TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Väitlusmeistrivõistluste mobiilirakenduse loomine Ionic raamistikus koos seda administreeriva veebiliidesega

bakalaureusetöö

Üliõpilane: Karl Mäe
Üliõpilaskood: 135190
Juhendaja: Roger Kerse, MSc

Tallinn
2017

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.

(*kuupäev*) (*allkiri*)

Annotatsioon

Antud bakalaureusetöö eesmärgiks on väitlusvõistluste tarbeks luua Ionic [1] raamistikul põhinev hübriidrakendus. Hübriidrakendus peab toimima nii Androidil kui ka brauseris. Ionicu puhul on tegu teegiga, mis koosneb mobiilile optimeeritud HTML-ist, CSS-ist ja Javascriptist. Täpsemini CSS-i laiendusest Sassist [2] ja Javascripti raamistikust Angularist [3].

Praktilise osa puhul on tegu aastal 2017 Tallinna Tehnikaülikooli poolt korraldavate Euroopa ülikoolide väitlusmeistrivõistluste tarbeks arendatava mobiilirakendusega. Rakendus peab võimaldama võistlejatele edastada reaalajas infot nende järgmise vooru ning üldiste teadaannete kohta, hõlbustama orienteerumist Tallinna Tehnikaülikooli hoonetes ja Tallinna linnas.

Rakendus jaguneb kolmeks suuremaks komponendiks:

- Mobiilirakendus
- Administraatoriliides veebis
- REST protokolli [4] kasutav server, mis suhtleb rakenduse ja administraatori liidesega

Administraatori liidese ülesandeks on reaalajas teadete edastamine ning haldamine, asukohtade märkimine kaardil ning vooru toimumiskohtade lisamise funktsionaalsus. Peamiseks probleemiks on infovahetus rakenduse erinevate komponentide vahel.

Töö tulemuseks on funktsioneeriv mobiilirakendus seda administreeriva liidesega ning toetava serveriga. Komponentide vahel peab olema tagatud toimiv suhtlus ning oluline on, et rakendust saaks edukalt kasutada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 48 leheküljel, 10 peatükki, 9 joonist.

Abstract

This bachelors thesis aims to build a hybrid mobile application for debating competition based on Ionic framework [1]. Hybrid application has to function on Android and in a browser. Ionic is a framework, which consists of mobile optimized HTML, CSS and Javascript. Specifically Sass [2], which is an extension for CSS and AngularJS [3] which is a Javascript library.

The practical part consists of developing a mobile application for European championships of universities in debating, which takes place in 2017. The application must enable its users to have live access to information about their competition, overall announcements and also help orientating in Tallinn University of Technology facilities and in Tallinn city.

The application consists of three large main components:

- The front-end of mobile application
- The administrator's interface on web
- Back-end server which communicates with the application and panel, using REST protocol [4]

The aim of the administrator's interface is to have an overview and control the real time messages sent to the users and administer venues and round start locations. The main problem is the exchange of information between different components of the application.

The result of the development is a functioning mobile application with the administrator's interface and a supporting server. There needs to be a communication between the separate components so that the mobile application could be successfully used.

The thesis is in Estonian and contains 48 pages of text, 10 chapters, 9 figures.

Lühendite ja mõistete sõnastik

API	***Application Programming Interface*** Rakendusliides, programmiliides. Reeglistik valmisprogrammiga suhtlemiseks. [5]
HTML	***HyperText Markup Language*** Standardne keel veebilehtede ja veebirakenduste loomisel. [6]
CSS	***Cascading Style Sheets*** Keel, mida kasutatakse HTML-i failide kujundamisel. [7]
AJAX	***Asynchronous JavaScript and XML*** Veebirakenduste loomise meetod, kus andmevahetus veebilehitseja ja veebiserveri vahel saab toimuda asünkroonselt, veebilehte uuesti alla laadimata. [8]
HTTP	***Hypertext Transfer Protocol*** Protokoll, mis on vundamendiks andmevahetuseks internetis. [9]
DOM	***Document Object Model*** Reeglid/mudel objektide esitamiseks veebilehel. [10]
JSON	***Javascript Object Notation*** Lihtsustatud andmevahetusformaat. [11]
REST	***Representational State Transfer*** Tarkvaraarhitektuuri stiil, mis pakub andmete kuvamist ja manipuleerimist läbi nende kujutamise veebiressursside ja seisunditeta operatsioonide kaudu. [4]
DAO	***Data Access Object*** Abstraktne liides suhtluseks rakenduse ja andmebaasi vahel. [12]

CLI	***Command-line Interface*** Käsurealiides, mis võtab vastu klaviatuurilt sisestatud käske üks rida korraga. [13]
URL	***Uniform Resource Locator*** Viide veebiressursile, mis täpsustab selle asukohta internetivõrgus. [14]
SQL	***Structured Query Language*** Relatsioonilistes andmebaasides kasutatav struktuuripäringukeel. [15]
GPS	***Global Positioning System*** Satelliitidel põhinev positsioneerimissüsteem. [16]

Sisukord

Jooniste nimekiri

1. Sissejuhatus

1.1 Taust ja probleem

Käesoleva bakalaureusetöö eesmärgiks on luua väitlusvõistluste tarbeks Ionic raamistikul põhinev hübriidrakendus ning seda administreeriv veebiliides. Hübriidrakendus võimaldab tõhusat funktsioneerimist mitmel erineval platvormil ning seda kõike ühel koodibaasil.

Praktilise osa puhul on tegu aastal 2017 Tallinna Tehnikaülikooli poolt korraldavate Euroopa ülikoolide väitlusmeistrivõistluste tarbeks arendatava mobiilirakendusega. Rakendus võimaldab võistlejatele edastada reaalajas infot nende järgmise vooru ning üldiste teadaannete kohta, hõlbustab orienteerumist Tallinna Tehnikaülikooli hoonetes ja Tallinna linnas.
Arendust alustati 2016. aasta veebruaris. Väitlusvõistluste toimumisaeg on 2017. aasta august, mis on ühtlasi ka rakenduse viimane valmimistähtaeg.

1.2 Ülesande püstitus

- Väitlusvõistluseid puudutava info reaalajas kasutajasõbralik kuvamine mobiilirakenduses.
- Välistudengitele võistluste toimumiskohas ja Tallinna linnas orienteerumist abistav mobiilirakendus.
- Võistluste ajal mobiilirakendust haldava isiku tööd abistav administraatori veebiliides.

1.3 Metoodika

Rakenduse tegemisel kasutab autor väledat arendusmetoodikat. Kogu töö on jagatud väikesteks kasutuslugudeks, mille edenemise üle on autoril Pivotal Tracker [17] keskkonnas ülevaade. Samaaegselt tegeleb autor nii rakenduse disaini, arenduse kui ka testimisega.

1.4 Ülevaade tööst

Töö teoreetilises osas analüüsib autor ehitatava väitlusvõistlusterakenduse toel lähemalt hübriidrakenduse eeliseid ja puuduseid võrdluses seadme- ja raamistikupõhiste rakendustega. Autor kirjeldab hübriidraamistiku, täpsemalt Ionicu tööks vajalike minevate komponentide ehitust ja tööpõhimõtteid. Lisaks tutvustab autor, kuidas toimub töö praktilises osas loodud REST protokolli järgivate teenuste abil infovahetus mobiilirakenduse, administraatori liidese ja serveri vahel.

2. Hübriidrakendus versus seadmepõhine rakendus

Mobiilirakendust luues ei saa eeldada, et kogu kasutajaskond omab ühe ja sama platvormiga seadmeid. 2015. aasta turu-uuringutest ilmneb, et ligi 82% kasutajatest omavad Android seadmeid, 13% iOS seadmeid ning 2% Windows Phone operatsioonisüsteemil jooksvaid nutiseadmeid [18]. Olenemata sellest, et Android seadmete turuosa on kõige suurem, tuleb arvestada, et väitlusmeistrivõistlustest osa võtvatel noortel tudengitel on paljudel ka iOS operatsioonisüsteemil jooksvad nutiseadmed.

Seadmepõhise mobiilirakenduse tegemine erinevat tüüpi seadmetele nõuab teadmisi kahest erinevast programmeerimistehnikast ja keskkonnast. iOS rakenduste arendus toimub Swift programmeerimiskeeles ning Android Javas. Olukorras kus arendajateks on ühed ja samad inimesed, mitte mitu erinevat tiimi, tõusevad arenduskulud juba selle pealt, et aega kulub kauem. Kahte erinevat koodibaasi on hiljem palju keerulisem hooldada.

Hübriidrakendus seevastu võimaldab täpselt sama koodibaasiga funktsioneerimist erinevates seadmetes. Ühe koodibaasiga funktsioneerimise mõistmiseks tuleb aru saada, kuidas hübriidrakendus sisuliselt toimib. Näiliselt on hübriidrakendus sarnane veebirakendusele, mis töötab veebilehitsejas, kuid see esitatakse kasutajale nii, nagu oleks tegu seadmepõhise rakendusega. Mobiilirakenduse arendusele ja valmistamisele kuluv aeg väheneb märgatavalt, sest erinevatel sihtseadmetel ja platvormidel jõutakse täpselt sama tulemuseni ühe koodibaasiga.

2.1 Seadmepõhise rakenduse toimimispõhimõtted

Seadmepõhist rakendust hübriidist eristab see, et rakenduse ehitamiseks kasutatakse sama programmeerimiskeel millel seade töötab. iOS rakenduste puhul Objective-C [19] või Swift [20] ja Androidil Java. Swifti ja Java keelte vajalikul tasemel valdamine võib võtta kauem aega kui HTML-i, CSS-i ja Javascripti puhul, kuid see-eest avaneb ka rohkem võimalusi. Kasutades sama keelt, millel seade töötab, on võimalik otse paljude erinevate funktsionaalsuste poole pöörduda. Näiteks ei ole vaja nutitelefoni kaamera avamiseks mõnda eraldi pistikprogrammi nagu hübriidrakenduste puhul. [21]

Põhiline eelis taoliste rakenduste puhul on oluliselt efektiivsem jõudlus ja töökindlus. Platvormipõhised rakendused jooksevad otse seadme riistvaral ning ei vaja selle jaoks eraldi brauserit.

