TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Pearu Sarv IADB179481

UUE EESTI HARIDUSE INFOSÜSTEEMI PROJEKTI 1.ETAPI KASUTAJALIIDESE ARENDUS

Bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad

Tallinn 2020

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: Pearu Sarv

30.04.2020

Annotatsioon

Antud lõputöö ülesandeks on Haridus- ja Teadusministeeriumi poolt tellitud EHIS2 projekti 1.etapi kasutajaliidese arendus. Kuna arendus kestab mitu aastat, seadis autor eesmärgiks valida sobivad tehnoloogiad ja meetodid, mis muudaksid arenduse jätkusuutlikuks ka rakenduse kasvades ning kiirendaksid arendusprotsessi järgmistes etappides.

Töös on kirjeldatud rakenduse arendusprotsessi, tuues välja suurimad takistused ning eesmärgini jõudmise teekonna.

Töö tulemusena valmis edukalt rakendus, mis on kliendile üle antud ning mille funktsionaalsus on tema valitud ametnike poolt testitud. Antud töö lõpus annab oma hinnangu ka töö autor.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 3 peatükki, 6 joonist, 0 tabelit.

Abstract

Front-end development of the new Estonian Education Information Management System

The goal of this thesis is completion of the EHIS2 projects 1st phase front-end development. EHIS2 project development is ordered by Estonian Ministry of Science and Education. EHIS2 is set to replace the current EHIS, which running costs have been increasing over the years.

Since the development of EHIS2 will last over multiple years, it is important that it would be sustainable. So the aim of the thesis is to choose correct technologies and methods and create the project in a way that it would stay easily managable as it scales.

The authors explains the process of development using the chosen method and by bringing out the most difficult challenges.

The application has been succesfully handed over to the client and its compliance to specification has been approved by officials. At the end of the thesis, its author also gives his opion.

The thesis is in Estonian and contains 34 pages of text, 3 chapters, 6 figures, 0 tables.

Lühendite ja mõistete sõnastik

CSS	*Cascading Style Sheets*, stiililehe keel
CSV	*Comma Separated Values*, komaga eraldatud väärtustega failivorming
DOM	*Document Object Model*, dokumendi objekti mudel
EHIS	Eesti Hariduse Infosüsteem
FormArray	Angulari klass vormivälja staatuse ja väärtuse jälgimiseks
FormControl	Angulari klass vormiväljade grupi staatuse ja väärtuse jälgimiseks
FormGroup	Angulari klass dünaamilise vormi vormiväljade kogumi staatuse ja väärtuse jälgimiseks
MutationObserver	Liides DOM'i puus toimuvate muudatuste jälgimiseks
NgForOf	Direktiiv Angularis vaates kollektsioonidest mallide loomiseks
REST	*Representational State Transfer*, tarkvara arhitektuuri laad
RxJS	Teek asünkroonse koodi kirjutamise lihtsustamiseks
Sass	Stiililehe keel, mis täiendab ning kompileerib CSS'iks
URL	*Uniform Resource Locator*, üldine infoallika asukohamääraja

Sisukord

Jooniste loetelu

1 Sissejuhatus

Töö eesmärgiks on uue Eesti Hariduse Infosüsteemi projekti 1. etapi kasutajaliidese valmimine. Teema valik kujunes välja autori praktika ajal, millele ta asus antud projekti arenduse alustamise hetkel. EHIS2 projekt on osa Haridus- ja Teadusministeeriumi poolt algatatud uue Haridusportaali arendusest, mis aitab tulevikus noori haridusteekonnal suunata. Üheks oluliseks aspektiks antud projekti juures on selle pikaajaline kestus. Seetõttu on autori arvates oluline, et valitaks kohe alguses sobivad töövahendid ja meetodid, millega töö eesmärgini jõuda, selleks et arendus oleks jätkusuutlik ka projekti järgmistes etappides.

Antud töö sisu on jaotatud kolme osasse. Esimene osa on teoreetiline. Selles osas uuritakse põgusalt EHIS1 puudujääke, kirjeldatakse EHIS2 projekti 1.etapi skoopi, valitakse antud projektile sobivad töövahendid, tehnoloogiad ja meetod ning kirjeldatakse arendusprotsessi. Töö teine osa keskendub praktilisele poolele ehk arendusele. Antakse ülevaade osadele ülesannetele ja tekkinud probleemidele ning leitud lahendustele. Töö viimases osas annab autor hinnangu saavutatud tulemusele ning analüüsib mida saaks tulevastes etappides veel paremini teha.

2 Analüüs

2.1 EHIS1 puudused

2011ndal aastal tellis Haridus- ja Teadusministeerium EHISe eksperthinnangu [1], milles hinnati selle jätkusuutlikust. Uuringust järeldus, et tulevased arendused muutuvad aina ressursi rohkemaks ning selle teenuse pakkujaid jääb aina vähemaks. Samuti leiti, et kasutatud tehnoloogiad on iganenud ning leidus ka probleeme andmetega. Kuigi põhiliselt seonduvad probleemid tagarakendusega, on autori hinnangul iganenud ka kasutajaliides. Tabelitega loodud paigutus ei vasta enam tänapäevastele normidele ning ka disain on töö autori arvates arhailiseks muutumas.

2.2 Töö skoop

EHIS2 arendus hõlmab ennast nii uue tagarakenduse kui ka kasutajaliidese loomist. Antud töö keskendub kasutajaliidese arendusele, tagarakenduse arendusega tegeles teine firma ning sellega autor seotud ei ole. Kasutajaliidese arenduses osales töö autor koos teise arendajast tiimiliikmega. Seetõttu pole antud töös kirjeldatud kõiki EHIS2 kasutajaliidese osi, vaid keskendatud just nendele, mille enamuses või täielikuks loojaks on töö autor. Kuna projekt on väga mahukas, siis ei ole antud töös kõikide lahenduste detailidesse laskutud. Kirjeldatud on üldisemalt kõikide töö autori poolt loodud vaateid ning nende loomise protsessi ja selle käigus tekkinud suurimad ning autori arvates huvitavamad probleemid, mille lahendused väärivad välja toomist.

2.3 Arendusprotsess ja töövahendid

Arendusemetoodikaks sai valitud agiilne Scrum raamistiku järgi toimuv arendus. Scrumi tunnusteks on iteratsioonidena arendus, väiksed tiimid, vahetu tagasiside kliendilt ning igapäevased koosolekud [2]. Antud projekti tiim oli seitsme liikmeline: disainer, testija, tiimi juht, kaks analüütikut ning kaks arendajat. Scrumi realiseerimiseks kasutati Atlassiani Jira tarkvara. Jira on üks enim kasutatavmaid rakendusi Scrumi tiimide halduseks, samuti on Atlassian partneriks firmale, milles autor töötab, mis oli veel üheks

suureks eeliseks konkurentide ees. Arendusprotsess toimus Scrumi tavade järgi. Iga kahe nädala järel toimusid sprindikoosolekud, millel osales ka tooteomanik. Koosoleku käigus arutleti eelmise sprindi saavutuste üle ning demonstreeriti valmis saadut. Seejärel valiti *backlog*ist järgmise sprindi ülesanded ning nende piletid tõsteti „Tegemiseks“ lahtrisse. Tiimisiseselt toimusid ka 15 minutilised iga hommikused koosviibimised, mille käigus tiimijuht sai ülevaate tiimi liikmete eelmisel päeval tehtust, hetkelistest takistustest ning plaanist algava päeva plaanidest. Ülevaate parandamisele aitas kaasa ka Jira, kus ülesannete piletid asuvad hetke staatusele vastavas lahtris. Ülesanded ise ning nende nõuded on analüütiku poolt ära kirjeldatud Atlassiani Confluence tarkvaras, millele piletid viitavad. Kui arendaja alustas ülesande täitmist, siis liigutas ta vastava pileti „tegemisel“ lahtrisse ning lõi vastavalt pileti nimele uue *git*i haru. Olles ülesandega valmis saanud, ühildati antud haru arenduskeskkonnaga ning liigutati pilet „ülevaatamiseks“ lahtrisse. Projekti alguses praktiseeriti ka enne testijale üleandmist koodi ülevaatust, mis kujutab endast kaasarendajate koodi ülevaatamist. Mõne aja pärast aga jõudsid töö autor koos oma kaasarendajaga järeldusele, et antud protsess ei tasunud ennast ära ja selle ajakulu ületas saadava kasu ning ise muudetud koodi üle vaatamine oli piisav. „Ülevaatamiseks“ lahtrisse liikunud piletite vastavust spetsifikatsioonile hakkas kontrollima testija. Kui valmis saanud osa ei töötanud nii nagu nõuetes märgitud, siis liikus pilet tagasi arendaja kätte „tegemisel“ lahtrisse kuniks vead olid parandatud. Kui ülesande teostus vastas nõuetele, siis liikus pilet „valmis“ lahtrisse. Antud projekt erines kergelt Scrumi protsessist selle poolest, et kuni projekti valmis saamiseni puudus laivkeskkond, ehk valmis saanud osad jäid arenduskeskkonda, mitte ei liikunud osadele lõppkasutajatele proovimiseks.

