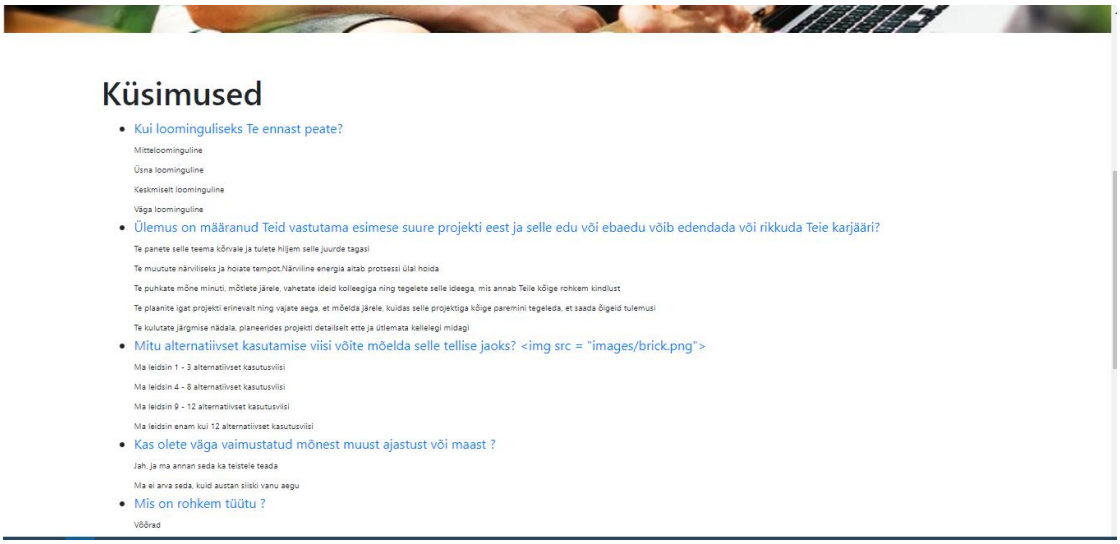
Joonis 23. Administraatori vaade.

Joonisel 24 on kujutatud uue küsimuse lisamise vormi vaade.



Joonis 24. Küsimuse lisamise vaade.

Joonisel 25 on kujutatud kõigi küsimuste vaatamise vaade.



Joonis 25. Küsimuste kuvamise vaade.