Suureks erinevuseks on animatsioonide sujuvus ja jõudlus, mistõttu on enamik mängu- või muud suurt jõudlust nõudvad rakendused seadmepõhised.

Seadmepõhised rakendused on kasutajale harjumuspärasemad ja mugavamad. Enamik mõne kindla platvormi kasutajatest on juba harjunud nuppude kuju ja värviga ning menüüde asetusega. Juhul kui hübriidrakendus pole kujundatud täpselt mõnda kindlat platvormi või seadet jäljendama, nõuab see tarbijalt mõningast ümberharjumist.

2.2 Hübriidrakenduse toimimispõhimõtted

Hübriidrakendused sarnanevad väga palju seadme resolutsioonile reageerivatele veebilehtedele. Resolutsioonile reageerivate veebilehtede puhul on tegu näiliselt tavaliste veebilehtedega, kuid need käituvad erinevatelt seadmetelt külastatuna erimoodi. Lauaarvutid, sülearvutid ja nutiseadmed omavad kõik erinevate suurustega ekraane ning veebileht, mis näeb lauaarvutis välja kasutajasõbralik ja kena disainiga, võib nutitelefonist külastamisel oluliselt erineda.

Reageerivate veebilehtede eesmärgiks on tõsta kasutajamugavust eri seadmetel ning tavaliselt saavutatakse see nuppude, menüüde, kirjasuuruse või ükskõik millise sisu ümberpaigutuse või suuruse muutmisega.

Hübriidrakenduse tuumik sarnaneb mitmel viisil tavalise reageeriva veebilehe/veebirakenduse mobiilse kujuga. Mõlema loomisel kasutatakse HTML-i, CSS-i ja Javascripti. Kogu sisu, kujundus ja loogika võivad mõlemal puhul olla samad. Hübriidrakenduse raamistik võtab veebirakenduse sisu ning pakendab selle nii, nagu on tegu seadmepõhise rakendusega.

Peamine erinevus tulenebki sellest, kuidas rakendusele ligi pääsetakse. Veebirakenduse puhul serveeritakse rakendus kasutajale brauseris, kuid hübriidrakendus laetakse alla vastava platvormi rakenduste poest ning avatakse täpselt samamoodi, nagu oleks tegu seadmepõhise rakendusega.

3. Ionic raamistik ja selle komponendid

Ionic raamistiku kasutamisele käesolevas projektis on mitu erinevat põhjust. Algselt oli planeeritud rakendus ehitada eraldi iOS ja Android platvormidele. Ajaliste ressursside kokkuhoiu mõttes võttis autor vastu otsuse kasutada mõnda hübriidraamistikku. Peale Ionicu oli veel valikus React Native ning suhteliselt värske hübriidraamistik nimega Famous [22], [23]. Ionic sai valituks põhjusel, et autor polnud varem React Native-ga kokku puutunud ning arvas, et Famous-il puudub veel piisav tugi ja dokumentatsioon.

Suurt rolli hübriidraamistiku valituks osutumisel mängis ka see, et arendatav rakendus ei tule eriti animatsioonirikas ja kohmakas. Eesmärgiks on võimalikult lihtne ja kiire vajaliku informatsiooni kuvamine kasutajale.

Ionicu beetaversiooni väljalase oli 2013. aasta viimases kvartalis, lõplik 1.0 versioon alles 2015. aasta mais ja 2.0 versioon sai saadavaks 2017. aasta algul. Tegemist on tehnoloogiaga, mis on jätkuvalt pidevas arenduses. Ionicu kasutajaskond on igal aastal tohutul kiirusel kasvanud. Võrdluses 2014. aastaga, lõid Ionicut kasutavad arendajad 2015. aastal üle 300% rohkem rakendusi. Viimaste koguhulk küündis tol aastal üle 1.3 miljoni rakenduse. [24]

Tänu üha suurenevale arendajate hulgale ja nutitelefonide jõudluse tõusule, on Ionic raamistiku võimalused pidevalt tunduvalt paranenud. Paremate nutitelefonide ja optimeerimise läbi on animatsioonid ning andmete pärimine muutunud sujuvamaks ja kiiremaks. Ionicul on korralik ja terviklik dokumentatsioon, kuid kui ei olda varem taolise raamistiku või mobiilse arendusega kokku puututud, võib õppimiskõver olla järsk. Ionicu kasvav ja aktiivne kogukond leiab tekkinud probleemidele sageli kiire lahenduse. Arendajate rohkuse tõttu on tõenäoline, et keegi on juba sarnase probleemiga silmitsi seisnud ning algajatel arendajatel on selle võrra kergem enda esimese rakendusega algust teha.

3.1 Apache Cordova

Apache Cordova on vabavaraline mobiilse tarkvara arendamise raamistik ja on ka üks Ionicu alustaladest. Algselt tunti Cordovat PhoneGap nime all, mis sai alguse juba aastal 2009. Varasemad versioonid ei olnud sugugi mitte nii paindlikud kui praegune. Varem oli iOS rakenduse loomiseks vaja Apple arvutit ning Windows rakenduste jaoks Windows arvutit. Alles 2012. aasta septembrist, kui Cordova raamistiku arendusega tegeles juba Adobe, sai võimalikuks rakenduste loomine korraga erinevatele platvormidele.

Nüüd võimaldab Cordova standardsete veebitehnoloogiatega luua mitmikplatvorm rakendusi, mis pakendatakse vastavalt erinevatele platvormidele nii, nagu oleks tegu seadmepõhise rakendusega. Tänu erinevatele pistikprogrammidele on need rakendused võimelised ka suhtlema erinevate seadme funktsioonidega, nagu näiteks kaamera, GPS, kontaktide nimistu jne. Cordova haldab suurt listi tavalistest tuumikpistikprogrammidest, kuid võimalik on külge haakida ka kolmanda osapoole laiendusi. [25]

3.2 AngularJS

Angular on Javascripti teek, mida kasutatakse dünaamiliste veebirakenduste loomiseks. See laiendab HTML-i süntaksit, et ehitatava rakenduse komponente selgemalt ja lühidamalt väljendada. Eelmainitu saavutatakse näiteks rakenduste jaoks kohandatud direktiividega ja hõlpsalt taaskasutatavate komponentide abil. Angulari loojate põhiline filosoofia ütleb seda, et Angular on HTML sellisel kujul, kui HTML olekski algselt mõeldud veebirakenduste ehitamiseks. [3]

Angular toetab kahepoolset andmesidumist. Kahepoolne andmesidumine tähendab seda, et muutused kasutajaliideses kajastuvad kohe ka mudelis ning vastupidi. Kahepoolne andmesidumine vähendab oluliselt serveri koormat esitada uusi dokumente. $scope objekt tuvastab muutusi mudelis ning muudab kontrolleri abil kasutajaliidest. See elimineerib vajaduse pidevalt DOM-e, ehk HTML-i objekte [10] manipuleerida ning soodustab rakenduse kiiret reaalajas prototüüpimist. Mudeli kaudu on võimalik väärtustada kasutajaliideses esinevaid elemente, mille puhul on tegu ühepoolse andmesidumisega. Kasutajaliideses sisalduva tekstivälja või mõne muu vormiga on võimalik selle kaudu väärtustada või ümberväärtustada ka mudelis esinevaid muutujaid. [26]

Angular mängib suurt rolli ka Ionicul põhinevate rakenduste jõudlusel. Cordova ja Ionicu Angulari koodi segunemisel suudab Ionic kasutada seadme riistvarakiirendust, mitte ei tule seda teha ekstensiivse DOM-i manipuleerimisega, nagu seda teeb näiteks jQuery. [1], [27]

3.3 HTML, CSS ja Sass

HTML, CSS ja Sass on peamised veebitehnoloogiad, millega Ionicu kasutajaliidest viimistletakse - nii nagu ka brauseripõhise veebirakenduse loomisel. Arendamine Ionicus on soodustatud, kuna paljud arendajaid on nende tehnoloogiatega varem kokku puutunud.

HTML-i abil luuakse lehe ehk mobiilse vaate struktuur ning CSS-i, Sassi ja veidi ka Javascripti abil välimus ning animatsioonid. CSS-i puhul on tegu standardse HTML-i failide kujunduskeelega ning Sass on üks levinumaid CSS-i laiendusi. Sass on CSS-i eelprotsessor, mis tähendab seda, et arendaja kirjutab koodi Sass keeles ning hiljem kompileeritakse see CSS-i. Tänu muutujate olemasolule, funktsioonidele, üksteise sees sisalduvatele reeglitele – omadused mis tavalisel CSS-il puuduvad, võimaldab Sass arendajatel paindlikumalt ja organiseeritumalt koodi kirjutada. [2]

Ionic sisaldab juba vaikimisi suurt hulka erinevaid komponente ja blokke, mille abil liidese skelett või isegi juba algeline töötav rakendus valmis ehitada. Soovitud disaini ja kujunduse saamiseks tuleb luua enda CSS fail või muuta ja üle kirjutada olemasolevate komponentide reegleid.

Üheks mooduseks rakenduse jõudlust suurendada, võimendab Ionic CSS-i animatsioonide üleminekuid ja muutusi, sest peamiseks animatsioonide tekitajaks on CSS. Võimendamisega on võimalik saada võimalikult palju tuge seadme graafikaprotsessorilt ja suurendatakse protsessori saadavaloleku aega. Kõige selle lõpptulemusena näivad animatsioonid kasutajale sujuvamad ja kiiremad. [1]

4. Andmete hoiustamine serveris ja RESTful API

Mobiilirakenduse jaoks vajalikke andmeid võib abstraktsemal tasemel kohelda kui eraldi olemeid või entiteete. Nende hoiustamine toimub serveris asuvas andmebaasis ning nende pärimiseks ja manipuleerimiseks on vaja projekti enda API-t.