Töö autori hinnangul on Scrumi raamistiku kõige suuremaks eeliseks traditsiooniliste arendusmetoodikate ees vahetu tagasiside. Nii mahukate projektide juures on oluline, et kui peaks tekkima erisusi arenduse ja kliendi soovide vahel, siis avastataks need võimalikult varakult, sest projekti lõpufaasis on nende likvideerimine palju kulukam. Antud projekti arendus käis kliendi poolt kinnitatud analüüsi ja spetsifikatsiooni järgi, mistõttu taoliseid olukordi ette eriti ei tulnud, kuid sellegi poolest pakkusid sprindikoosolekud hea võimaluse pakkuda uusi lahendusi ning täiendusi. Antud projekt eelarve oli fikseeritud ehk ühte suurt eelist, milleks on objektiivsemad mahuhinnangud, täielikult utiliseerida ei saanud. Sellegi poolest aitab töö protsessi väiksemateks ülesanneteks jaotamine paremini mõista, mis põhjustas ootamatuid takistusi ja mille

lahendamine läks oodatust paremini ning sellest tuleviku jaoks järeldusi teha. Samuti on hea kui kliendil on olemas pidev ülevaade hetkelistest takistustest, nii et need aja pikku ei koguneks ja ei tuleks järsku halva üllatusena, tekkidades erimeelsusi. Autori hinnangul võivad Scrumi puhul negatiivsed küljed esineda, kui seda liiga rangelt järgida ja mitte oma vajadustele adapteerida. Praktikas tuleb aegajalt ette olukordi, kus sprinti võetud piletid tuleb kõrvale lükata ning võtta ette mõni vahele tulnud ülesanne, samuti kui tööjärg käes siis vahest on mõistlik arendajal endal mõni pilet sprinti võtta, selle asemel, et käske ootama jääda.

2.4 Tehnoloogiad

EHIS2 põhineb REST arhitektuuril, mille tõttu oli vaja valida sobiv raamistik kasutajaliidese loomiseks. Nagu enamus tänapäevastel REST arhitektuuril põhinevatel kasutajaliidestel, oli valik kolme suurima Javascripti raamistiku Vue, Reacti ning Angulari vahel.

Vue on antud tehnoloogiatest uusim. See sündis mõned aastad tagasi Evan You nägemusest raamistikust, mis pakuks paremat vaate interaktiivsuse haldamist kui Backbone ja AngularJS, mida ta tollel hetkel Googles töötades kasutas [3]. Üheks erinevuseks võrreldes teise kahe raamistikuga, on see, et selle arendus ei toimu suure korporatsiooni poolt. Selle positiivseks pooleks on, et kogukonna poolt pakutud täiendused ei pea läbima mitut hierarhilist astet, et teoks saada, samas miinuseks on väiksem ressurss. Vue suurimateks eelisteks on selle lihtsus ja paindlikus. Autori hinnangul on Vuega kõige kergem algust teha ning alustalad selgeks õppida. Samas on Vue paindlik ning võimaldab lisada mitmeid funktsionaalsusi, mistõttu on teda võimalik kasutada ka suuremate projektide jaoks.

Statistiliselt 2020 alguse seisuga on enim kasutavamaiks raamistikuks React [4]. Reacti arendab Facebook ning sellele tehnoloogiale on üles ehitatud näiteks Twitteri ja Instagrami veebi kasutajaliides. Üheks Reacti tunnuseks on JSX'i kasutus. JSX on Javascripti süntaksi laiendus, mida Reactis kasutatakse kasutajaliidese defineerimiseks. Alguses võib JSX'i süntaks tunduda harjumatu, kuid võrreldes puhta Javascriptiga kiirendab see mallide kirjutamise protsessi. React kasutab DOM'i manipuleerimiseks virtuaalset DOM'i. Virtuaalse DOM'i kasutuse eeliseks on kiirem DOM'i uuendamine, eemaldades ebavajalikud muudatused, mis DOM'is on väga aeglased. Kui toimub

muudatus, siis kogu virtuaalne DOM uuendatakse ning seejärel võrreldakse seda korrespondeeriva DOM'iga, milles muudetakse vaid muutunud objektid [5]. Nagu ka Vue, ei sea ka React eriti arhitektuurilisi piiranguid ning projekti struktuur on paindlik.

Angular on Google poolt arendatav raamistik. 2016ndal aastal asendus AngularJS Angular 2ga ja seda on järkjärgult arendatud ning tänaseks on stabiilne versioon 9+. Iga versiooni vahel toimub evolutsioon, mitte revolutsioon, ehk pidevalt lisatakse täiendusi lõhkumata olemasolevaid. Angular seab ka mõningased arhitektuurilised piirangud ja suunised, mis väiksemate, loominguliste projektide puhul võib olla miinuseks, kuid suure infosüsteemi puhul kindlasti eeliseks. Veel üheks Angulari miinuseks väiksemate rakenduste puhul on selle suurus võrreldes teise kahe raamistikuga, mille tõttu võtab ka rakenduse laadimine kauem [6]. Samas tuleneb see paljude funktsionaalsuste sisse ehitamisest, ehk suure rakenduse kontekstis, kus neid vaja läheb ning teiste raamistike puhul juurde tuleb lisada, on tegemist pigem positiivse küljega. Siin kohal tuleb veel märkida, et antud võrdlus on tehtud enne hiljutist Angulari 9ndat versiooni, mis vähendab pakki suurust [7] ehk tuleviku jaoks on antud argument väheoluline. Angular kasutab mudeli defineerimiseks Javascripti asemel Typescripti. On ka võimalik kasutada Javascripti, kuid soovituslik on Typescript kasutus ning selles on ka kogu dokumentatsioon. Typescript lisab Javascriptile tüübid, mis jällegi väiksema projekti puhul võib olla üleliigne, kuid REST arhitektuuril põhineva, tugevalt tüübitud keelega tagarakendusega, tuleb see kasutajaliidese loomisel tuleb abiks. Samuti võimaldab Typescript kasutada uusi ECMAScripti funktsionaalsusi. Antud projekti kontekstis on veel üheks suureks Angulari plussiks, et see on töö autori tiimis kasutusel ka varem alustatud Haridusportaali projektis.

Võttes arvesse antud töö eesmärki, saigi kõige mõjuvamaks argumendiks Angulari kasutus Haridusportaalis. Kuna EHIS2 ja Haridusportaal on omavahel seotud ning nende paljud komponendid on sarnased, siis võimaldab sama raamistiku kasutus ning komponentide jagamine tublisti kiirendada arendusprotsessi. Lisaks on tagarakendus loodud Javas ehk Angulari hea Typescripti tugi on kindlasti eeliseks ning projekti mahtu arvestades tulevad ka Angulari poolt seatavad arhitektuurilised suunised kasuks.

Stiilide kirjutamiseks sai valitud Sass, täpsemalt SCSS süntaks. Sass on stiililehe keel, mis lisab CSS'ile funktsionaalsust ning võimekust [8]. Autori arvates on Sassi kõige suuremaks eeliseks CSS'i ees põimitud süntaks. See vähendab kirjutatava koodi hulka

ning muudab selle paremini loetavamaks. Suurte projektide puhul on väga lihtne lasta stiililehtede struktuur käest ning pärast on neid keeruline korrastada kui pole aru saada kust mingi stiil tuleb. Sass pakub võimaluse luua ka muutujaid, mis on olemas ka CSS3s, kuid mõningate erinevustega. Peamisteks erinevusteks on skoop ning võimalus CSS'i muutujaid Javascriptiga manipuleerida, mis Sassi puhul puudub, kuna brauseri jaoks on need kompileeritud CSS'i väärtusteks. Üheks miinuseks CSS'i muutujate puhul on Internet Explorer 11 toe puudumine. Sassi veel mõningateks eelisteks on võimalus luua funktsioone, *mixin*'id, mis lihtsustavad näiteks meediapäringute kirjutamist, ning importimise võimalus.

2.5 Arhitektuur ja meetod

Rakenduse arhitektuur on oluliseks osaks töö eesmärgi jõudmiseks. Selleks et rakendus püsiks kasvades jätkusuutlik, peab sellel olema korralik arhitektuur. Angulari suurimateks arhitektuurilisteks ehitusblokkideks on NgModule'd [9]. NgModuled loovad komponentide kompileerimise konteksti ning eraldavad erineva eri funktsionaalsusega komponendid. Antud rakenduse puhul sai NgModule'ga eraldatud taaskasutatavad komponendid ning vaadete komponendid. Rakenduse kasvades lihtsustab antud lahendus rakenduse haldust. Vaadete komponendid on jaotatud NgModule-tesse äriloogika järgi. Näiteks on õppeasutuste vaated erinevas NgModules kui õppekavad. Antud lahendus võimaldab kasutada laiska laadimist ja selle eeliseid. Kui kasutaja avab rakenduse ning satub õppeasutuse vaatasse, siis laaditakse vaid õppeasutuse NgModule kontekstis olevate ja sinna imporditud komponentide kood, mitte kogu rakenduse oma, mis kiirendab rakenduse alglaadimist. Rakenduse paremaks haldamiseks on veel Javascripti mooduliteks jagatud direktiivid, *pipe*d, liidesed ning teenused. Teenuste kasutamine Angulari komponentides käib läbi sõltuvuse süstimise, muutes komponendi klassid puhtamaks.