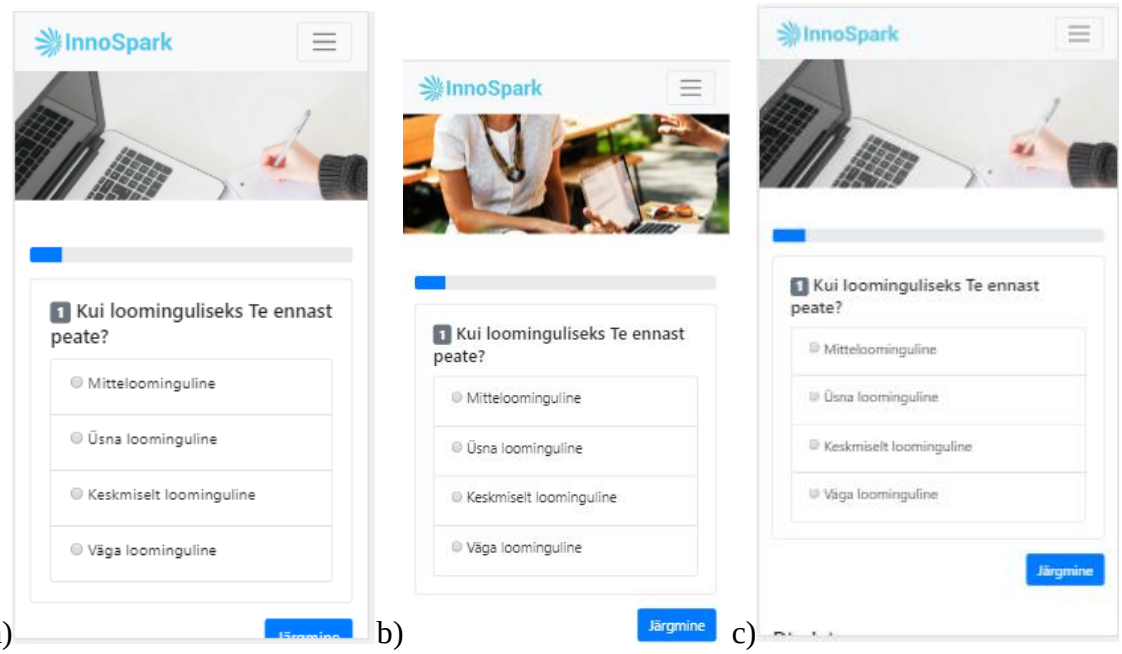
5 Tulemus

Käesoleva töö eesmärgiks oli leida universaalse tehnoloogia, mille rakendamine tagaks veebirakenduse korrektse kuvamise võimalikult suures arvus seadmetes. Järgnevalt kontrollitakse, kas eesmärk saavutati.

Selleks et valideerida tulemus, kasutatakse Google DevToolsi: võetakse vana rakenduse versioonide testimisel kasutatud ekraanisuurused ning võrreldakse neid omavahel. Oodatav on korrektne küljendus kõigil katsetel. Joonistel 25-26 on demonstreeritud vana ja uue versiooni kujundust, kust on näha, et uue versiooni küljendus on korrektne, sisu ja pildid on dünaamilised sõltuvalt seadmest, järelikult tehnoloogia on universaalne.

) b) c)

Joonis 26. Veebilehe kujundus a) 640x360 b) 667x375 c) 736x414 suurusel ekraanil.



Joonis 27. Uus veebilehe kujundus a) 640x360 b) 667x375 c) 736x414 suurusel ekraanil.

Antud töö objektiks oli http://innospark-ict.eu/innosparktool/ lehel asuv veebirakendus, mis tehti ümber vastavalt leitud probleemidele ja määratud nõuetele. Töö tulemuseks on reaalselt töötav rakendus, mis on valmis tööserverisse paigaldamiseks. Kuna rakendus koosneb moodulitest, siis võib funktsionaalsust laiendada.

Olemasoleval rakendusel oli terve rida probleeme, mida uues versioonis õnnestus vältida. Näiteks sisu korrektse küljenduse eest vastutas Bootstrap 4.

Tulemuseks oli kõigi esialgsete nõuete realiseerumine:

- Süsteem peab toetama populaarsemaid seadmeid.
- Süsteem peab toetama populaarsemaid brausereid.
- Süsteem peab toetama populaarsemaid ekraani suuruseid.

Töö skoopi ei kuulunud administraatori autentimine, aga see oleks tarvis realiseerida, kasutades PassportJS-i.

Valmis rakendus on korralikult struktureeritud vastavalt MVC mustrile, mis võimaldab kergesti lisada mooduleid ja funktsionaalsust. Praegu sisaldab projekt pealehte ning küsimuste ja tulemuse lehte. Rakendus suhtleb andmebaasiga läbi realiseeritud CRUD API. Server ja klient on kiired ja paindlikud tänu JavaScriptile.

Rakenduse näitel realiseeriti MEAN täis-pinu tehnoloogia, mille kombinatsioon Bootstrap 4 tagas korrektse kuvamise populaarsetes seadmetes. Sellega on eesmärk saavutatud: otsitavaks tehnoloogiaks võib pidada MEAN täis-pinu ja Bootstrap 4 kombinatsiooni.

6 Kokkuvõte

Bakalaurusetöö eesmärgiks oli leida universaalne tehnoloogia, mille ksutamine tagaks veebirakenduse korrektse kuvamise võimalikult suures arvus seadmetes. Antud juhul tähendas korrektne kuvamine identset funktsionaalsust ja sisu loetavust.