Olemid nagu näiteks Teade ja Kasutaja, on kirjeldatud mudelitena tavaliste Java klasside kujul, milles on kirjeldatud neile vastavad muutujad ja meetodid. Hibernate ORM(Object Relational Mapping) [28] raamistiku abil kaardistatakse objekt-orienteeritud mudelid relatsioonilise andmebaasi tasemele. Antud projekti puhul on relatsioonilise andmebaasi puhul tegu MySQL-i haruga nimega MariaDB. Koos andmemudeli objektiga kaardistatakse ka objektiga seotud Java andmetüübid SQL andmetüüpideks. [29], [30]

4.1 Projekti API RESTful veebiteenustena

Projekti API järgib RESTful (Representational State Transfer) arhitektuuri [4]. Andmeobjekte käsitletakse ressurssidena ning on URI-de [31] kaudu veebist ligipääsetavad. RESTful arhitektuur sobib antud projekti jaoks hästi, sest andmeobjektid on oma URI-l rakendusele lihtsal kujul kättesaadavad ja serveripoolseid muutuseid on võimalik läbi viia ilma, et see rakenduse tööd kuidagi mõjutaks. Veebist kättesaadavuse saavutamiseks peab eelnevas peatükis mainitud objekti mudeliga olema seotud ressursi-, teenuse- ning DAO Java klassid. [12]

Teenuseklassi sisestatakse sõltuvus DAO klassist ja luuakse sellest uus objekt. Seejärel tehakse meetodid Teadete lisamise ja kogu Teadete hulga pärimise jaoks ning seotakse need DAO liidesega. Teenuseklassi programmikood on leitav Lisa 5.

Ressursiklass peab sisaldama määratud baas-URI annotatsiooni, mille kaudu on võimalik andmetele läbi veebi ligi pääseda. Ressursiklassi sisestatakse sõltuvus teenuseklassist ja luuakse sellest uus objekt, tänu millele on nüüd võimalik teenuseklassis defineeritud meetodeid rakendada ning määratud baas-URI-l Jackson protsessori abil JSON kujul kuvada või vastu võtta [11], [32]. Ressursiklassi kood on leitav Lisa 6.

DAO klassi puhul on tegu liidesega, mis muudab operatsioonide teostamise andmebaasiga oluliselt abstraktsemaks. Java meetodite ümberkaardistamise kaudu on võimalik teostada andmeoperatsioone ilma, et oleks teada täpsemad andmebaasi detailid. DAO puhul on tegu

justkui vahelüliga andmebaasikihi ja rakenduskihi vahel. See vähendab suuresti rakenduse kahe osa omavahelist sõltuvust. Väikese muutuse korral andmebaasiloogikas või vastupidi, äriloogikas, piisab DAO implementatsiooni modifitseerimisest, ilma, et oleks vaja tervet rakendust või andmebaasi loogikat muuta. [12] Ka siin leiab kasutust Hibernate ORM raamistik. DAO klassi kood on leitav Lisa 7.

RESTful veebiteenuse implementeerimise hõlbustamiseks Javaga on autor projektis kasutusele võtnud Jersey raamistiku. [33] Jersey toetab Java JAX-RS [34] annotatsioonide lugemist ja implementatsiooni. Tänu sellele on võimalik meetodit ümbritseva annotatsiooniga teha kompileerijale selgeks, et antud meetod vastab HTTP GET [35] tüüpi päringutele.

4.2 Jetty veebiserver

Vajalikele ressurssidele – Teadetele, Asukohtadele, HTTP päringute kaudu ligi saamiseks tehakse projekti API Java serveriprogrammi abiga kättesaadavaks. Antud projekti jaoks sobib hästi Jetty veebiserver, sest et projekti API on kirjutatud Javas. Jetty veebiserver toetab alati hiljutist Java serveriprogrammi API-t [36]. Serveriprogrammi konfiguratsioonifailis on määratud, milliselt pordilt on võimalik API-le ligi pääseda. Ära tuleb ka märkida, millise aadressi kaudu saab serveriprogramm andmebaasile ligi pääseda.

5. Administraatori liides

Administraatori liides mängib väitlusvõistluste toimumise ajal rakenduse juhtimise osas suurimat rolli. Liidese kaudu on võimalik kontrollida pea kõike rakenduses toimuvat. Liides on vastavate õigustega isikule veebist ligipääsetav.

Liidese loomiseks kasutas autor standardseid veebitehnoloogiad nagu HTML, Sass, jQuery ning Bootstrap [37]. Arenduskeskkonnas liidese ja API vahelise suhtluse testimiseks peab ka liides mingil serveril töötama, sest AJAX päringute jaoks on tarvis, et suhtlus toimuks samal võrgupordil. Veebiserverina arendusseadmel kasutab autor Apache veebiserverit. [38], [8]

5.1 Päringute tegemine samal pordil töötavale API-le

Ühel ja samal võrgupordil nii Apache virtuaalserveri ja Jetty serveri käivitamisel tekib konflikt, sest see võrguport on juba esimesena käivitatud rakenduse poolt hõivatud. Konflikti lahendas autor kahe serveri käivitamisel erineval võrgupordil ning kasutades mod_proxy mooduli liidest võõrustavas Apache serveri konfiguratsioonifailis httpd.conf. Väljavõtet failist httpd.conf on näha Joonis **1**.

Administraatori liidesele ligi pääsemiseks tuleb Apache serveri konfiguratsioonifailis esmalt luua virtuaalne server ning kirjeldada selle aadress, võrguport ning lähtefailide asukoht. Seejärel on mod_proxy mooduli abil võimalik tegevusi virtuaalserveris vastavate URL-i tingimustel proxy kaudu edastada ja ümber pöörata. On vajalik, et liideselt tehtavad päringud API-le suunatakse ümber teisele võrgupordile, et jääks mulje, nagu päringud tulekski samalt pordilt millel töötab API server. Kindlale URL-i teele päringute tegemiseks saab konfiguratsioonifailis kehtestada reeglid, mis suunavad kõik administraatori liidesest sooritatud seda URL-i teed sisaldavad URL-i külastused hoopis teist serverinime omavat URL-i teele. Seeläbi jääb Jetty serverile mulje, et päringud administraatori liidesest tulevad samalt võrgupordilt.

```
<VirtualHost *:7071>
ServerName localhost
DocumentRoot C:/Users/Karl/Desktop/10100111001/EUDC/EUDC-2017-Admin-Panel
<Directory />
Require all granted
Options Indexes FollowSymLinks Includes ExecCGI
AllowOverride All
Order deny,allow
Allow from all
</Directory>
ProxyRequests Off
ProxyPreserveHost On
ProxyPass /rest http://127.0.0.1:7070/rest
ProxyPassReverse /rest http://127.0.0.1:7070/rest
ErrorLog C:/Apache24/logs/error.log
CustomLog C:/Apache24/logs/cess.log combined
</VirtualHost>
```

Joonis 1. Apache virtuaalserveri konfiguratsioon

5.2 Uudistevoo teadete haldamine

Eeldusel, et projekti API ja ka administraatori liides töötavad samaaegselt ja nendevaheline suhtlus toimib, on läbi liidese võimalik luua turniiriteateid.

Teade koosneb pealkirjast, kirjeldusest, loomise kuupäevast ja teate illustreerimiseks väikesest ikoonist. Administraatoril on võimalik liidese kaudu täita vorm, mille kaudu saab sisestada nii teate pealkirja kui ka kirjeldust. Pealkiri ja kirjeldus tuleb sisestada tekstiväljadele. Ikoon on erinevatel teate tüüpidel vaikimisi määratud ning loomise kuupäeva sätestab andmebaas automaatselt.

Vajutades seejärel nuppu ’Create’, kutsutakse jQuery-ga välja AJAX päring. POST-tüüpi [35] päringu eesmärgiks on saata JSON-kujul objekt koos vajalike parameetritega teadete infot hoidvasse serverisse. Päringu õnnestumise ja parameetrite valideerimise õnnestumise korral luuakse andmebaasi uued kirjed vastloodud teate kohta. Seejärel käsitsi või automaatse

mobiilirakenduse uudisvoo värskendamise korral on võimalik rakenduses uut teadet juba näha. Uuendatud turniirivoog on näha ka administraatori liideses.

```
$.ajax({
      type: "POST",
      url: "rest/card",
      data: JSON.stringify({
        "title": $("#card-title").val(),
        "description": $("#card-description").val()
      }),
      contentType: "application/json; charset=utf-8",
      dataType: "json",
      success: function(data) {
        $('.create-card-notification').text("Card creation successful!");
        $('#add-card').each(function() {
          this.reset();
        });
        return false;
      }
    })
```

Joonis 2. AJAX päring uue teate loomiseks

Uudistevoos sisalduvad teated kuvatakse administraatorile samuti AJAX päringu ja jQuery abil. Pöördudes administraatori liideses uudistevoo lehele, lähetatakse lehe laadimisel teele ka AJAX päring, mis pärib API-lt andmeid teadete kohta. Päringu õnnestumise korral luuakse jQuery-ga iga teadete kollektsiooni elemendi kohta uudistevoogu ühe teate kuvand.

```
$.ajax({
      url: 'http://188.166.104.203:7070/rest/card',
      dataType: 'json',
      success: function(data) {
          var formattedDate, d, m, y, h, min, date;
        $.each(data, function(key, value) {
            formattedDate = new Date(value.created);
            d = formattedDate.getDate();
            m = formattedDate.getMonth();
            y = formattedDate.getFullYear();
            h = formattedDate.getHours();
            min = formattedDate.getMinutes();
            date = d + "." + m + "." + y + " " + h + ":" + min;
            $('.cards').append('<div class="col-xs-12 col-md-6 card"><div
class="col-xs-12 card-header"><h3>' + value.title + '</h3></div><div
class="col-xs-12 card-date">' + date + '</div><div class="col-xs-12 card-
text">' + value.description + '</div></div>');
        });
      }
    });
```

Joonis 3. AJAX päring turniiriteadete kuvamiseks

5.3 Väitlusvooru algust meeldetuletavad teated

Administraatoril on võimalik luua uudistevoogu teateid, milles sisaldub meeldetuletus kasutajale enne algava vooru algust. Teade kuvatakse võistlejale koos vooru teema, võistleja poole(poolt või vastu), vooru alguseni loendava taimeri ning õppehoone ja ruumiga, kus kohal peab olema.

