Tallinna Ülikool

Digitehnoloogiate instituut

GURMEETEATER MOBIILIRAKENDUSE ARENDAMINE

Bakalaureusetöö

Autor: Vjatšeslav Torkin

Juhendaja: Jaagup Kippar

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

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

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

Tallinn 2017

Autorideklaratsioon

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

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Vjatšeslav Torkin (sünnikuupäev: 10.05.1987)

1. Annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Gurmeeteater mobiilirakenduse arendamine mille juhendaja on Jaagup Kippar säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, ...

Sisukord

Sissejuhatus

Tänapäeva meelelahutusel puuduvad piirid. Mõeldakse välja igasuguseid huvitavaid, imelikke, põnevaid, jaburaid tegevusi millega ennast lõbustada ning konkurents on suur. Seega tuleb välja mõelda ideid ning lahendusi mis tooksid teenused suurest hallist massist esile. Miks mitte teha seda läbi tänapäeva ülimenukate nutiseadmete? Just sellise ideega tuli ettevõttesse, kus käesoleva bakalaureusetöö autor töötab, klient kes soovis enda meelelahutusteenusele, Gurmeeteater, luua mobiilirakendus mis eristaks Gurmeeteatrit veel enam sellest suurest hallist massist.

Käesoleva bakalaureusetöö autor osutus Gurmeeteater mobiilirakenduse arenduse projekti vastutavaks isikuks ning arendajaks olema ja seetõttu tuli autoril idee ka konkreetse projekti kohta bakalaureusetöö kirjutada. Lisaks sellele tunneb autor suurt huvi mobiilirakenduste arendamise vastu ning usub, et arendusprotsessi dokumenteerimine annab edasiseks arenguks kindlasti teadmisi juurde.

Bakalaureusetöö eesmärgiks on kirjeldada Gurmeeteater mobiilirakenduse arendusprotsessi alustades kliendi vajaduste välja selgitamisest kuni rakenduse valmimise- ning reaalsetes tingimustes testimiseni.

Bakalaureusetöö jaguneb kolmeks osaks. Esimese osas toob autor välja kliendi vajadused ning nõudmised funktsionaalsustele. Teises osas kirjeldab projektitöö korraldamise poolt ning kolmandas osas keskendub pikemalt projekti arenduse etappidele ning rakenduse arendamise kirjeldamisele.

Mõistete loetelu

API - (Application Programming Interface). Teenuse pakkujate poolt hästi dokumenteeritud liides teenuse kasutamiseks (http://sproutsocial.com/insights/what-is-an-api/)

JavaScript - Veebiprogrammeerimise keel mis muudab veebilehed interaktiivseteks (https://www.thoughtco.com/what-is-javascript-2037921)

HTML - (HyperText Markup Language). Veebiprogrammerimise keel mida kasutatakse veebilehe sisu kuvamiseks (https://developer.mozilla.org/en-US/docs/Web/HTML)

CSS - (Cascading Style Sheets). Veebiprogrammeerimise keel mida kasutatakse kujunduse loomiseks (https://developer.mozilla.org/en-US/docs/Web/CSS)

WebKit - Veebilehitseja mootor (https://webkit.org/)

PHP - Vabavaraline veebiarenduse keel (http://php.net/manual/en/intro-whatis.php)

.NET - Microsofti operatsioonisüsteemi platvorm (http://www.webopedia.com/TERM/D/dot_NET.html)

NPM - JavaScripti pakikoguhaldur (https://www.npmjs.com/)

CMD - (Command Prompt Commands). Windows operatsiooni käsurealiides.

CMS - (Content Management System). Sisuhaldussüsteem (http://www.ithooldus.ee/kusimused-ja-vastused/mis-on-kodulehe-sisuhaldussuesteem-ehk-administreerimisliides)

cURL - Vahend millega on võimalik andmebaasist andmeid saata või pärida (https://www.lifewire.com/curl-definition-2184508)

1. Kliendipoolsed vajadused

Kliendi, Gurmeeteater ürituse eestvedaja, visioon on pakkuda inimestele elamust mis ühendab endas gurmeeõhtusööki ning interaktiivset teatrielamust, kus publik satub etenduse sündmuste keerisesse. Kuna me elame nutiajastus, tekkis kliendil idee luua Gurmeeteater mobiilirakendus. Mõte seisnes selles, et rakendus asendaks külastajate pileteid, muudaks piletikontrolli mugavamaks, võimaldaks publikul etendusest osa võtta, soovi korral nupuvajutusega teenindaja lauda kutsuda ning ürituse lõpus jätta tagasiside. Klient soovis, et rakendus oleks saadaval vähemalt Android ja iOS operatsioonisüsteemidega seadmetel.