Töö objektiks oli näidisrakendus, mille põhjal leiti tehnoloogia, mis tagas töö eesmärgi. Töö käigus analüüsiti olemasolevat näidisrakendust, sõnastati selle põhjal probleemid ja püstitati funktsionaalsed nõuded. Olemasoleva rakenduse analüüsi käigus testiti ühilduvust erinevate brauserite ja ekraanisuurustega.

Realiseerimine täideti, kasutades MEAN täis-pinu tehnoloogiat, mis koosneb MongoDB, AngularJS, ExpressJS ja NodeJS tehnoloogia kombinatsioonist. Kujunduse osa vormistati, kasutades Bootstrap 4 raamistikku. Tulemust testiti ja saavutatud eesmärk valideeriti.

Otsitavaks tehnoloogiaks võib pidada MEAN täis-pinu ja Bootstrap 4 kombinatsiooni.

Kasutatud kirjandus

[1] Andmekaitse ja infoturbe leksikon, Cybernetica SA [WWW] https://akit.cyber.ee/ (12.05.2018)

[2] „AJAX Introduction“, W3Schools [WWW] https://www.w3schools.com/xml/ajax_intro.asp (12.05.2018)

[3] Goel L. , Majumdar R, “Automation of weblink validation using a generic & reusable automation framework” in *International Conference on Advances in Computer Engineering and Applications (ICACEA)*, 2015, pp. 440-443 [WWW] https://ieeexplore.ieee.org/document/7164745/ (15.05.2018)

[4] Keshk A, Ibrahim A., “Ensuring the Quality Testing of Web Using a New Methodology”, in *IEEE International Symposium on Signal Processing and Information Technology 2007*, 2007, pp. 1071-1076 [WWW] https://ieeexplore.ieee.org/document/4458215 (15.05.2018)

[5] Vince T., Lukac P., Schweiner D., “Android application supporting developed web applications testing”, *International Conference on Modern Electrical and Energy Systems (MEES), 2017*, 2017, [WWW] https://ieeexplore.ieee.org/document/8248941/ (15.05.2018)

[6] Omar F., Ibrahim S., “Designing Test Coverage for Grey Box Analysis”, *10th International Conference on Quality Software (QSIC), 2010*, 2010, [WWW] https://ieeexplore.ieee.org/document/5562984/ (15.05.2018)

[7] „The Difference Between Manual vs Automated Testing“, A Smartbear company [WWW] https://crossbrowsertesting.com/blog/test-automation/difference-manual-automated-testing/ (12.05.2018)

[8] „Happy Path“, Conversion Uplift [WWW] https://www.conversion-uplift.co.uk/glossary-of-conversion-marketing/happy-path/ (12.06.2018)

[9] XUnit Test Patterns Glossary [WWW] http://xunitpatterns.com/happy%20path.html (12.06.2018)

[10] Lu P., Fan W., Sun J., “Webpage cross-browser test from image level”, *IEEE International Conference on Multimedia and Expo (ICME), 2017*, 2017 [WWW] https://ieeexplore.ieee.org/document/8019400/ (15.05.2018)

[11] “Web Application Testing Guide “, Ranorex [WWW] https://www.ranorex.com/resources/testing-wiki/web-application-testing/ (12.05.2018)

[12] W3Counter Browser & Platform Market Share [WWW] https://www.w3counter.com/globalstats.php (12.05.2016)

[13] Rahman A., Devi C., “A framework for ultra-responsive light weight web application using Angularjs”, *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*, 2015, pp. 1-4 [WWW] https://ieeexplore.ieee.org/document/7453857 (12.05.2016)

[14] S. K. Pater, Developing Responsive Web Applications with AJAX and jQuery, Birmingham, UK: Packt Publishing, 2014 [WWW] http://ebookcentral.proquest.com/lib/tuee/reader.action?docID=1706435

[15] R. Neero, „Paindliku veebidisaini arendamine weprint.ee näitel”, Magistritöö, Tallinna Tehnikaülikool, Tallinn, Eesti, 2010

[16] Wahono R. S., Jingde C., “Extensible requirements patterns of web application for efficient web application development”, *First International Symposium on Cyber Worlds, 2002*, Proceedings [WWW] https://ieeexplore.ieee.org/document/1180908/ (15.05.2018)