Teadete loomise vorm ja uudistevoog administraatori liideses on leitav Lisa 2.

6. Ionic projekti struktuur ja mobiilivaated

6.1 Andmevahetus läbi erinevate kihtide

Ionic rakenduse sisemuses eristub selgelt kolm erinevat kihti: andmekiht, vaatekiht ja kontroller [39]. See tuleneb AngularJS-i ülesehitusprintsiipidest ja on sisuliselt väga sarnane tüüpilisele MVC(Model-view-controller) arhitektuurimustrile. Ionicu ülesehituse mõistmine on oluliselt lihtsam, kui ollakse varem MVC mustriga kokku puututud. [40]

Andmekihist pärinevad rakenduses serveeritavad andmed. Andmetele pääsetakse ligi päringuga välisest serverist või veebiteenuse kaudu. Antud töö raames kasutab autor veebiteenuseid ja projekti API-t. Päringu õnnestumise korral on kontrolleril võimalik vastuses olevate andmetega manipuleerida ja neid kasutada.

Vaatekiht koosneb erinevatest mallidest. Iga vaate puhul on tegu eraldi HTML failidega, mis esitavad kasutajale füüsilisel kujul reaalse oleku või lehe, milles sisalduvad ka andmed kontrolleri objektist $scope. Eraldi vaadeteks on näiteks sisselogimise-, uudiste voo-, menüü-, kaardi- ja teate detailvaade.

Vaadetes on võimalik kasutada Angulari direktiive [41]. Direktiivide puhul on sarnaselt CSS-i klassidega tegemist kindlate märgistega HTML-i elementide küljes, mis ütlevad Angulari kompileerijale, kuidas vastavate elementidega kompileerimise käigus käituda. Direktiividega on võimalik HTML-i elemente või nende alamelemente paindlikumalt manipuleerida.

Vaatekihi ja andmekihi vahelist tegevust juhib kontroller, mis juhib andmete liikumist läbi andmete sidumise(data-binding) andmete kihist vaadete kihti ning vastupidi. Pärast kasutaja suundumist rakendusesiseselt mõnele lehele ehk URL-ile, kutsutaksegi kõigepealt välja kontroller ja vaate mall. Kontroller kasutab vaate esitamiseks malli, ning seob selle andmetega, mis tulevad andmekihist projekti API veebiteenuse kaudu. Andmed salvestatakse $scope objektis, mida kontroller kasutab vaate esitamiseks.

6.2 Rakenduse konfiguratsioon

Vaadete ja kontrollerite laadimise erinevates rakenduse olekutes eest hoolitseb rakenduse konfiguratsioon. Üheks rakenduse olekuks on teatud URL-il viibimine. Konfiguratsioonis kirjeldatud olekute läbi teab rakendus täpselt, millist kontrolleri ja vaate kombinatsiooni

kasutajale esitada. Konfiguratsioonisätetes on võimalik kehtestada vaikimisi olekud või seisundid. Vaikimisi olekud tulevad kasuks sisselogimise ja kasutaja autentimise lahendamisel. Saab defineerida reegli, et autentimisprotsessi läbimata ei ole võimalik muid rakenduse olekuid saavutada. Üritades pärast rakenduse avamist ilma sisse logimata minna uudistevoo URL-ile, suunatakse kasutaja automaatselt URL-ile, kus ta peab end kõigepealt sisse logides rakenduse jaoks tuvastama.

7. Turniiriteadete pärimine ja kuvamine kasutajale

Rakenduse põhilise osa, ehk turniiriteadete edastamine ja kuvamine, toimib eelmises peatükis kirjeldatud kihtidevahelise suhtluse abil. Teadete kuvamine kasutajale toimub vaates nimega tournamentFeed.html, andmed päritakse teenuse serverCallService.js abil ning andmete sidumisega kasutajaliidese ja teenuse vahel tegeleb kontroller tournamentFeedCtrl.js.

Teenuse serverCallService abil on võimalik pöörduda serveris paiknevate andmete poole. Turniiriteadetega seotud objektid, ehk kaartide hulk JSON kollektsioonid asuvad API ressursiklassi poolt saadavaks tehtud URL-il. Teenuses serverCallService on defineeritud funktsioon makeGet, mis on sisuliselt tavaline HTTP GET väljakutse, mida saab turniiriteadete kontrolleris tournamentFeedCtrl.js kasutada.

Funktsioon makeGet saab parameetriks URL-i, millel päritavad andmed serveris paiknevad, andmeid hoidva muutuja ning päringu õnnestumise ja ebaõnnestumist käsitlevad funktsioonid. Ebaõnnestumise korral kuvatakse kasutajale veateade, õnnestumise korral käivitub funktsioon, kus käideldakse päringu vastusega kaasnenud infot.

```
makeGet: function(url, params, successCallback, errorCallback,
finallyCallback) {
makeCall(url, 'GET', params, true, successCallback, errorCallback,
finallyCallback);
}
```

Joonis 4. Funktsioon makeGet teenuses serverCallService

Õnnestumise korral kutsutakse välja funktsioon nimega ’success’, mis väärtustab objekti $scope muutuja nimega cards väärtuseks päringu vastuses tulnud andmed. Ehk $scope.cards väärtuseks on JSON kujul list kaardiobjektidest koos nende parameetrite ja väärtustega.

```
function success(data) {
        $scope.cards = data;
}
```

Joonis 5. Funktsioon success uudistevoo kontrolleris

$scope objektile väärtustatud muutujad on võimalik kasutajaliideses kuvada, kui nad on HTML-is kirjeldatud kahekordsete loogeliste sulgude vahel. Pärast seda kui serveripoolse päringu vastuseks on üks element nimega ’title’ ja selle väärtustamisel kontrolleris $scope.title

muutujana, tekib mallis elemendi {{title}} asemele pärast vaate formuleerimist serverilt päritud väärtus.

Juhul kui andmepäringu vastuseks on objekte rohkem kui üks, ei tule tänu Angularile iga objekti jaoks HTML-is eraldi uut plokki kirjutada, vaid on võimalik üle kogu kollektsiooni korraga itereerida ning malli täpselt nii palju kordi välja kutsuda, kui on kollektsioonis objekte. Selle ülesande jaoks on Angularis olemas direktiiv ngRepeat, mis toimib sarnaselt klassikalisele for-each tsüklile [42]. Kollektsiooni $scope.cards elemente on võimalik mallis ükshaaval nime all 'card' välja kutsuda direktiivi ng-repeat="card in cards" abil. Teatest tekib kirjeid täpselt nii palju, kui neid päringuga vastuseks tuli ning lisaks saab hõlpsalt vastava teateobjektiga seotud nimi/väärtus paaridele ligi pääseda. Teateobjektiga on seotud teate nimi, pealkiri, loomise kuupäev, kasutajad kellele teadet näidatakse ning kaks tõeväärtustüüpi andmevälja.

```
<ion-item ng-repeat="card in cards | orderBy: ['-pinned', '-created']"
class="item feed-card">
    <a ng-click="goToCardDetail(card)" href="">
        <div class="item-avatar-left">
            <i class="ion-icon ion-ios-paper-outline"></i>
            <h2>
                <i class="ion-pin pinner" ng-show="{{card.pinned}}"></i>
                {{card.title}}
            </h2>
            <p>{{card.description | limitTo:90}}...</p>
            <div>
                <p>{{ card.created | date:'dd MMMM, HH:mm' }}</p>
            </div>
            <ion-option-button class="button-assertive" ng-
click="destroyCard(card)">
                Delete
            </ion-option-button>
        </div>
    </a>
</ion-item>
```

Joonis 6. Turniiriteadete kuvamine uudistevoo mallis

7.1 Taimeriga teadete kuvamine

Lisaks tavalistele teadete pärimise funktsioonile on kontrolleris tournamentFeedCtrl.js eraldi funktsioon taimerteadete pärimise jaoks. Taimerteate objekti pärimise protsess API kaudu on täpselt sama nagu tavalise teate korral, kuid selle kuvamine uudiste voos märksa keerulisem.

Peale staatilisele teate pealkirjale, kirjeldusele ja kuupäevale, on teates tarvis kuvada dünaamiline allaloenduv taimer ning aja vähenedes anda teate kujunduse muutmisega kasutajale lähenevast voorust märku.