1.1 Nõudmised funktsionaalsusele

Rakendus pidi olema lihtsasti kasutatav, kõik vajalik info oleks saadaval ühelt vaatelt. Kliendi soov oli, et rakenduse sisu oleks muutuv (ürituse logo, menüüvalikud, info, taust, värvid). Avamisel toimub mingit tüüpi sisse logimine, sisestatud andmete põhjal saab süsteem aru mis üritusega tegemist on ning vastavalt sellele kuvatakse ekraanile üritusega seotud sisu (personaalne tervitus, ürituse info, piletiinfo, ürituse asukoht, toidumenüü koos joogikaardiga, mõned peidetud valikud, mida oleks võimalik ürituse käigus avada ning nupp, mille abil saab külastaja teenindaja lauda kutsuda). Lisaks sellele soovis klient, et rakendusel oleks teavituste vastuvõtmise funktsionaalsus, et saaks kasutajale kindlates olukordades vajadusel teavitusi saata nii personaalseid kui üldiseid.

Vestluse käigus erinevate funktsionaalsuste kirjeldamisega selgus, et lisaks kliendi rakendusele oleks vaja luua ka administreerimise rakendus, mille abil saaks teostada piletikontrolli, jälgida teenindaja logi, kus on näha mis lauast ning istekohast tellimus tuli, vaadata külaliste nimekirja, aktiveerida kliendi rakenduses olevad peidetud valikud, saata välja teavitusi ning hallata administreerimise rakenduse kasutajaid.

1.2 Ajaline faktor

Kuna tegemist oli üritusesarjaga, millel olid määratud täpsed kuupäevad ning mitme kuu ulatuses kõik piletid välja müüdud, siis tuli arvestada ajalise faktoriga, milleks oli poolteist kuud esimese ürituse alguseni. Vastavalt sellele oli vaja kiiresti disaineri ning lisaarendajaga tööaeg kooskõlastada ning ülesanded ära jagada.

2 Projektitöö korraldus

Järgnevas peatükis toob käesoleva bakalaureusetöö autor välja Gurmeeteater mobiilirakenduse projektitöö korralduse. Kirjeldab kuidas toimus arendajatevaheline suhtlus, *front-end* ja *back-end* päringute kooskõlastamine, millised probleemid tekkisid, kuidas need lahendati ning suhtlus kliendiga.

2.1 Arendajatevaheline suhtlus

Projekti algusfaasis oli määratud töö autor, kui mitte arvestada disainerit ning kujunduse lahtilõikajat, projekti ainukeseks arendajaks. Arvestades kliendi nõudmisi ning ajafaktorit tehti otsus töö ära jagada kolme arendaja vahel. Üks arendaja tegeles andmebaasi ning *back-end* arendamisega, teine arendaja administreerimise rakenduse *front-end* arendamisega ning käesoleva töö autori ülesandeks oli arendada Gurmeeteater rakenduse *front-end*, organiseerida arendajate tööülesandeid ning suhelda kliendiga.

Vastava tööjaotuse tõttu on arendajatevaheline suhtlus väga tähtis ning pidi toimuma pidevalt, et ei tekiks arenduse käigus möödarääkimisi, mis võivad põhjustada koodi ümber kirjutamise ehk aja raiskamise. Ideaalis on hea teha arendajate vahelisi pistelisi koosolekuid, kus saab arutada arendamise etappe, tehtud tööd ning kooskõlastada järgneva arendustöö järjestust, et *back-end* ning *front-end* käiksid võimalikult käsi-käes ning valminud funktsionaalsuseid saaks võimalikult kohe ka testida. Lisaks sellele on hea kasutada mõnda suhtlusrakendust, mille abil saab kiireid muudatusi või tekkinud muresid omavahel hõlpsasti kooskõlastada. Üks selline populaarne rakendus, mida kasutavad väga paljud ettevõtted meeskonna siseseks suhtluseks, on Slack. Gurmeeteater mobiilirakenduse arendamise projekti raames kasutas töö autor arendajatega suhtlemiseks enamjaolt just Slack suhtlusrakendust, kuna *back-end* arendaja töötab kodukontoris ning satub ettevõte kontorisse harva.