Vastavalt Angulari arhitektuurile ja töö eesmärgini jõudmise sobivusele sai meetodiks valitud komponentide põhine arendus. Arvestades ka kliendi vaadete valmimise tähtaegade ootuseid, ei võetud päris puhtakujulist komponentide põhist arendusmeetodit, kus esmalt luuakse valmis kõik taaskasutatavad komponendid ning alles seejärel alustatakse vaadete arendusega. Selle asemel analüüsiti enne iga uue vaate arendama

hakkamist, milliseid taaskasutatavaid komponente selles kasutatakse ning kas oleks vaja luua mõni uus.

3 Teostus

Selle peatüki raames on kirjeldatud rakenduse tehnilist teostust. Nagu varem mainitud, on teostus loodud analüütikute poolt koostatud nõuete järgi. Peatükist ei ole kirjeldatud kõiki rakenduse vaadete lahendusi, vaid neid mille taga on töö autor. Vastavalt valitud töö metoodikale on peatükis paljuski keskendatud komponentide loomisele. Peatükid on arendusprotsessi järjekorras.

3.1 Administraatori vaated

Kasutajaliidese arendus algas administraatori vaadetest. Sinna alla kuulusid klassifikaatorite, parameetrite ja teavituste mallide haldamise vaated.

Kõige esimesena hakati arendama klassifikaatorite vaateid. Need koosnevad klassifikaatori nimistu vaatest, klassifikaatori juurde kuuluvatest klassifikaatori ridade nimistu vaatest ning klassifikaatori rea detailvaatest ja selle rea juurde kuuluvate seoste vaade, samuti ridade ja nende seoste lisamise ning muutmise vaade. Administraatori vaadete alla kuulusid veel parameetri vaade ning teavituste mallide nimistu, detail ning muutmise ja lisamise vaade.

Kuna klassifikaatori rea detailvaate näol oli ühe esimeste arendatavate vaadetega, siis oli oluline leida korduvaid elemente, millest saaks luua taaskasutavaid komponente, mis kiirendaksid arendusprotsessi. Autori esimeseks komponendiks oligi väga lihtne nuppude mähiskomponent, mis temasse saadetud nupud asetab õigesse kohta ning annab neile vahed. Vaade ise koosneb andmete tulpadest, kus on võtme-väärtuspaaridena kuvatud klassifikaatori definitsiooni, rea tekstide ning atribuutide väärtused. Algselt said vaate andme lahtrid valmis tehtud kasutades HTML'i <dl> märgendeid, mille alla käivad <dt> märgend võtme kohta ja <dd> väärtuse kohta. Kuid ka selle sai komponendiks ümber teha, mis võimaldas lühendada vaate faili ning kiirendada järgmiste loomist. Komponendi sisendiks on kuvatav objekt, tema võtme-väärtuspaarid käiakse komponendi vaates NgForOf'iga läbi ning iga objekti võtme-väärtuspaari kohta kuvatakse <dt> märgendi alla võti, mida muudetakse tõlke pipe'ga ning väärtus <dd> märgendite vahele. NgForOf on Angularis direktiiv, mis vaates võimaldab iga kollektsiooni üksuse kohta malli kuvada [10] ning pipe'd on Angularis klassid, mis implementeerivad PipeTransform liidese *transform* funktsiooni ning võimaldavad muuta andmeid sobivale kujule [11]. Antud

lahendus oli ka hea kuna tõlgete kasutamine oli samuti üks nõuetest. Hilisemalt sai komponenti veel eri juhtude jaoks täiendatud ning sisendiks saab ka saata loendi objekti võtmetest, mida ei soovita kuvada, või siis loendi objekti võtmetest, mida soovitakse kuvada kindlas järjekorras. Vaate juurde kuuluvad ka ridade vaheliste seoste andmeplokid, mis asuvad seoste saki all. Sakkide komponent oli juba varasemalt lahendatud Haridusportaali arendusel ning selle sai sealt üle võtta. Kogu lehe sisu läheb <block> märgise sisse ning selle all läheb iga saki sisu <block-content> märgise sisse. „Block“ komponendis moodustatakse ContentChildren dekoraatoriga „block-content“ komponentidest sakkide QueryList, vaates luuakse iga saki kohta saki nupp ning kuvatakse vaid hetkel aktiivset olevat saki sisu. Klassifikaatori rea seoste saki all kuvatakse klassifikaatori rea seoseid teiste ridadega ning ka need sai teostada varem kirjeldatud andmebloki komponendiga.

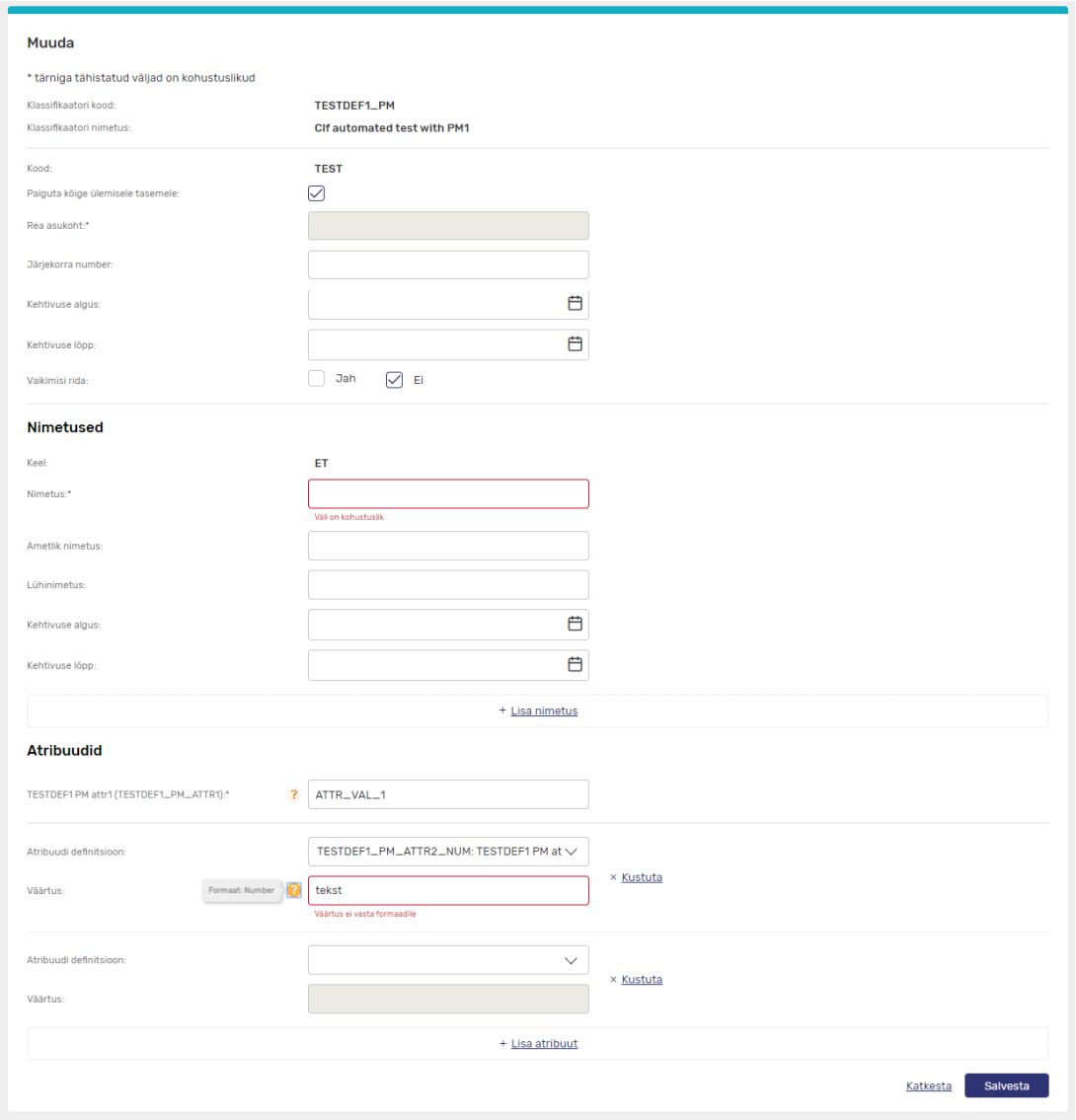
Muutmise ja lisamise vaadete arendamisel tuli kõigepealt vastu võtta otsus, millist Angulari vormide loomise tehnoloogiat kasutada. Valikuks olid *template-driven* ja *reactive* vormide loomise tehnoloogiad [12]. *Template-driven* vormid on mõeldud pigem väiksemate ja nagu nimigi viitab, vaatest tingitud vormide jaoks. Kuna vaates on väga keerukad, mahukad ning struktureeritud andmetega vormid, siis kasutusele võeti *reactive* vormid, mis on keerulisem, kuid ka paindlikum ja mõeldud andmemudelil põhinevate vormide jaoks. Üheks eeliseks *reactive* vormidel on veel andmete liikumine vaate ja mudeli vahel sünkroonselt, mis muudab vormi käitumise etteaimatavamaks.

Angularis hoitakse vormivälja väärtust ja staatust FormControl klassi instantsis. Kui vormis on mitu vormivälja, siis saab need grupeerida FormGroup instantsi, mis jälgib tema all olevate FormControli’de väärtuseid ja staatust. Igal FormControl’il on FormGroup’is nimi, mille järgi neid sealt pärida. Kui ei ole täpselt teada kui mitu välja vormis on, või kui neid saab dünaamiliselt lisada ja eemaldada, siis on mõttekas FormGroup’i asemel kasutada FormArray’sid. FormArray’s puuduvad FormControl’idel nimed ning neile pääseb ligi läbi indeksi.