Allaloenduva taimeri kuvamiseks teates võttis autor kasutusele kolmanda osapoole Angulari direktiivi nimega Angular Timer. [43] Angular Timer tuleb taimerteate toimiseks seadistada end-time režiimile. Direktiivil on sisendiks vaja taimeri lõppkuupäeva ja kellaaega. Direktiivi kuvamisel hakkab see loendama päevi, tunde, minuteid, sekundeid kuni määratud lõppkuupäeva kättejõudmiseni. Lõppkuupäeva saab direktiiv kätte muutujast endDate, mis väärtustatakse uudiste voo kontrolleris objekti $scope muutujaks pärast õnnestunud lõppkuupäeva päringut läbi projekti API ja andmebaasi.

```
<timer end-time="endDate"><br>{{hours}} hours, {{minutes}} minutes,
{{seconds}} seconds.</timer>
```

Joonis 7. Taimeri kuvamine taimeriga teates

Pärast taimeri jõudmist määratud minutite kaugusele lõppkuupäevast ja kellaajast, muutub taimerteate kujundus väljatorkavamaks, et anda kasutajale uuesti märku lähenevast voorust. Selle saavutamiseks kontrollib taimerteate pärimise funktsioon alati vahe praeguse ajahetke ja lõppkuupäeva vahel. Hetkest mil see vahe on soovitud ajaühikust 15 minutit väiksem, antakse turniirivaate mallis taimerteadet kuvava elemendi CSS klassiks uus väärtus nimega “time-alert”. Seejärel muutub taimerteate elemendi kujundus vastavalt CSS-failis defineeritud reeglitele silmatorkavamaks, antud juhul tumedamaks. Uudistevoog ja ühe teate detailvaade on leitav Lisa 1.

```
function timercardSuccess(data) {
        $scope.timercards = data;
        // takes the first element from array(should be only one timercard
at once for one person)
        $scope.endDate = data[0].endDate;

        var currentDate = new Date();
        var endDate = new Date(data[0].endDate);
        //checks if there is less than 15 minutes between current time and
timercard end time
        if (endDate - currentDate <= 900000) {
            $scope.time = "time-alert";
        }
    }
```

Joonis 8. Väljavõte taimerteadete pärimise funktsioonist

```
.time-alert {
    background-color: rgba(0, 0, 0, 0.5) !important;
}
```

Joonis 9. Väljatorkavama taimerteate CSS reegel

8. Tallinna linnas ning TTÜ ülikoolilinnakus orienteerumist hõlbustav kaardivaade

8.1 Tallinna kaart tähtsamate huvipunktidega

Mobiilirakendus sisaldab Tallinna linnas orienteerumist hõlbustavat kaardivaadet, kuna rakenduse sihtgrupi moodustab valdavas osas välistudengid Euroopast. Eraldi brauserit või mõnda kolmanda osapoole rakendust avamata, on kasutajal võimalik väitlusrakenduse menüüst talle enim olulisi asukohti kuvavale kaardile kiirelt ligi pääseda.

Tallinna kaardivaate vundamendi moodustab Google kaardirakenduse avalik programmiliides [44]. Standardne tasuta kasutusmaht on kuni 25 000 kaardi laadimist 24 tunni jooksul. Antud projekti jaoks on see täiesti piisav.

Pärast kasutaja sisenemist Tallinna kaardi vaatesse, kuvatakse kaardileht Tallinna Tehnikaülikooli lähiümbruskonnast ning kasutaja enda täpne asukoht on tähistatud rohelise markeriga. Kasutaja täpne asukoht on võimalik kätte saada tänu Cordova Geolocation pistikprogrammile [45]. Cordova Geolocationi abil on võimalik seadmelt pärida tema asukoha pikkus- ja laiuskraadi läbi GPS-i, traadita internetivõrkude ja ka GSM ehk globaalse mobiilsidevõrgu identifikaatorite järgi. Õnnestunud positsioneerimise korral edastatakse asukoha andmed Google Maps API-le, mis kasutab neid asukoha kuvamiseks. [44]

Tähtsamad huviobjektid, võistlusi ja transporti puudutavad asukohad on märgitud punaste markeritega. Nendele peale vajutades ilmub nähtavale infoaken, milles sisaldub täpsem info, mis objektiga on tegu. Kõikide markerite kuvamiseks itereeritakse RESTful serveris vastava URL-i all olevad elemendid läbi ning saadud koordinaatide, infoakna sisu põhjal luuakse iga vastava elemendi kohta uus markeriobjekt.

8.2 Tallinna Tehnikaülikooli linnaku kaart

Ülikoolilinnaku kaardivaates kuvatakse kasutajale terve linnaku kaart koos erinevate majade, asukohtade ja nende tähistustega. Linnaku kaart ei ole interaktiivne, vaid kuvatakse lihtsalt suure resolutsiooniga pilt, mis on võistlejale kiireks esmaseks teejuhiks.

Taimerteatesse ilmub võistlejatele võimalus vaadata toimumiskoha väljavõtet, mille tegi autor käsitsi ülikoolilinnaku kaardi alusel. Vajutades toimumiskohale taimerteatel, suunatakse kasutaja vaatesse, kus kuvatakse väljavõte ülikoolilinnaku kaardist. Väljavõttel on silmatorkavalt ära märgitud vastav õppehoone, kus võistleja peab kohal olema. Tallinna ja Tallinna Tehnikaülikooli linnaku kaardivaated rakenduses on leitav Lisa 3.

8.3 Kaardivaate haldamine administraatori liideses

Administraatori liideses on võimalik kaardile märgitud markereid nii lisada kui ka eemaldada. Uue markeri lisamiseks on vaja vajutada soovitud kohale kaardiaknas, täita asukoha nime tähistav lahter ning vajutada ’Add’ nuppu. Pärast nupuvajutust sooritatakse AJAX päring, mis saadab markeri pikkus- ja laiuskraadi ning nime andmed serverisse. Päringu õnnestumise korral lisandub uus marker kaardiakna kõrval asuvasse markerite nimistusse ning kirje vastavast markerist ilmub nähtavale ka mobiilirakenduse vaates.

Nimistus olevaid markereid on administraatoril nupuvajutusega võimalik kaardil kuvada ning vajadusel kustutada. Kustutamisoperatsioon viiakse läbi jällegi AJAX päringuga, mis saadab serverile info millise identifitseerimisnumbriga marker eemaldada.

Nimistus eksisteeriva objekti markeri kuvamise soovi korral kustutatakse kaardilt olemasolu korral seni nähtaval olnud marker ning markeri koordinaate arvesse võttes liigutatakse kaarti uude piirkonda. Objekti täpsele kohale paigutatakse seejärel taaskord punane marker.

Sarnase loogika alusel on administraatoril võimalik opereerida ning omada ülevaadet ka voorude toimumiskohtadest. Pärast uue toimumiskoha kirjelduse ning pildi asukoha sisestamist serverisse on neid seeläbi võimalik hakata siduma taimerteadetega. Mõne toimumiskoha muutumisel või juurde tekkimisel tagab see funktsionaalsus administraatori liideses paindlikkuse seda kiirelt muuta. Tallinna ja Tallinna Tehnikaülikooli linnaku haldamise vaated administraatori liideses on leitav Lisa 4.

9. Rakenduse testimisvõimalused Ionic raamistikus

Rakendust on vaja pidevalt kontrollida ja testida, kuidas senine arendus reaalsel seadmel välja näeb. Mobiilivaadet on võimalik simuleerida laua- või sülearvutis ka brauserite ja veidi täpsemini spetsiifiliste Androidi või iOS simulaatorite abil, kuid tõelist riistvara see ei asenda.

9.1 Rakenduse testimine lauaarvutis

Rakenduse testimine kohalikus võrgus ja masinal on Ionicus võimalik ühe käsuga Ionicu CLI(Command Line Interface) kaudu [46]. Käsu *ionic serve* abil on võimalik rakendus nii arendus- kui ka testimiseesmärkidel lauaarvuti brauseris või samasse võrku ühendatud seadme brauseris käivitada. Käsu sisestamisel läheb kohalikus masinas käima NodeJS server ning kohalikult veebiaadressilt on võimalik läbi brauseri rakendusele ligi saada. [47]

Üheks brauseris testimise puuduseks võib lugeda seadmepõhiste funktsionaalsuste puudumise. Brauserid ei võimalda jäljendada mõningaid kindla seadme, eriti selle riistvaraga seotud omadusi. Heaks näiteks on seadme GPS funktsionaalsuse sisselülitamine. GPS funktsionaalsuse testimiseks on vaja rakendust emuleerida või testida vastavaid võimalusi omavas seadmes.

Kohalikus võrgus ja masinal töötaval rakendusel on võimaldatud funktsionaalsus nimega LiveReload, mille abil on võimalik koodis tehtud muutusi kohe reaalajas töötava rakenduse kujul näha. LiveReload jälgib muutusi vastavas kataloogis kuhu kuuluvad tavaliselt HTML, CSS ja Javascript failid. Pärast igat uut failisalvestust värskendatakse brauserit, kus rakendus töötab. Automaatne brauseri värskendamine tõstab oluliselt rakenduse arendamise kiirust ja mugavust, kui tehakse tihti erinevaid muudatusi. Antud omadus kujunes autorile kasulikuks disainimise ja kujundamise etapil. Tänu LiveReloadile ei olnud autoril erinevaid värvilahendusi proovides kogu aeg käsitsi brauseriakent tarvis värskendada. [48]

9.2 Rakenduse seadmepõhine prototüüpimine

Ionicu CLI-le on sisse ehitatud ka seadmepõhise töötava rakenduse ehitamine. Käsuga *ionic run android* käivitub protsess, mis kogub kokku rakenduse hetkeseisu ning ehitab sellest töötava rakenduse USB-ga arvuti külge ühendatud seadmele. Sama protsess on võimalik ka iOS seadmetel, kuid selle jaoks peab olema tasuliselt registreeritud Apple arendajana. Rakenduse

testimine toimus Android operatsioonisüsteemiga nutitelefonidel Huawei Y550 ja Samsung Galaxy S4.

9.3 Pideva integratsiooniga arendus

Sülearvuti brauseris on võimalik nii mobiilirakendust, administraatori liidese, andmebaasi ning projekti API-t kõike korraga tööle panna, kuid konkreetselt nutitelefonil testimiseks peavad projekti serveripoolsed komponendid olema välises serveris üleval, et neile nutitelefonist üle võrgu ligi saaks pääseda.