[17] S. Withall, Software Requirement Patterns, Redmond, Washington, United States of America: Microsoft Press, 2007

[18] R. R. Young, “The requirements Engineering handbook”, Boston, United States of America: Artech house, 2004 [WWW] http://ebookcentral.proquest.com/lib/tuee/reader.action?docID=227599

[19] R. R. Young, “The requirements Engineering handbook”, Boston, United States of America: Artech house, 2004 [WWW] http://ebookcentral.proquest.com/lib/tuee/reader.action?docID=227599

[20] M. Piispanen, „Modern architecture for large web applications”, B. A. thesis, University of Jyväskylä, Finland, 2007 [WWW] https://jyx.jyu.fi/dspace/bitstream/handle/123456789/54129/URN:NBN:fi:jyu-201705272524.pdf?sequence=1 (12.05.2018)

[21] “Components of web-based applications“, IBM Knowledge Center [WWW] https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/intro/src/tpc/db2z_componentsofwebapplications.html (12.05.2018)

[22] A. Q. Haviv, MEAN Web Development, Birmingham, UK: Packt Publishing Ltd, 2014.

[23] „MVC architecture“, Mozilla [WWW] https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture (12.05.2018)

[24] K. A. Coombs. A. J. Hollister, Open Source Web Applications for Libraries, Medford, United States of America: Information Today, 2010 [WWW] http://ebookcentral.proquest.com/lib/tuee/reader.action?docID=3316122 (16.05.2018)

[25] P. Louridas, “Component Stacks for Enterprise Applications”, *IEEE Software,* vol. 33, issue 2, Mar.-Apr. 2016 [WWW] https://ieeexplore.ieee.org/document/7420497/ (16.05.2018)

[26] Poulter A. J., Johnston S. J., Cox S. J., “Using the MEAN stack to implement a RESTful service for an Internet of Things application”, *IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015*, 2015 [WWW] https://ieeexplore.ieee.org/document/7389066 (16.05.2018)

[27] Khue T. D., Binh N. T., Chang W., “Design and implementation of MEAN stack-based scalable real-time Digital Signage System”, [WWW] Information and Communication *8th International Conference of Technology for Embedded Systems (IC-ICTES), 2017,* 2017 [WWW] https://ieeexplore.ieee.org/document/7958779/ (16.05.2018)

[28] Rufus, V. (2014). Angularjs web application development blueprints. Retrieved from http://ebookcentral.proquest.com

[29] Ramappa V., Bein D., “MusiqGlobe.fm using MEAN stack”, Computing and *IEEE 8th Annual Communication Workshop and Conference (CCWC), 2018,* 2018 [WWW] https://ieeexplore.ieee.org/document/8301685/ (15.05.2018)

[30] „The Modern Application Stack – Part 1: Introducing The MEAN Stack“, MongoDB [WWW] https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack (12.05.2018)

[31] „Basic routing“, Express [WWW] http://expressjs.com/en/starter/basic-routing.html (12.05.2018)

[32] „Dynamic Schema Design“, MongoDB [WWW] https://www.mongodb.com/scale/dynamic-schema-design (12.05.2018)

[33] „JSON and BSON“, MongoDB [WWW] https://www.mongodb.com/json-and-bson (12.05.2018)

[34] „MEAN Stack Architecture: AngularJS, NodeJS, ExpressJS and MongoDB“ Evince Development [WWW] https://evincedev.com/blog/mean-stack-architecture/ (12.05.2018)

[35] AngularJS dokumentatsioon [WWW] https://docs.angularjs.org/api/ngResource/service/$resource (12.05.2018)

[36] „About Adobe ColdFusion“, Adobe [WWW] https://helpx.adobe.com/coldfusion/using/about-coldfusion.html (17.05.2018)

[37] „ASP.NET Web Site Project Precompilation Overview“, [WWW] https://msdn.microsoft.com/en-us/library/bb398860.aspx (17.05.2018)