Vormide koostamiseks sai loodud vormivälja jaoks „form-item“ komponent, mis kõige lihtsama tekstivälja puhul saab sisenditeks FormControl instantsi ning välja tüübi. Vormivälja pealkiri võetakse ng-content’iga komponendi märgendite vahele kirjutatud tekstist. Ng-content võimaldab Angularis kuvada komponendi märgiste vahele lisatud

sisu komponendi vaates [13]. Antud lahendus võimaldas peita vormivälja loogika ja stiilid ning erinevad väljad erinevad vaate komponendis vaid sisendite poolest.

Klassifikaatori rea muutmise vaates saab muuta rea enda andmeid ning lisada, eemalda ja muuta rea juurde kuuluvaid nimetusi ning atribuute. Vormi FormGroup struktuuriks sai loodud klassifikaatori rea iga andmerea kohta FormControl instants ning rea nimetuste ja atribuutide jaoks FormArray instantsid. Nõuete kohaselt pidid vaate avamisel vormiväljad olema eeltäidetud. Klassifikaatori rea andmete eeltäitmiseks sai kasutada FormGroup'i *patchValue* meetodit, mis päringu vastuse objektist saadud väärtused lisab FormGroup'is olevate FormControl'ide väärtusteks. Iga rea olemasolevad nimetuse ja atribuudi väärtused muudetakse *map* funktsiooniga ümber uueks FormGroup instantsiks, mis lisatakse FormArraysse.



Joonis 1. Klassifikaatori rea lisamise/muutmise vaade.

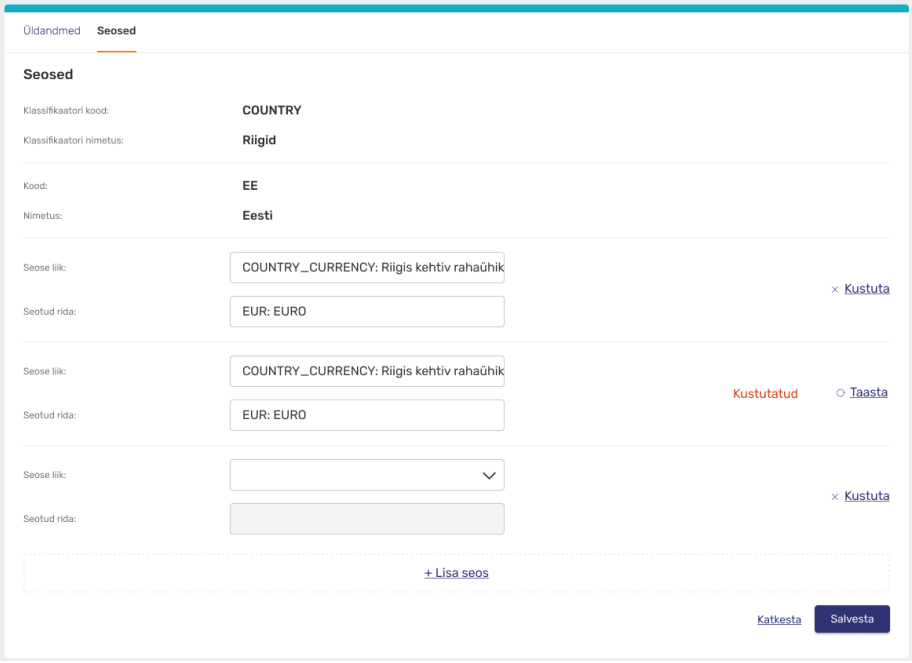
Rea enda andmed koosnesid algselt peamiselt tekstiväljadest ning seetõttu leidis autor, et andmete muutmisel saaks kasutada sarnast komponenti nagu kuvamise vaates ehk luua komponent, mis koostaks ise vormi, selle asemelt, et peaks käsitsi kõiki vormivälju vaatesse üks haaval lisamata. Komponent saab sisendiks vormi FormGroup instantsi ning komponendi vaates luuakse selle võtme-väärtuspaaridest NgForOf direktiivi kasutades iga FormContol instantsi kohta uus vormiväli. Vormivälja pealkirjaks on tõlke pige'ga muudetud FormControli nimi. Sarnaselt ka andmete kuvamise komponendile, saab sisendisse saata massiivi FormControl'i nimedega, mida vormis ei kuvataks, või siis massiivi õiges järjekorras olevate FormControl'i nimedega. Algselt oli antud komponendiga võimalik luua vaid tekstiväljadest koosevaid vorme, hiljem see täiustus vormivälja tüüpe kirjeldava objekti sisendiga, võimaldades kasutada komponenti ka vormide jaoks, mis sisaldasid kuupäeva välju.

Klassifikaatori rea muutmise vormis oli vajalik ka vormiväljade valideerimine. Vormivälja valideerimine käib Angularis lisades FormControl instantsile validaator funktsioone. Põhilised validaator funktsioonid, mis kontrollivad kohustuslikkust, maksimum pikkust jne. on Angulari sisse ehitatud, kuid neid saab luua ka ise eri otstarveteks. Selleks, et vead kasutajani jõuaks oli vajalik ka täiendada vormivälja komponenti. FormControl klassil on olemas *errors* ja *dirty* parameetrid. Kui sisestatud väärtus on vigane, siis Angular lisab vea objekti *errors* massiivi. *Dirty* väärtus näitab, kas kasutaja on muutnud välja väärtust. Lahenduseks sai *errors* massiivi pikkuse ja *dirty* väärtuse olemasolul lisada vormiväljale klass, mis muudab selle äärise punaseks ning kasutades tõlke *pipe,* kuvab *errors* massiivis olevate objektide võtmeid selgitava tekstina. *Dirty* väärtus tuleb kontrollida, kuna kasutajale ei ole kohustusliku välja puhul mõtet kuvada veateadet, kui ta ei ole veel sinna midagi sisestanud.

Muutes klassifikaatori rea nimetusi saab kasutaja lisada uusi ning muuta ja kustutada olemasolevaid nimetusi. Lahendusena on kasutatud FormArray instantsi, kuhu uue nimetuse lisamisel lisatakse FormGroup instants, milles on nimetuse andmete väljad. Olemasolevate nimetuste vormist kustutamiseks sai kasutada FormArray *removeAt* meetodit.

Sarnaselt nimetustele saab lisada, muuta ja kustutada ka klassifikaatori rea atribuute. Atribuudid koosnevad atribuudi definitsioonist ning väärtusest. Lisades uut atribuuti on väärtuse väli niikaua *disabled,* kuniks pole valitud definitsiooni väärtust. Atribuudi definitsiooni valik käib valiku nimekirjast. Selleks oli vaja kõigepealt lisada „form-item“ komponendile valiku nimekirja tüüp. Lahendusena sai integreeritud ng-select komponent. Valitud atribuudi definitsiooni määrab ära atribuudi väärtuse formaadi ehk vastavalt atribuudi definitsiooni objektile tuleb valideerida, et väärtus vastaks formaadile. Lahenduseks sai jälgida definitsiooni FormControl instantsi Observable tüüpi *valueChanges* parameetrit. Observable on tüüp, mille instantsi saab jälgida kasutades *subscribe* meetodit [15]. Lahenduses muudetakse atribuudi definitsiooni välja muutuse korral atribuudi väärtuse vormiväli aktiivseks ning lisatakse tema FormControl instantsile *setValidators* funktsiooniga spetsiaalne formaadi validaator. Angulari hea tava kohaselt tuleb ka kõik jälgimised komponendi hävitamisel lõpetada, et vältida mälu lekkeid. Atribuudi väärtuse lubatud formaati tuli ka näidata vormivälja kõrval olevas vihjemullis. Vihjemulli komponent sai loodud ng-bootstrap’i vihjemulli baasil, mille stiilid sai vastavalt antud rakendusele üle kirjutatud.