Autor võttis kasutusele pideva integratsiooni tarkvara nimega Jenkins. [49] Jenkinsi eesmärk on katkematult jälgida projekti kolme haru: mobiilirakendust, administraatori liidest ning projekti API repositooriume Git [50] keskkonnas ja iga uue muudatuse korral luua sellest ka välisesse serverisse uus versioon. Eeldusel, et serveris on üleval töötav versioon projekti API-st ja ka andmebaas, on võimalik mobiilirakendusel API kaudu andmebaasist võimalik samamoodi infot pärida nagu oleks tegu kohaliku võrgu ja seadmega.

10. Kokkuvõte

Töö põhieesmärgiks oli luua väitlusvõistluste tarbeks Ionic raamistikul põhinev mobiilirakendus ning seda administreeriv veebiliides.

Väitlusrakenduse toimimiseks arendatud komponendid jagunevad kolmeks osaks: mobiilirakendus, administraatori liides ja projekti API. Töö tulemusena valmisid rakenduse tähtsaimad funktsionaalsused – teadete kuvamine uudiste voos ning orienteerumist hõlbustav kaardivaade. Autor tagas suhtluse rakenduse komponentide vahel ning leidis lahenduse turniiriteadete ning orienteerumist hõlbustava info kuvamiseks mobiilirakenduses.

Ühe järeldusena ilmnes autorile, et hübriidse mobiilirakenduse arendamine sarnaneb veebibrauseris töötava veebirakenduse loomisega. Kasutatakse samu programmeerimistehnikaid ning tehnoloogiaid. Lisaks sellele järeldab autor, et kuigi rakendust saab kiirelt ja reaalajas brauseris testida, on kõikide funktsionaalsuste põhjalikumaks testimiseks oluline, et hübriidrakendust testitakse ka seadmepõhiselt.

Selle bakalaureusetöö raames seisnes rakenduse arendus ühe kindla väitlusvõistluse jaoks. Seda edasi arendades oleks võimalik rakendust muuta või kasutada mallina ka muudel tulevastel väitlemisega seotud üritustel ja võistlustel.

Põhitulemused:

- Valmis võistluste ajal rakendust haldava isiku tööd hõlbustav administraatori liides.
    - Administraatori liidese kaudu on võimalik näha uudistevoos olevaid teateid.
    - Administraatori liidese kaudu on võimalik luua uus turniiriteade.
    - Administraatori liidese kaudu on võimalik hallata üritusega seotud kohti Tallinnas ja voorude toimumiskohti Tallinna Tehnikaülikooli linnakus.
- Valmis turniiriteateid ja tudengite orienteerumist hõlbustav mobiilirakendus.
    - Mobiilirakenduses on võimalik kasutajal näha uudiste voos nii tavalisi turniiriteateid kui ka algava vooru meeldetuletust allaloenduva taimerteate näol.
    - Mobiilirakenduses on kasutajal võimalik näha eraldi turniiriteate detailvaadet.

- Mobiilirakenduses on kasutajal võimalik vaadata üritusega seotud kohti Tallinnas ja voorude toimumiskohti Tallinna Tehnikaülikooli linnakus.

Eesmärgi luua väitlusvõistluste tarbeks Ionic raamistikul põhinev mobiilirakendus ning seda administreeriv veebiliides võib lugeda saavutatuks. Rakenduse komponentide omavahelise suhtluse ja toimimise näol on näha põhilise funktsionaalsuste, ehk uudistevoo kuvamise ja haldamise toimimine ning oluliste asukohtade haldamine ja kuvamine kaardivaates.

Summary

The aim of this bachelor thesis was to create a mobile application based on Ionic framework for debating competition and an administrator's interface for the mobile application.

The necessary components which were developed for the debating application to function properly can be divided into three parts: the mobile application, the administration panel and the project's API. The work resulted in developing the most important functionalities – displaying notifications in news feed and the map view. The author had to establish a successful communication between different components of the application and find a way to display the tournament notifications and map view with location to the end user.

The author concludes that developing a hybrid mobile application is very similar to developing a regular web application which works in a browser. The author also finds that even if it is possible to quickly build and test the application in a browser, it is important that during the development of a hybrid application, it is regularly tested also on native devices to get a more comprehensive overview of all the functionalities.

The aim of this bachelor thesis was to create a mobile application and an administrator's inteface for one specific competition, but with further development it could be used as a template for other upcoming debating related events and competitions.

Main results:

- An administrator's interface which facilitates the work of a person administrating the mobile application during the competition.
    - Notifications in news feed are displayed in the administrator's interface.
    - New tournament notifications can be created through the administrator's interface.
    - Venues and round locations can be managed through the administrator's interface.
- A mobile application displaying the tournament feed and helping competitors to orientate in Tallinn and in campus of Tallinn University of Technology.

- The end user is able to see the regular tournament notifications and reminders of a starting round with a countdown timer in tournament feed.
- The end user is able to visit the detail view of a regular tournament notification.
- The end user is able to get information about the event venues and round locations.

The goal to create a mobile application and the administrator's interface could be regarded as a success. The ability to display and manage the tournament feed, managing important venues and locations and displaying them in the mobile application and administrator's panel with a supporting server could be seen functioning in the application.

11. Kasutatud kirjandus

[1] „Ionic (mobile app framework),“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework).

[2] Sass, „Sass (Syntactically Awesome StyleSheets),“ [Võrgumaterjal]. Available: http://sass-lang.com/documentation/file.SASS_REFERENCE.html.

[3] „What is Angular?,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/introduction.

[4] „Representational state transfer,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer.

[5] „Application programming interface,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Application_programming_interface.

[6] „Hypertext Markup Language,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/HTML.

[7] „CSS,“ [Võrgumaterjal]. Available: http://kuutorvaja.eenet.ee/wiki/CSS.

[8] „Asynchronous JavaScript and XML,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Ajax_(programming).

[9] „Hypertext Transfer Protocol,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[10] „Document Object Model,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Document_Object_Model.

[11] „JSON,“ [Võrgumaterjal]. Available: http://www.json.org/.

[12] „Data Access Object,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Data_access_object.

[13] „Vallaste e-teatmik,“ [Võrgumaterjal]. Available: http://vallaste.ee/.

[14] „URL,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/URL.

[15] „Structured Query Language,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/SQL.

[16] „Global Positioning System,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Global_Positioning_System.

[17] „Pivotal Tracker,“ [Võrgumaterjal]. Available: https://www.pivotaltracker.com.

[18] „International Data Corporation,“ August 2015. [Võrgumaterjal]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[19] „Objective-C,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Objective-C.

[20] „Swift (programming language),“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Swift_(programming_language).

[21] A. Ziflaj, „Native vs Hybrid App Development,“ 15 August 2014. [Võrgumaterjal]. Available: https://www.sitepoint.com/native-vs-hybrid-app-development/.

[22] „React Native,“ [Võrgumaterjal]. Available: https://facebook.github.io/react-native/.

[23] „Famous,“ [Võrgumaterjal]. Available: https://famous.co/.

[24] M. Lynch, „Ionic blog,“ 5 Jaanuar 2016. [Võrgumaterjal]. Available: http://blog.ionic.io/how-2015-went-for-ionic/.

[25] Apache, „Architectural overview for Cordova platform,“ [Võrgumaterjal]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[26] „What are Scopes?,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/scope.

[27] „jQuery,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/JQuery.

[28] „Hibernate (framework),“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Hibernate_(framework).

[29] „MySQL,“ [Võrgumaterjal]. Available: https://www.mysql.com/.

[30] „MariaDB,“ [Võrgumaterjal]. Available: https://mariadb.org/.

[31] „Uniform Resource Identifier,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

[32] „FasterXML - Jackson,“ [Võrgumaterjal]. Available: https://github.com/FasterXML/jackson.

[33] Oracle Corporation, „Introduction to RESTful Web Services and Jersey,“ 2010. [Võrgumaterjal]. Available: https://docs.oracle.com/cd/E19776-01/820-4867/6nga7f5ml/index.html.

[34] „Java JAX-RS,“ [Võrgumaterjal]. Available: https://jax-rs-spec.java.net/.

[35] „HTTP Methods: GET vs. POST,“ [Võrgumaterjal]. Available: https://www.w3schools.com/tags/ref_httpmethods.asp.

[36] Eclipse, „Jetty,“ [Võrgumaterjal]. Available: http://www.eclipse.org/jetty/.

[37] „Bootstrap,“ [Võrgumaterjal]. Available: http://getbootstrap.com/.

[38] „Apache HTTP Web Server Project,“ [Võrgumaterjal]. Available: https://httpd.apache.org/.

[39] A. McGivery, „Structure of an Ionic App,“ 3 August 2014. [Võrgumaterjal]. Available: http://mcgivery.com/structure-of-an-ionic-app/.

[40] „Model–view–controller,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[41] „Creating Custom Directives,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/directive.

[42] „ngRepeat,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/api/ng/directive/ngRepeat.

[43] S. Hameed, „Angular Timer,“ 2013. [Võrgumaterjal]. Available: https://siddii.github.io/angular-timer/.

[44] „Google Maps API,“ [Võrgumaterjal]. Available: https://developers.google.com/maps/.

[45] „cordova-plugin-geolocation,“ [Võrgumaterjal]. Available: https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/.

[46] „Testing your app,“ [Võrgumaterjal]. Available: http://ionicframework.com/docs/guide/testing.html.

[47] „NodeJS,“ [Võrgumaterjal]. Available: https://nodejs.org/en/.