Tahes-tahtmata tekib igal arendajal ajaga oma niinimetatud kodeerimise käekiri, mis mitme arendaja puhul sama projekti kallal töötamisel võib põhjustada segadust ning ajaraiskamist. Seetõttu on hea kaasarendajatega, lisaks pidevale suhtlusele, ka kooskõlastada koodi kirjutamise ning struktureerimise standardid mida järgides on kergem lugeda ning vajadusel muuta teine-teise koodi.

2.2 Millised probleemid tekkisid ning kuidas need lahendati?

Vaatamata sellele, et käesoleva töö autoril puudus projekti juhtimise kogemus, sujus projektitöö üllatavalt hästi ning täielikku kaost või suuri probleeme töö korraldamise käigus ei ilmnenud. Isegi *back-end* arendajaga, kellega toimus suhtlus peamiselt suhtlusrakenduse kaudu, sujus töö väga hästi.

Väiksemat tüüpi möödarääkimised siiski esinesid, peamiselt seoses kliendi vajadustega, kuid need lahenesid kliendiga kohtumisel. Suuremaks probleemiks, mis ilmnes peab autor asjaolu, mis on seotud rakenduse funktsionaalsusega, millest tuleb rohkem juttu järgnevates peatükkides. Kokkuvõtvalt öeldes on Gurmeeteatril olemas üldine veebileht mis on rakenduse andmebaasist ning *back-end*'ist eraldatud. Selleks, et kasutada Gurmeeteater veebilehe andmebaasis olevaid ürituse ning piletite andmeid, on loodud veebilehe *back-end*'i andmete ületoomise funktsioon. Seda funktsiooni arendas aga Gurmeeteater veebilehe arendaja. Juhtus olukord, kus rakenduse arendamise aeg kattus veebilehe arendaja puhkuse ajaga ning selgus, et andmete ületoomise funktsioonis ilmnesid vead ning arendaja ei delegeerinud enda tööd edasi teisele arendajale kes vajadusel saaks teda asendada. Sellega seoses tekkis väikene segadus ning probleemi lahendamiseks kulus, mitte küll pikk, kuid aeg mida oleks saanud efektiivsemalt ära kasutada. Kuna tegemist oli noore arendajaga, siis sellest olukorrast õppimine oli mõlemapoolne.

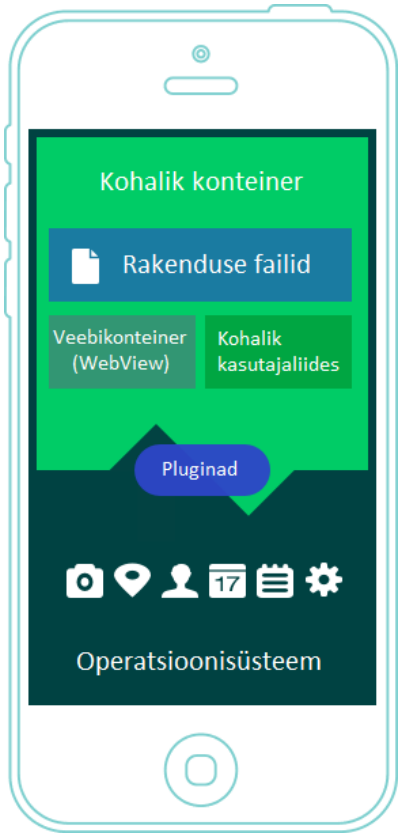
2.3 Suhtlus kliendiga

Sarnaselt arendajate vahelise suhtlusega toimus pidev suhtlus ka kliendiga, et viia klient kurssi rakenduse arengu faasidega, vajadusel funktsionaalsused üleräkida, kooskõlastada ideed ning ettepanekud. Enamjaolt toimus suhtlus kohtumiste näol, kuid kiirete küsimuste või ettepanekutega toimus suhtlus telefonitsi.

3 Arenduse etapid

Käesolevas peatükis kirjeldab autor Gurmeeteater mobiilirakenduse arenduse etappe, alustades arendusraamistike ning tehnoloogiate valikust, disaini kooskõlastamisest kliendiga kuni töötava lahenduse valmimiseni ning reaalsetes tingimustes testimiseni.