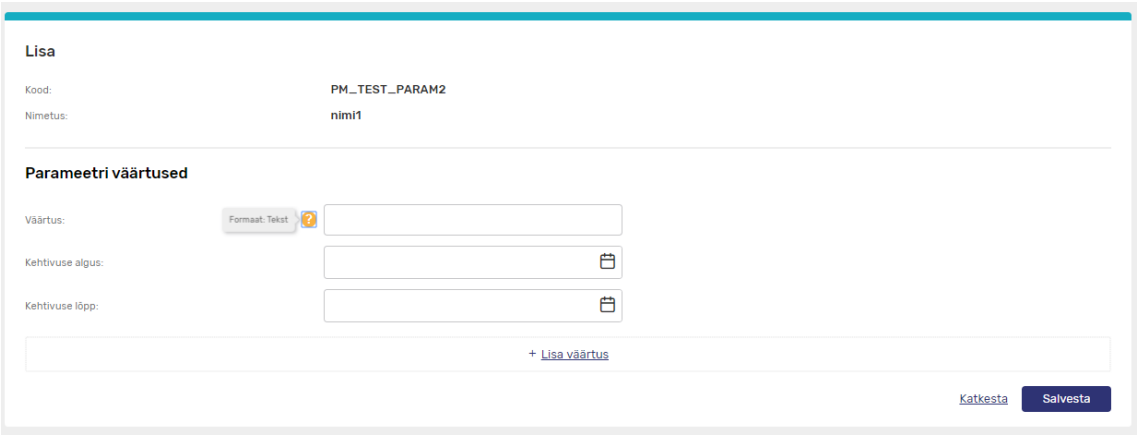
Klassifikaatori rea seoste muutmise vorm sarnaneb atribuutide omale, kasutaja saab lisada, muuta ning eemaldada klassifikaatori ridade vahelisi seoseid. Iga rea seose juures saab valiku nimekirjast muuta seose liiki ning seose rida, mille vormiväli pole aktiivne, kuni pole valitud seose liiki. Seotud rea valik käib kasutades automaatäitmise välja kasutades, mille võimalikud valikud tulenevad valitud seose liigist. Selleks oli kõigepealt vaja luua „form-item“ komponendile automaattäitmise tüüp. Lahendusena sai integreeritud NG Bootsrapi Typeahead [15]. „Form-item“ komponenti sisenditele lisandus massiiv valikute variantidest ning lisaks on võimalik sisendisse anda ka spetsiaalse loogikaga vastete filtreerimis funktsiooni. Hiljem sai veel komponenti täiustatud nii, et valikuvariandid saavad tulla ka asünkroonselt ehk sisendisse saab saata funktsiooni, mis tagastab päringu, mis omakorda tagastab kasutaja sisendile vastavad valikuvariandid. Klassifikaatori rea seose seotud rea valikuvariandid sõltuvad kasutaja valitud seose liigist. Lahenduseks sai seose liigi vormivälja muutuste korral kontrollida, kas väärtusele vastavaid seotud ridasid on juba päritud või mitte. Kui mitte siis tehakse päring võimalike vääruste saamiseks ning salvestatakse see vahemälu objekti ning võimalikud väärtused võetakse antud objektist. Lahenduse põhjuseks on, et sama seose liigiga võib olla mitu klassifikaatori rea seost ehk ning ei ole mõtet iga sama liigiga seose jaoks uut päringut tegema.



Joonis 2. Klassifikaatori rea seoste muutmise vaade.

Parameetri vaates saab kasutaja muuta parameetri väärtuseid. Vormi loogika sarnaneb klassifikaatori rea atribuutide omale, kasutaja saab lisada, eemaldada ning muuta

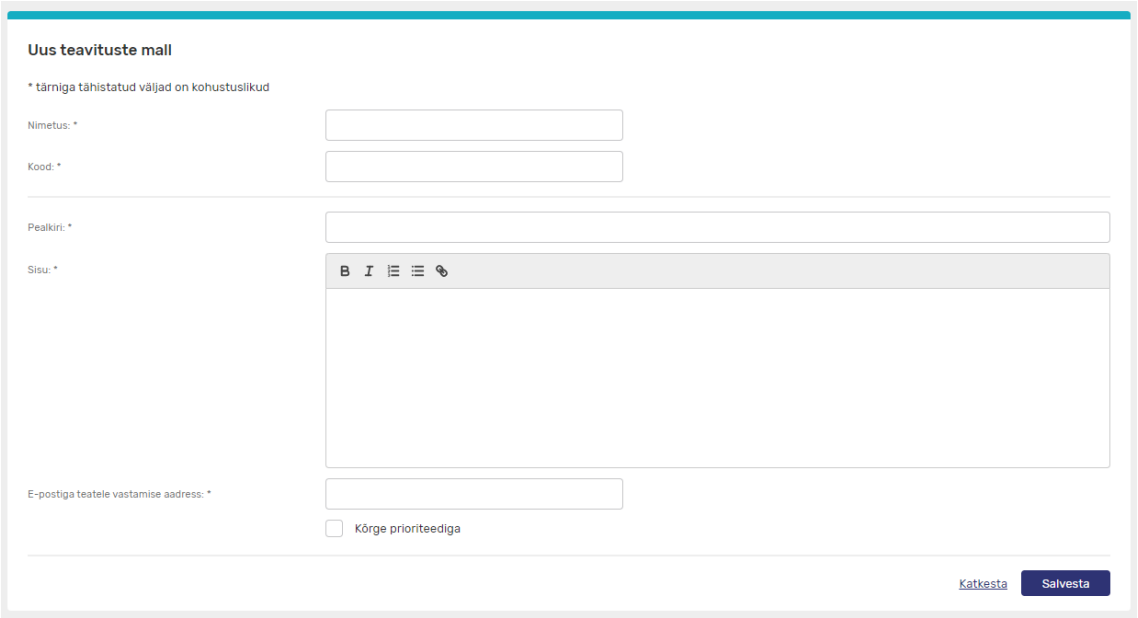
parameetri väärtuseid. Iga parameetri väärtuse vormiblokis on parameetri väärtuse väli ning kehtivuse alguse ning lõpu väli. Parameetri välja väärtuse formaati valideeritakse vastavalt parameetri definitsioonis lubatud formaadile.



Joonis 3. Parameetri vaade.

Administraatori vaadetes asuvad ka teavituste mallide vaated. Teavituste malle kasutatakse järgmisest alapeatükis juttu tulevate teavituste automaatseks täitmiseks. Teavituse malli nimistu vaates ning detailvaates uusi lahendusi ei tulnud ning need sai luua varem loodud komponentidega. Teavituse malli muutmise ja lisamises vaates sai nagu ka klassifikaatori rea muutmise/lisamise vaate puhul kasutada sama vaate komponenti ning ka siin nõudega, et muutmise vaates pidi vorm olema eeltäidetud. Vormis olev teavituse sisu väli kujutas endast tekstiredaktorit, mida varem rakenduses kasutanud ei olnud. Lahenduseks otsustati lisada see uue tüübina „form-item“ komponendi alla, kasutades mõnda valmis tehtud tekstiredaktori komponenti ning seejärel konfigureerida ning muuta selle stiilid rakendusele vastavaks. Esialgselt tundus sobivat CKEditor 5, kuid peale integreerimist selgus, et see ei täida Internet Explorer 11 toe nõuet, mistõttu tuli see ngx-quill'i peale ümber vahetada [17]. Ngx-quill'i puhul sai takistuseks oli tõlgete nõue, ehk komponendi sees ei tohi olla *hard-coded* sõnet, mis ngx-quill komponendis eksisteerisid lingi lisamise *widget*'i nuppudes, mis tekkisid sinna CSS'i *content* atribuudi väärtusest. Komponendis on aga olemas *onEditorCreated* väljund, mis peale uue redaktori instantsi loomist emiteerib selle. Lahenduseks sai peale uue instantsi loomist selle CSS'is lingi lisamise vidina nuppude *content* atribuudi väärtus tühjaks muuta ja selle asemel lisada nendele *innerHTML*'i väärtus. Siit tekkis aga järgmine probleem: ühel nupul võis olla kaks väärtust, sõltuvalt sellest kas parasjagu lisati linki või muudeti seda ning see avaldus vaid elemendi klassist. Lahenduseks sellele probleemile sai MutationObserver'i liidese kasutuselevõtt. MutationObserver'i instants

võimaldab jälgida DOM'is toimuvaid muutuseid nende hulgas ka elemendi klasse, mille järgi sai otsustada, millist väärtust hetkel nupul kuvada [18].



Joonis 4. Teavituse malli lisamise/muutmise vaade.

Nõude kohaselt tuli peale malli salvestamist kasutaja ümber suunata nimistu vaatesse ning kuvada teadet eduka salvestamise kohta. Teavituse komponendi sai Haridusportaalist üle tuua, kuid sellele tuli teha mõningased täiendused. See oli varasemalt lahendatud järgmiselt: teate komponent tellib teadete teenuses olevat Subject tüüpi teate välja, mis uue teate korral väljastab teate objekti. Kui teate komponendi id vastab väljastatud objekti id'le siis komponent kuvab antud teavitust. Probleem tekkis kui teadet tuli kuvada mõnes teises vaates, kuna teenuses uue teate loomise hetkel teate komponenti veel ei eksisteeri. Uueks lahenduses sai teenuses teate tüübiks muuta BehaviourSubject'iks, sest erinevalt Subject tüübist hoiab see ka hetkeväärtust. Teate komponenti sai muudetud nii, et see initsialiseerides kontrollib teenusest kas tema id'ga teade on juba olemas.