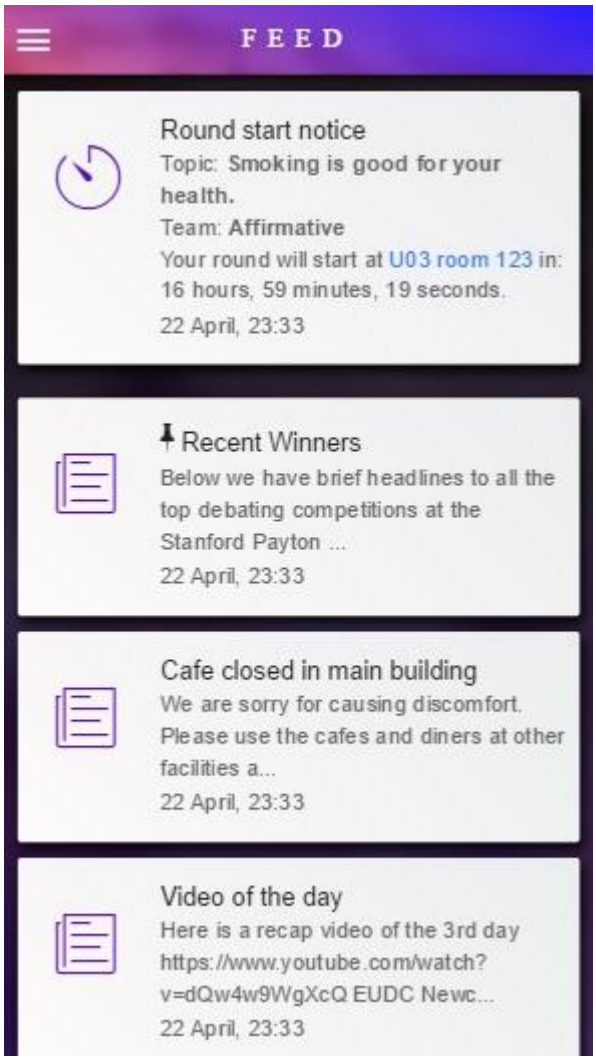
[48] „Live Reload App During Development,“ [Võrgumaterjal]. Available: http://ionicframework.com/docs/v1/cli/run.html.

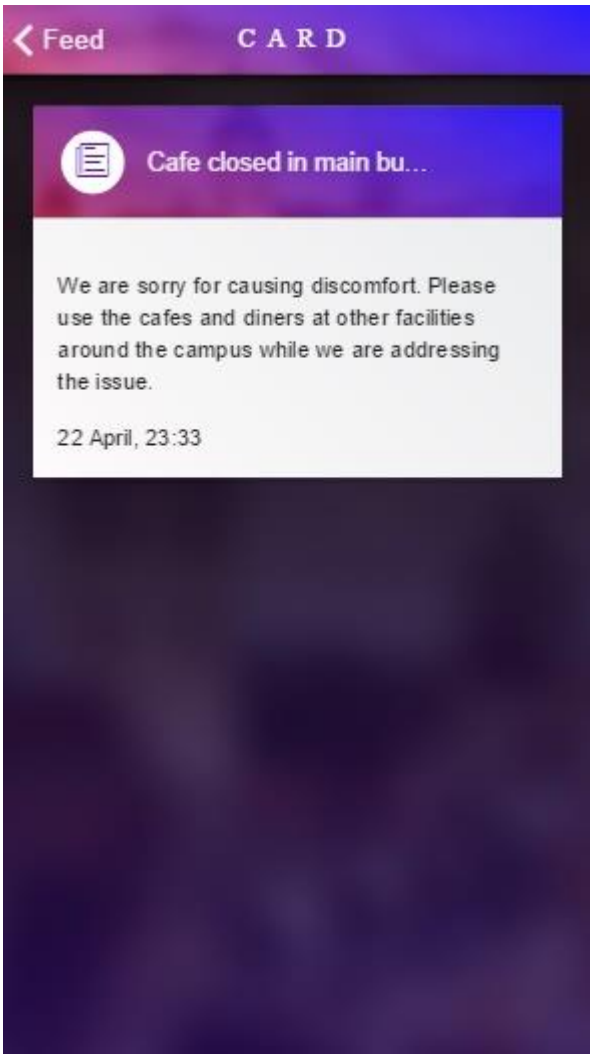
[49] „Jenkins,“ [Võrgumaterjal]. Available: https://jenkins.io/.

[50] „GIT,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Git.

Lisa 1

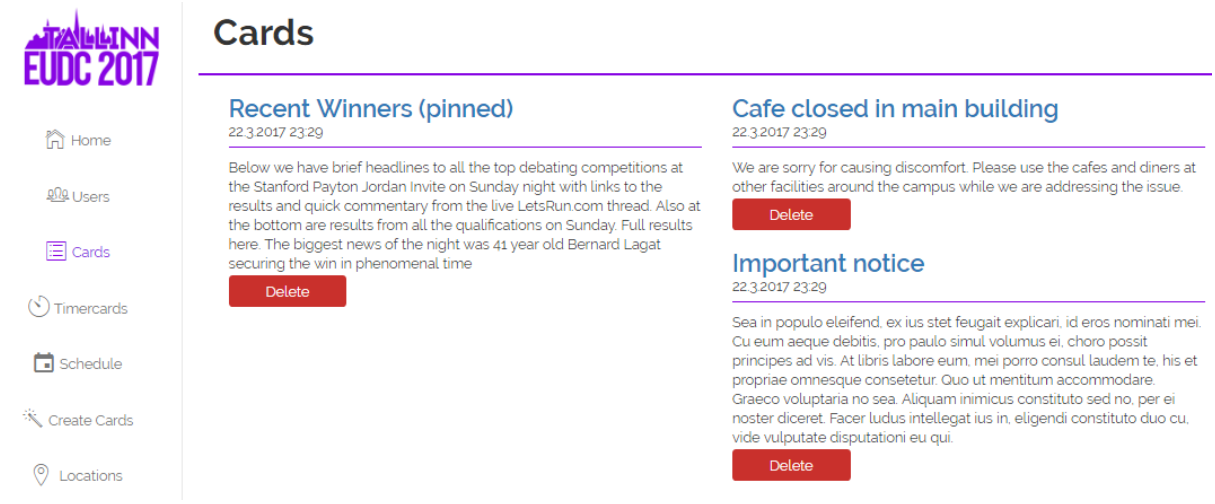
Uudistevoog ja ühe teate detailvaade.



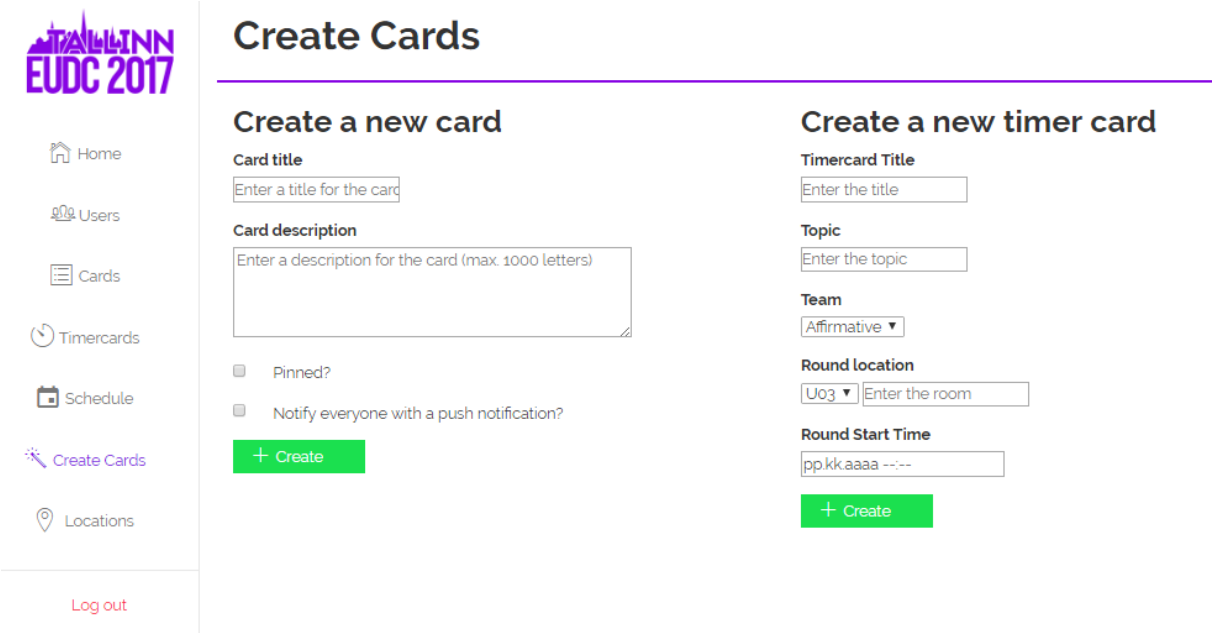


Lisa 2

Uudistevoog koos teadetega administraatori liideses.