Mobiilirakenduse arendamisel on tänapäeval kaks peamist arendusmeetodit "*native application development*" ehk operatsioonisüsteemile omane mobiilirakenduse arendus või *"hybrid application development"* ehk hübriidmobiilirakenduse arendus, kus sisuliselt luuakse veebirakendus mida kuvatakse operatsioonisüsteemile omases veebikuvamise konteineris (iOS puhul UIWebView ja Android puhul WebView) ning ligipääs seadme riistvarale käib läbi pluginate ehk API'de mis võimaldavad JavaScript koodil suhelda operatsioonisüsteemile omase koodiga (vt. Joonis 1). (Ziflaj, 2014)



Joonis 1 Hübriidmobiilirakenduse toimimise diagramm

Operatsioonisüsteemile omase mobiilirakenduse arenduse eelis on funktsioneerimine, jõudlus ning kiirus, mis on tagatud operatsioonisüsteemile omase programmeerimiskeele kasutamisest. Lisaks sellele täielik ligipääs seadme riistvarale. Puudus on aga see, et

operatsioonisüsteemile omane mobiilirakenduse arendus nõuab tugevaid teadmisi konkreetse süsteemi arendamisest ning soovides arendada mitmele süsteemile tuleb tunda mõlemat. Lisaks sellele tuleb uuenduste sisseviimiseks muuta iga eraldi seisvat projekti koodi mis kõik kokkuvõttes on väga aega ja ressurssi nõudev. (Ziflaj, 2014)

Hübriidmobiilirakenduse arenduse eelis on see, et ühe koodiga on võimalik genereerida rakendus mitmele erinevale operatsioonisüsteemile ning rakenduse loomiseks piisab veebiarenduse kogemusest ning selliste programmeerimiskeelte tundmisest nagu HTML, CSS ja JavaScript. Mistõttu kulub arendamiseks vähem aega ning ressurssi. Peamiseks puuduseks on see, et hübriidmobiilirakendused on sõltuvad operatsioonisüsteemi omasest veebisisu kuvamise keskkonnast, mistõttu jõudlus, kiirus ning funktsionaalsus on piiratud ning erineb vastavalt sellele mis süsteemiga on tegemist. Seetõttu tuleb arendamise meetodi valimisel arvestada erinevaid faktoreid, võimalusi ning vajadusi. (Ziflaj, 2014)

3.1 Arendusraamistike ning tehnoloogiate valik

Gurmeeteater mobiilirakenduse arendamise kriteeriumeid ning arendajate võimalusi arvestades otsustati hübriidmobiilirakenduse kasuks. Arendusraamistiku valikul on nimekiri üsna pikk ning erinevaid arvustusi lugedes kipuvad korduma järgmised nimed:

- Ionic (https://ionicframework.com)
- React Native (https://facebook.github.io/react-native/)
- jQuery Mobile (https://jquerymobile.com)
- PhoneGap (http://phonegap.com)
- Kendo UI (http://www.telerik.com/kendo-ui)
- Native Script (https://www.nativescript.org)
- Intel XDK (https://software.intel.com/en-us/intel-xdk)

Igal raamistikul on oma head ja vead ning erinevad allikad järjestavad neid erinevalt, kuid lõppude lõpuks jääb valik alati arendaja eelistuse taha. Käesoleva bakalaureusetöö autoril on kogemust sellise raamistikuga nagu Apache Cordova, mida kasutavad rakenduste genereerimisel alusena ka mitmed eelpool mainitud raamistikud (Ionic, PhoneGap, Kendo UI,

Intel XDK). (Cordova, kuupäev puudub) Sisuliselt on Apache Cordova ning PhoneGap üks ja sama raamistik.