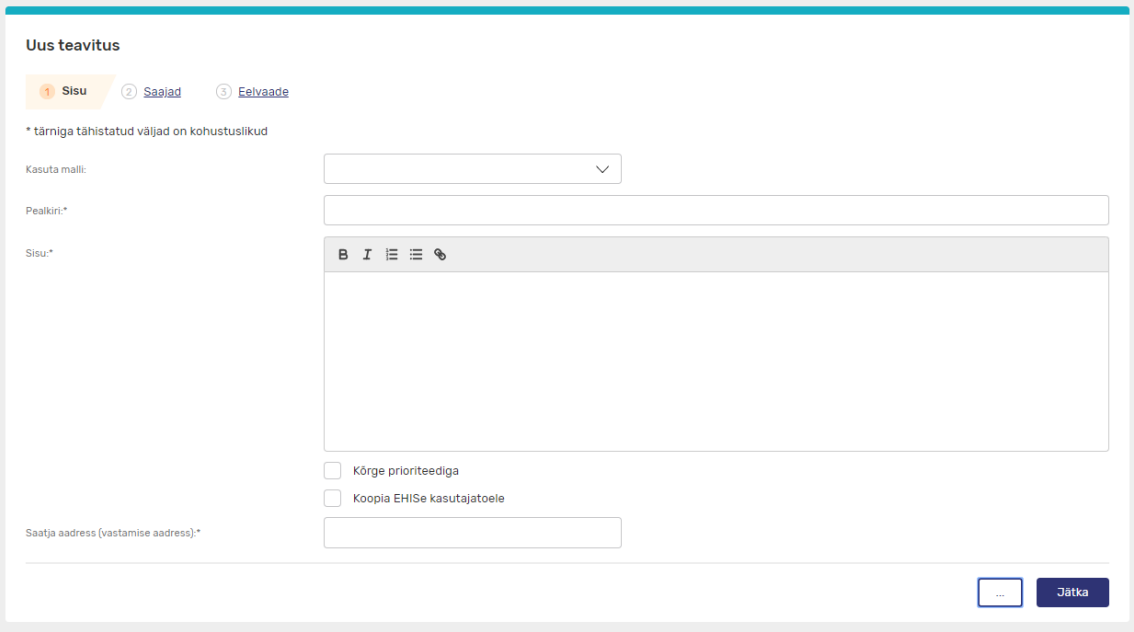
3.2 Teavituste moodul

Teavituste mooduli all saab infosüsteemi sisse loginud kasutaja vaadata temale saadetud teavitusi ning teatud privileegidega kasutajad saavad ka saata teavitusi. Mooduli kasutajaliides koosneb saadetud teavituste ja mustandite nimistu ning detailvaatest, teavituse koostamise vaatest ja saabunud teavituste nimistu ning detailvaatest.

Mooduli kasutajaliidese arendus algas saadetud teavituste ja mustandite vaadetest. Saadetud teavituste ja mustandite nimistu vaade on disainilt vaadates, mis eraldatud sakkidega. Siiski tundus parem lahendus luua kaks eraldi vaate komponenti, millel on erinev aadress. Põhjuseks, et saadetud teavituste sakki avamisel ei oleks vaja teha ka mustandite päringut ning nad oleks aadressiga eristatavad. Sakkide komponent sellist lahendust veel ei võimaldanud ja seda tuli täiustada. Lahenduseks oli kasutada alam-navigeerimist. Navigeeritavate blokkide puhul sai lahenduseks aktiivse saki sisu näitamise asemel kuvada RouterOutlet'i väljund. RouterOutlet on Angularis direktiiv, mis kuvab ruuteri parasjagu aktiivse lingile vastavat sisu [19]. Saki nuppude asemele said saki lingid, mis viitavad õigele vaatele. Saki komponendis aktiivse saki kontroll sai muudetud hetke aadressi järgi.

Saadetud teavituse ja mustandi detailvaated koosnesid peamiselt teavituse andmetest ning need sai luua olemasolevatest komponentidest.

Teavituse koostamise vaade koosnes kolmest sammust. Esimesel sammul on teavituse vorm, teisel sammul teavituse saajate lisamine ning viimasel sammul teavituse eelvaade. Kõigepealt tuli luua sammude komponent, lahendus sai sarnaneb sakkide omale, erinedes selle poolest, et samme peab saama vahetada ka komponendi väliselt. Lahenduseks sai aktiivse sammu kontrollimine läbi indeksi. See võimaldas vaatekomponendis aktiivset indeksi muuta ning selle seejärel sammude komponenti saata. Sammu komponendi siseselt sammu muutes emiteerib see uue aktiivse indeksi, et vaate komponent püsiks sünkroonis.



Joonis 5. Teavituse koostamise vaade.

Teavituse sisu vorm sarnaneb teavituse malli vormile. Vormi sisu saab täita käsitsi või valides valiku nimekirjast teavituse malli. Valiku tehes sooritatakse päring ning vastest saadud teavituse malli andmetega täidetakse FormGroup *patchValue* meetodiga vormiväljad.

Teavituse koostamise vaate teisel sammul saab kasutaja valida teavituse saajad. Selleks on valikuid mitmeid: valik asutuse alusel, asutuse tüübi alusel, importides saajad failist või importides asutus failist . Üldjoonelt on lisamise protsess sarnane, eri viisidel valitud isikud kuvatakse modaalis lehekülgede vahel jaotatud tabelina, iga isiku rea ees on märkeruut, mille väärtuse järgi otsustatakse, kas isik lisatakse saajate hulka või mitte. Tabeli päises on võimalik valida ka märkeruutu, mille väärtus muudab kõigi kasutajate märkeruute. Kui kasutajad kinnitatakse, lisatakse nad saajate hulka ning nad kuvatakse saajate tabelis.

Lisades saajad asutuse või asutuse tüübi valiku alusel, avatakse modaal milles saab valida kas siis asutuse või asutuse tüübi ning ka kasutajate tüübi. Asutuse valimine toimub automaattäitmis väljaga, mis alatest teisest tähemärgist alustab vastete pakkumist. Täidetud andmete põhjal tehakse kasutajate päring ning edasine protsess jätkub nagu eelnevalt kirjeldatud.

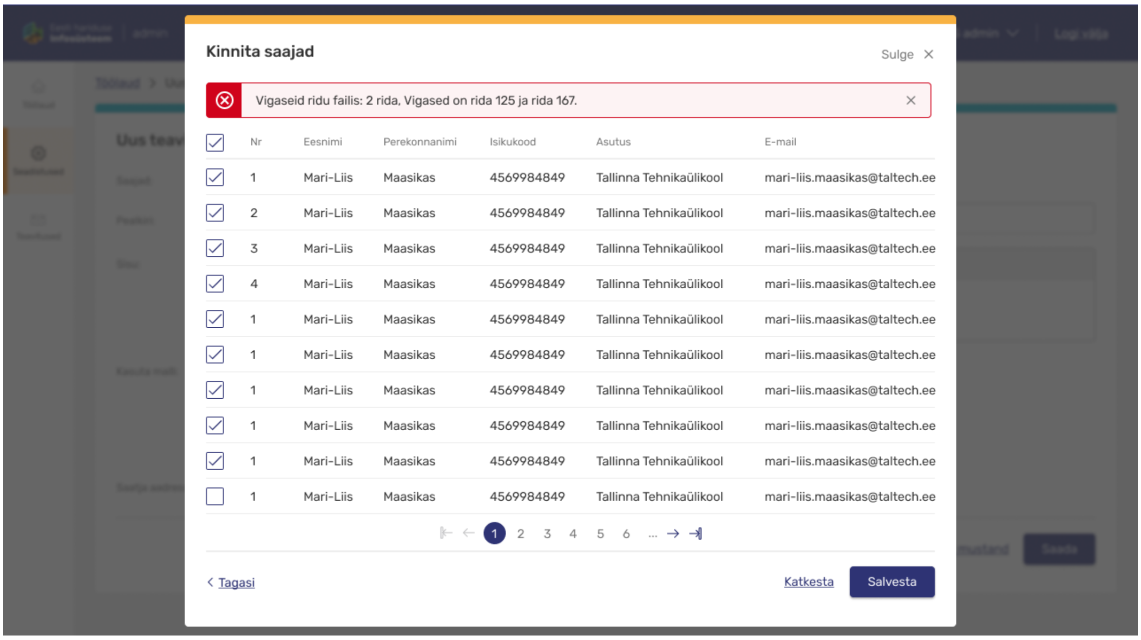
Importides saajad failist, laetakse üles kasutajatega CSV fail. Selleks oli kõigepealt vaja luua faili üleselaadimise komponent. Disainilt kujutab komponent ennast välja, millel

asuval nupule vajutamisel saab valida faili või siis saab lohistada faili väljale. Nupu lahenduseks sai vajutusel peidetud faili tüüpi vormi välja vajutamine. HostListener dekoraatoriga jälgib komponent ka lohistamise ning kukutamise sündmuseid. Lohistamise korral lisatakse vastavad stiilid ning faili kukutamise korral alustatakse faili üleslaadimist. Komponendile saab sisendisse saata ka faili formaadi tüübid, kui lohistatud fail tüüpidele ei vasta, siis muutetakse stiile kasutajat teavitamaks ning faili üles ei laeta. Kui faili formaat sobib, siis emiteeritakse fail komponendi väljundist ning vaate komponent saab sellega edasi tegeleda. Saajate impordi vaate korral hakkab seejärel toimuma kasutajate failist lugemine ning ka faili sisu valideerimine. Üles laaditav CSV fail peab olema kindla struktuuriga ning kui see ei vasta sellele, siis faili lugemine lõpetatakse koheselt, kuvatakse kohane veateade ning ühtegi adressaati ei lisata. Antud vaates valideeritakse ka iga rida eraldi, isikukoodi olemasolu on kohustuslik ning kui mõnel real on see puudu, siis rida jäetakse vahele, ning rea indeks lisatakse vigaste ridade massiivi, mis peale kõikide isikute failist lugemist veateates välja kuvatakse. Kui vigaseid ridu on üle kümne, siis tsükkel lõpetatakse ning adressaatide kinnitamise tabelise lisatakse vaid isikud, mis selle hetkeni failist välja loeti. Samuti kontrollitakse iga rea juures, et antud isikukoodiga adressaati juba lisatud ei oleks. Kui isik on juba adressaatide hulga siis tema isikukood lisatakse olemasolevate adressaatide massiivi ning peale faili lugemist kuvatakse info teade, et antud isikukoodidega kasutajad olid juba saajate hulgas.