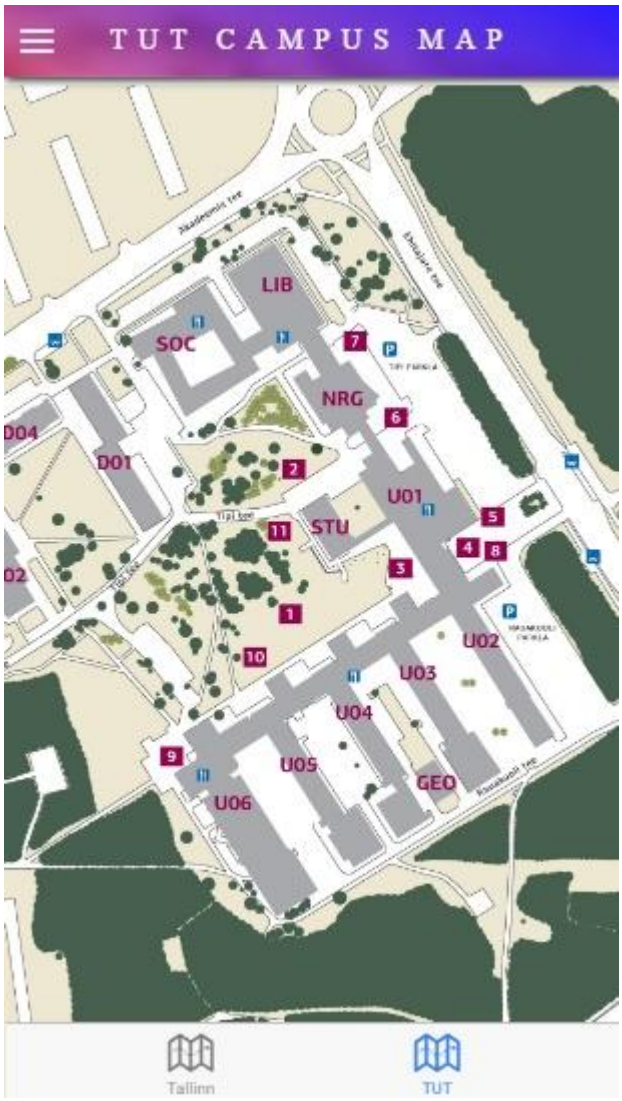


Uute teadete loomise vormid administraatori liideses.



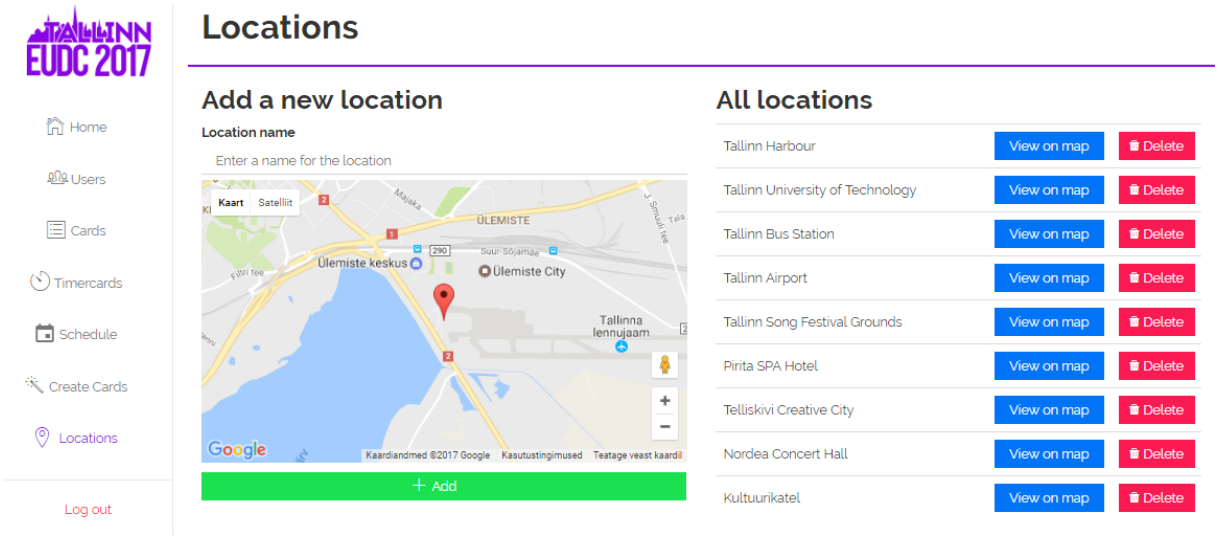
Lisa 3

Tallinna kaardi ja Tallinna Tehnikaülikooli linnaku kaardivaated.

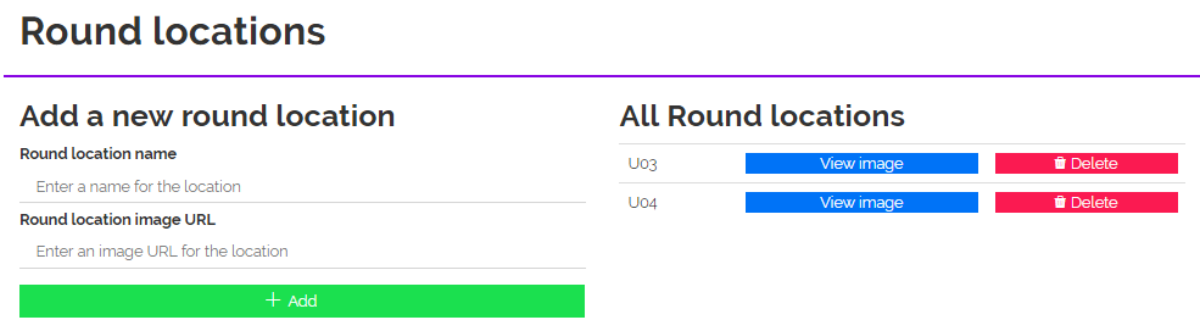


Lisa 4

Tallinna linnas olevate üritusega seotud kohtade haldamine administraatori liideses.



Vooru toimumiskohtade haldamine administraatori liideses.



Lisa 5

```
public class CardService {

    @Inject
    private CardDAO cardDAO;

    @Inject
    private UserService userService;

    public Card saveCard(Card card) {
        card.setUsers(userService.getAllUsers());

        card.setCreated(DateTime.now());
        return cardDAO.saveCard(card);
    }

    public List<Card> getAllCards() {
        return cardDAO.findAll();
    }

    public List<Card> getUsersCards(User user) {
        return cardDAO.findUsersCards(user);
    }

    public void deleteUserCard(User user, long cardId) {
        cardDAO.deleteUserCard(user, cardId);
    }
    public void deleteCardAsAdmin(long cardId) {
        cardDAO.deleteUserCardAsAdmin(cardId);
    }
}
```

Lisa 6

```
@Path("card")
public class CardResource {

    @Inject
    private CardService cardService;

    @Inject
    private OneSignalService oneSignalService;

    private SecurityContext securityContext;

    private static final AuthUtils authentication = new AuthUtils();

    @Context
    public void setSecurityContext(SecurityContext securityContext) {
        this.securityContext = securityContext;
    }


    @POST
    @Produces(MediaType.APPLICATION_JSON)
    public void addCard(Card card) throws Exception {
        if (card != null) {
            cardService.saveCard(card);

            if(card.getSendPushAll() == true){
            oneSignalService.sendAll(card.getTitle());
            }
        } else {
            throw new Exception("No card");
        }
    }
```

```
@GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Card> getAllCards() {
        AuthenticatedUser authenticatedUser =
authentication.getAuthUser(securityContext);

        if (authentication.isUserAuthenticated(authenticatedUser)) {
            return cardService.getUsersCards(authenticatedUser.getUser());
        } else {
            return cardService.getAllCards();
        }
    }

    @DELETE
    @Path("{cardId}")
    @Produces(MediaType.APPLICATION_JSON)
    public void deleteUsersCard(@PathParam("cardId") long cardId) {
        AuthenticatedUser authenticatedUser =
authentication.getAuthUser(securityContext);
        if (authentication.isUserRoleUser(securityContext)) {
            if (authenticatedUser != null) {
                cardService.deleteUserCard(authenticatedUser.getUser(),
cardId);
            }
        } else if (authentication.isUserRoleAdmin(securityContext)) {
            cardService.deleteCardAsAdmin(cardId);
        }
    }


}
```

Lisa 7

```
public class CardDAO {

    @Inject
    private EntityManager entityManager;
    @Inject
    private UserDAO userDAO;

    public List<Card> findAll() {
        return entityManager.createQuery(
                "SELECT c FROM Card c ORDER BY c.pinned DESC, c.created
DESC",
                Card.class).getResultList();
    }

    public Card saveCard(Card card) {

        Card merged;
        try {
            merged = entityManager.merge(card);
            entityManager.persist(merged);
        } catch (PersistenceException e) {
            e.printStackTrace();
            throw new RuntimeException("Exception when persisting card.");
        }

        return merged;
    }
    public Card testMakeCard(String title, String description) {
        Card card = new Card();
        card.setTitle(title);
        card.setDescription(description);
        card.setUsers(userDAO.findAll());
        card.setCreated(DateTime.now());

        saveCard(card);

        card = findAll().get(findAll().size() - 1);

        return card;
    }
```

```
public void delete(Card card) {
        entityManager.remove(card);
    }

    public List<Card> findUsersCards(User user) {
        TypedQuery<Card> findByUser = entityManager
                .createQuery("SELECT c FROM Card c LEFT JOIN FETCH c.users
as u WHERE u = :user",
                        Card.class);

        List<Card> cards = null;

        try {
            cards = findByUser.setParameter("user", user).getResultList();
        } catch (Exception e) {
            //ignore
            e.printStackTrace();

        }

        return cards;
    }
    public List<Card> findCardsByCardId(long cardId) {
        TypedQuery<Card> findById = entityManager
                .createQuery("SELECT c FROM Card c LEFT JOIN c.users as u
WHERE c.id = :card",
                        Card.class);

        List<Card> cards = null;

        try {
            cards = findById.setParameter("card", cardId).getResultList();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return cards;
    }
```

```
public void deleteUserCard(User user, long cardId) {
        entityManager
                .createNativeQuery("DELETE FROM Card_User WHERE user =
:user AND card = :cardId")
                .setParameter("user", user.getId())
                .setParameter("cardId", cardId).executeUpdate();
    }

    private void deleteCard(long cardId) {
        entityManager
                .createNativeQuery("DELETE FROM Card WHERE id = :cardId")
                .setParameter("cardId", cardId).executeUpdate();
    }

    private void deleteCardRelation(long cardId) {
        entityManager
                .createNativeQuery("DELETE FROM Card_User WHERE card =
:cardId")
                .setParameter("cardId", cardId).executeUpdate();
    }

    public void deleteUserCardAsAdmin(long cardId) {
        deleteCardRelation(cardId);
        deleteCard(cardId);
    }

}
```