Saajate määramisel asutustuste impordil failist laetakse üles asutustega CSV fail. Ka selle faili korral kontrollitase struktuuri. Isikute kätte saamine toimub järgmiselt: iga faili reas on asutuse registrikood, mille alusel sooritatakse asutuste päring. Sama registrikoodiga asutusi võib olla rohkem kui üks ning iga asutuse kohta sooritatakse asutuse kasutajate päring. Kuigi töö autori arvates pole antud arhitektuuriline lahendus optimaalne ning protsess oleks palju kiirem kui kasutaja otsitaks andmebaasi tasemel, siis lahendus tuli teha nende nõudmiste järgi. Lahenduses kasutatakse RxJS teegi *switchMap* meetodit, et muuta asutuse päringu observable kasutajate päringu observable'ks, mis lisatakse päringute massiivi. *SwitchMap* meetodi kasutus võimaldab ära hoida päringule tellimist teise päringu tellimise sees, mida loetakse Angularis halvaks tavaks. Kui kõik faili read on loetud, siis tehakse RxJS'i teegi *forkJoin* funktsiooniga massiivis olevad päringud. *ForkJoin* funktsiooni kasutatakse selleks, et päringute vastused tuleksid samaaegselt. Antud funktsiooni kasutades tuleb meeles pidada, et kui vähemalt üks *ForkJoin*'i sisendisse saadetud päringutes tagastab vea, siis tagastatakse viga, et seda vältida, tuli igat

päringule RxJS teegi *catchError* funktsiooniga vea püüdmist teha [20]. Nagu ka teiste saajate lisamise meetodite puhul, lisatakse päringust saadud isikut, saajate lisamise tabelisse.

Saajate lisamise tabelis sai esialgse lahendusena kasutatud FormArray'd, kuhu iga isiku kohta lisati FormControl instants, mille väärtusest sõltub kas isik lisatakse saajate hulka või mitte. Probleem tekkis aga kui vajutada märkeruutu, mis muudab kõigi isikute ees oleva märkeruudu väärtust. Suure isikute arvu korral võttis kõikide FormControl instantside väärtuste muutmise protsess aeglasemas arvutis aega mitukümmend sekundit. Lõpplahenduseks jäi kasutajale tekitada illusioon, et ülem märkeruudu väärtus muudab kõikide märkeruutude väärtust. Reaalsuses muudetakse vaid nende isikute FormControl instantside väärtust, mis on leheküljel, millel asub kasutaja. Ükshaaval muudetud märkeruutude indeksid lisatakse massiivi ning vastavalt ülem märkeruudu väärtusele antud indeksitel asuvad isikud lisatakse või ei lisata saajate hulka.



Joonis 6. Teavituse saajate lisamise tabel.

Teavituse lisamise kolmandas sammus ehk teavituse eelvaates kasutatakse teavituse andmete kuvamiseks interpolatsiooni, milles lihtsalt kuvatakse esimesel sammul oleva vormi väärtused.

Saabunud teavituse nimistu vaade sarnanes teistele nimistu vaadetele ning selle sai luua eksisteerivatest komponentidest. Samuti ei olnud vaja luua uusi lahendusi saabunud teavituse detailvaate jaoks.

Üheks osaks teavituse moodulist oli veel sisse loginud kasutajale teavituse menüüpunkti kõrval lugemata teavituste arvu kuvamine. Lahendus oli uus teenuse loomine. Teenusel on meetod, peale mille välja kutsumist, näiteks sisse logimisel, hakkab teenus intervalli tagant tegema päringuid lugemata teavituste otsa. Päringu vastusel uuendatakse BehaviourSubject tüüpi lugemata teavituste numbrit, mida teenust kasutavad komponendid saavad jälgida. Kui teenus hävitatakse siis lõppeb ka päringute tegemine.

3.3 Avalikud vaated

EHIS2e 1.etapi avalike vaadete alla kuuluvad koolide, õppekavade ning tegutsemisõiguste vaated. Kasutaja saab otsida koole, õppekavu ja tegutsemisõiguseid ning kuvada nende andmeid.

Kõige suuremaks erisuseks antud vaadete juures oli tagarakenduse poolne Elasticsearch'i kasutuselevõtt. Elasticsearch võimaldab REST arhitektuuril otsida ning sorteerida andmeid [21]. Selleks, et kasutamist ühtlustada ning lihtsustada, sai loodud uus teenus. Teenus sisaldab endas erinevaid Elasticsearch'i pakutavaid päringu meetodeid sisaldavaid funktsioone, mille abil sai edaspidi päringuid luua.

Avalikke nimistuvaated sarnanevad varem loodud nimistuvaadetega, koosnedes filtritest ning tulemuste ridadest. Erinevalt teistest nimistuvaadetest on filtrid aga mahukad ning kujutavad endast ka automaattäitmis välju ning ka valikuvälju, mille väärtused tulid enamuselt klassifikaatoritest.

Üheks nõudeks filtrite puhul oli nende väärtuste kajastumine aadressiribal ehk kasutaja peab saama URL'i kopeerides jagama leitud tulemusi. Lahendusena sai filtri komponendile loodud abifunktsioonid, mis filtri väärtuste põhjal lisavad väärtused URL'ile ning URL'i parameetrite järgi lisavad filtrite väärtused.

Avalike vaadete nimistuvaated suunavad kõik õppeasutuse vaatesse. Kui kasutaja tuli õppekavade või tegutsemiseõiguse nimistu vaatest, siis kuvatakse õppeasutuse vaates vastavalt õppekavade või tegutsemisõiguste sakki. Üheks nõudeks oli, et olles tulnud õppeasutuste nimistuvaatest, kus filtreerimise tulemusena on 20 või vähem tulemust, peab kasutaja saama ilma nimistu vaatesse naasmata edasi-tagasi või valiku nimistu kaudu nende vahel navigeerida. Lahenduse sai loodud komponent, mis sisenditeks saab andmete massiivi, algselt valitud andme objekti, URL'i prefiksi ning objekti parameetri mille järgi

navigeerimine toimub. Kasutades edasi-tagasi nuppe või valiku nimistut, navigeeritakse URL'i prefiksi ja sisendina objekti parameetri väärtuse järgi.

3.4 E-tunnistused

EHIS2 osaks on ka E-tunnistuste moodul, mis loodi EHISes olemasolevale haridust tõendavate dokumentide alamregistrile lisaks. Tunnistuse omanik või tunnistuse kasutaja näeb temale ligipääsetavaid tunnistusi Haridusportaalis, mis kasutab E-tunnistuste mooduli teenuseid. Koolide poolt kasutatav E-tunnistuste mooduli kasutajaliides asub aga EHIS2s. Antud vaadete hulka kuuluvad tunnistuste nimistu vaade, tunnistuste eelvaade ning tunnistuse andmete vaade.

Tunnistuste nimistu vaates kuvatakse tunnistusi, mille on väljastanud kool, mille rollis kasutaja parasjagu asub. Nimistu vaates saab ka alla laadida tunnistuste ära kirjad ning samuti saab suunduda tunnistuse andmete vaatesse või tunnistuste eelvaatetesse. Tunnistuste eelvaatesse suundumiseks või alla laadimiseks valib kasutaja kas siis märkeruutudest iga tunnistuse eraldi, mida ta soovib alla laadida või vaadata, või siis valida ülem märkeruudu, mis valib kõik tunnistused. See sarnanes peatükis 3.1 kirjeldatud teavituse saajate lisamise tabelile, tänu millele sai seal loodud lahenduse üle võtta.

Tunnistuste eelvaate vaade koosneb tunnistuses ning „järgmine" ja „eelmine" nuppudest, mille kaudu saab kasutaja liikuda nimistu vaates valitud tunnistuste ja nendega kaasas käivate hinnete lehtede vahel. Vaate arendamine algas E-Tunnistuse komponendi loomisega. Kõige keerulisemaks antud ülesande puhul oli luua komponent viisil, et see säilitakse proportsioonid igas suuruses. Nimelt kui mujal on kasutatud relatsioonilisi *rem* ühikuid, siis tunnistus on loodud täpselt disainis võetud pikslite väärtustega, et suhe pikkuse, laiuse, teksti suuruse, reavahede jne. vahel püsiks pidevalt sama. Keerukus seisneb selles, et komponent peab ise ikkagi dünaamiline olema. Lahenduseks sai tunnistuse ümbritsemine konteineriga, mis võtab oma suuruse talle antud ruumi järgi. Tunnistus konteineri sees on absoluutse positsiooniga ning selle suurus arvutatakse ning muudetakse CSS'i *transform: scale* parameetri väärtust iga kord kui komponent luuakse või kui brauseri akna suurust muudetakse. Ehk kui tunnistuste eelvaate vaates oli üheks nõudeks tunnistuse mahtumine täies suuruses ekraanile, siis sai tunnistuse komponenti ümbritsevale konteinerile anda brauseri vaateava suhtes relatiivse ühiku *vh*'ga kõrguse

ning tunnistuse komponent arvutab ise kui palju ta peab skaleeruma, et ennast konteinerisse mahutada.

E-Tunnistuse eelvaate jaoks tuli veel ka luua tunnistuse juurde käiva hinnetelehe komponent. Hinneteleht puhul ei ole A4 formaadi hoidmise nõuet, mille tõttu oli komponendi loomine palju lihtsam ning see koosneb peamiselt tekstilõikudest ning tabelist.

Veel üheks väljakutseks eelvaate puhul oli nõue, et „järgmine“ ja „eelmine“ nupp peavad püsima samal positsioonil, selleks et kasutaja saaks dokumentide vahel liikuda ilma hiirt liigutamata. Esmaseks lahenduseks sai nuppude puhul kasutada *fixed* positsiooni. Probleemiks antud lahenduse juures oli see, et *fixed* positsiooniga elementide kohal kerimine ei kandu üle *parent* elementidele. Ehk kuigi see ei ole probleemiks tunnistuste puhul, mis pidid täismahus ekraanile mahtuma, siis pikemate hinnete lehtede puhul peaks kasutaja hiirega nupu pealt lahkuma, selleks et ta saaks hinnetelehe lõppu kerida, mis tõttu kaotaks esialgne nõue oma mõtte. Paranduseks sai antud vaate korral keritavale elemendisse hiire kerimise sündmuse kuulaja lisamine. Kui sündmuse sihtmärgiks on üks nuppudest, siis keritava elemendi *scroll*'i väärtust muudetakse.

Viimaseks vaateks E-Tunnistustes jäi E-Tunnistuse andmete kuvamine. Vaade koosneb akordionitest, milles kuvatakse tunnistuse omaniku ja muid tunnistusega seonduvaid andmeid, tunnistuse ligipääse ning tunnistuse vanemaid versioone, mida saab avada ja modaalis kuvada. Akordionite komponent oli juba varasemalt Haridusportaalis teostatud ning selle sai mõningate stiili muudatustega sealt üle võtta.

4 Hinnang tulemusele

Rakenduse funktsionaalsuse testimine toimus arendusprotsessi käigus töö autori tiimis oleva testija poolt ning rakenduse üleandmisel kliendile selleks väljavalitud ametnike poolt, mis töö kirjutamise hetkeks on edukalt läbitud. Seetõttu ei keskenduta antud peatükis funktsionaalsusele, vaid töö autor annab oma kvalitatiivse hinnangu töö eesmärkide saavutamise kohta.

Töö peamiseks eesmärgiks oli luua rakendus viisil, mis lihtsustaks ja kiirendaks arendusprotsessi järgmistes projekti etappides. Angulari rakenduse puhul avaldub see peamiselt taaskasutatavate komponentide loomisest. Rakenduse arenduse käigus sai loodud 50 taaskasutatavat komponenti ning 20 teenust. Autori hinnangul oli lahenduse efekte tunda juba esimeses etapis, eriti just nimistu ja detailvaadete juures, kus vaate põhi/komponendid püsisid valdavas enamuses samad ning muudatused tulenesid peamiselt vaid erinevatest andmekoosseisudest.

Teiseks eesmärgiks oli projekti hallatavuse säilimine. Üheks probleemiks suurte veebirakenduste loomisel on stiilide haldus, kus paljude vaadete stiilid kasvavad ülemäära suureks. Ka selle probleemi lahenduseks oli taaskasutavate komponentide loomine ja kasutamine. Kasutajaliidese 35 loodud vaate puhul oli vaja luua eraldi stiilid vaid 21le vaatele ning mis enamuses on vaid mõne realised. Kokku on vaadete stiili failides 530 rida koodi ehk keskmiselt ühe vaate umbes 15 rida stiilide koodi, mis on autori arvates väga hea tulemus.

4.1 Täiendused tulevikuks

Sellegipoolest vajavad autori hinnangul veel mõned komponendid täiendamist. Ühe näitena saab tuua ka antud töös mitmeid kordi mainitud „form-item“ komponendi näol, kuhu sai järkjärgult, vastavalt vajadusele pidevalt lisatud uusi vormivälja tüüpe ning nendega seotud loogikat. Kuigi loogika on vaate failide eest peidetud, on autori hinnangul seda saanud ühte komponenti liiga palju ning muutnud selle raskemini hallatavaks. Samuti on antud komponendi stiilid käest läinud ning eri tüüpide stiilid on ajapikku läbisegi läinud. Autori arvates oleks heaks lahendiks kasutada pärilust, kus lisaloogikaga vormivälja tüübid pärinevad baas „form-item“ komponendist. Teise muudatusena oleks veel hea muuta mõningaid sisu projektsiooni komponente nagu näiteks modaali

komponent, mis hetkel kasutav projekteerimiseks ng-content'i. Ng-content kuvab kogu komponendi märgiste vahele lisatud sisu ja kuigi tegu pole kuidagi vale lahendusega, siis antud lahenduse korral initsialiseeritakse märgiste vahel sisse saadetud komponendid koos vanem komponendiga, isegi kui neid ei ole veel vaatesse renderdatud. Antud käitumine võib tekitada ootamatusi ja seetõttu on autori hinnangul mõistlik üleminna mõnele alternatiivsele lahendusele nagu näiteks template outlet'i kasutus [22].

5 Kokkuvõte

Antud töö eesmärgiks oli Haridus- ja Teadusministeeriumi poolt tellitud EHIS2 projekti 1.etapi kasutajaliidese loomine. Kuna tegu oli esimese etapiga mitmetest, siis pidas töö autor oluliseks valida sobiv meetod, mis lihtsustaks ning kiirendaks järgmiste etappide arendust ja et rakendus oleks ka selle kasvades hallatav.

Rakenduse arenduse meetodiks sai valitud taaskasutatavate komponentidest lähtuv arendus. Kasutajaliidese arenduse käigus sai loodud poolsada taaskasutavat komponenti, millede kasutamine kiirendab tulevast arendusprotsessi. Samuti parandab see koodi hallatavust, vähendades korduvaid koodiridu vaadete stiilide ning mallide failides. Seega saab autor väita, et antud töö eesmärgid said edukalt täidetud. Sellegipoolest on peatükis 4.1 toodud välja veel mõned võimalikud täiustused tulevikuks.

Töö kirjutamise hetkeks on EHIS2 kliendile üleantud ja selle funktsionaalsus Haridus- ja Teadusministeeriumi poolt määratud ametnike poolt testitud ehk saab ka kindlalt väita, et see vastas kliendi poolt esitatud nõudmistele.

Kasutatud kirjandus

[1] Haridus- ja Teaduministeeriumi seminar [WWW] http://ern.ee/files/seminar19062017/20170619-htm.pdf (30.04.2020)

[2] What is Scrum [WWW] https://www.scrum.org/resources/what-is-scrum (30.04.2020)

[3] Vue Documentary, 2020, [WWW] https://youtu.be/OrxmtDw4pVI?t=280 (30.04.2020)

[4] Elliot, E., Top JavaScript Frameworks and Topics to Learn in 2020 and the New Decade, 2020, [WWW] https://medium.com/javascript-scene/top-javascript-frameworks-and-topics-to-learn-in-2020-and-the-new-decade-ced6e9d812f9 (30.04.2020)

[5] React: The Virtual DOM [WWW] https://www.codecademy.com/articles/react-virtual-dom (30.04.2020)

[6] Srivastav, A., Benchmarking Angular, React and Vue for small web applications, 2019 [WWW] https://blog.bitsrc.io/benchmarking-angular-react-and-vue-for-small-web-applications-e3cbd62d6565 (30.04.2020)

[7] Fluin, S., Version 9 of Angular Now Available — Project Ivy has arrived!, 2020 [WWW] https://blog.angular.io/version-9-of-angular-now-available-project-ivy-has-arrived-23c97b63cfa3 (30.04.2020)

[8] Sass documentation [WWW] https://sass-lang.com/documentation (30.04.2020)

[9] Angular architecture guide [WWW] https://angular.io/guide/architecture (30.04.2020)

[10] Angular documentation: NgForOf [WWW] https://angular.io/api/common/NgForOf (30.04.2020)

[11] Angular documentation: Pipes [WWW] https://angular.io/guide/pipes (30.04.2020)

[12] Angular forms overview [WWW] https://angular.io/guide/forms-overview (30.04.2020)

[13] Hicks, J., What Is Ng-Content?, 2018, [WWW] https://medium.com/@joshblf/wtf-is-ng-content-8382b2a664e1 (30.04.2020)

[14] Ng-select [WWW] https://github.com/ng-select/ng-select (30.04.2020)

[15] Angular documentation: Using obervables to pass values [WWW] https://angular.io/guide/observables (30.04.2020)

[16] Typeahead [WWW] https://ng-bootstrap.github.io/#/components/typeahead/api (30.04.2020)

[17] Ngx-quill [WWW] https://www.npmjs.com/package/ngx-quill (30.04.2020)

[18] MutationObserver [WWW] https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver (30.04.2020)

[19] Angular documentation: RoutrOutlet [WWW] https://angular.io/api/router/RouterOutlet (30.04.2020)

[20] Rxjs documentation [WWW] https://rxjs.dev/guide/overview (30.04.2020)

[21] Elasticsearch [WWW] https://www.elastic.co/elasticsearch/ (30.04.2020)

[22] Pietraszko, M., Angular: Why you should consider using template outlet instead of content projection., 2017, https://medium.com/@pietmichal/angulars-content-projection-trap-and-why-you-should-consider-using-template-outlet-instead-cc3c4cad87c9 (30.04.2020)