3.1.1 Apache Cordova ehk Phonegap?

Selleks, et rääkida Apache Cordovast tuleb alustada PhoneGapist. PhoneGap loodi Nitobi ettevõtte poolt aastal 2008. Aastal 2011 Adobe omistas Nitobi ning seoses sellega PhoneGap lähtekood annetati *Apache Software Foundation*'le (ASF) ning projekt sai endale uue nime Apache Cordova. (Camden, 2016)

Sisuliselt nagu ka eelnevalt mainitud käsitletakse neid kahte kui ühte ja seda sama, need kasutavad samu tehnoloogiaid ning isegi pluginad toimivad ühtemoodi mõlemal raamistikul. Erinevus seisneb selles, et PhoneGap'i omab Adobe ning vaatamata sellele, et nad lubavad, et PhoneGap jääb vabavaraks, ei saa selles täiesti kindel olla. Lisaks sellele kasutab PhoneGap natuke teistsugust käsurealiidest (ingl. k. CLI ehk Command Line Interface) ning omab lisavidinaid mis väidetavalt muudavad rakenduse arendamise palju mugavamaks. Sellegipoolest kasutab PhoneGap enda nn mootorina Apache Cordova't nagu paljud veebilehitsejad kasutavad WebKit'i. (Camden, 2016)

Käesoleva töö autor eelistab Apache Cordovat seetõttu, et Cordova on võrreldes teiste raamistikega mis kasutavad mootorina Cordovat, algrakenduse loomisel struktuuri poolest nn puhtam. Nii nagu ka algrakendusi nimetatakse "tühi" rakendusteks, siis Cordova puhul võib seda ka väita, et algrakendus on tühi ning arendajal on võimalik oma käe järgi kujundust, komponente ning funktsionaalsuseid lisada.

3.1.2 *Back-end* arendusraamistik Laravel

Andmebaasiga suhtluse loomisel kasutas *back-end* arendaja PHP-raamistikku Laravel. See on avatud lähtekoodiga veebiarendusplatvorm, mis sündis juunis 2011. Laraveli autor, Taylor Otwell, kasutas nendel aegadel igapäevaselt PHP raamistikku CodeIgniter, kuid tundis, et CodeIgniter raamistikul puuduvad tema arvates veebiarenduses hädavajalikud funktsionaalsused nagu sisseehitatud autentifitseerimine (kasutajate sisse ja välja logimine) ning üleüldiselt ei saanud ta lisada kõiki vajalikke omadusi ilma, et peaks raamistiku sisekoodi kallale minema. Ta soovis midagi puhtamat, lihtsamat ning paindlikumat. Need vajadused ning Taylor'i .NET taust panidki aluse raamistikule milleks sai Laravel. (O'Brien, 2016)

Autor küsitles arendajat seoses sellega miks *back-end* arendamisel osutus valituks just Laravel raamistik. Järgnevalt toob autor välja mõned põhjused miks tuleks kaalutleda Laravel raamistiku kasutusele võttu.

- Võrreldes viit populaarset PHP raamistiku (Phalcon, CodeIgniter, Zend, CakePHP, Laravel) otsinguid Google Trend statistika lehel on Laravel tänapäeva kõige otsituim raamistik (Google Trends, kuupäev puudub) mis tähendab seda, et huvi ning kasutajate kogukond on raamistikul väga suur.

- Rikkalik dokumentatsioon mis on abiks nii alustavale kui edasijõudnud raamistikuga arendajale.

- Laravelil on mugav CLI - Artisan (*Command-Line Interface* ehk käsurealiides), kuid Laravel nõuab, et seda ka kasutatakse. Näiteks luues käsitsi kontroller ning üritades routeris (kus defineeritakse, mobiilirakenduse puhul, *front-end* pöördumised *back-end* kontrollerite pihta) selle kontrolleri pihta pöördumised saata, siis see ei õnnestu. Läbi CLI luues kontrollerit Laravel registreerib selle kontrolleri ning tänu sellele saab ka kasutada kontrollereid routeris.

- Andmebaasi migreerimine on lihtne ja mugav kasutades selleks migratsioonifaile

3.1.3 AngularJS

AngularJS on kliendipoolne JavaScript raamistik mis toetab MVC (Model View Controller ehk mudel vaade kontroller) arhitektuuri ning on mõeldud veebirakenduste loomiseks. Selle abil on võimalik muuta staatiline HTML dünaamiliseks. Täpsemini laiendades HTML'i võimalusi kasutades selleks sisseehitatud omadusi ning komponente. (TutorialsTeacher.com, kuupäev puudub)

Kuna Cordova kasutab veebiarenduse standard keeli HTML, CSS ja JavaScript, siis AngularJS sobib ideaalselt kasutamiseks hübriidrakenduse loomisel. Lisaks sellele on AngularJS kasutamine peaaegu, et tavaks muutunud Apache Cordova mootorit kasutavate raamistike seas ning on lisatud ka dokumentatsioonidesse kui raamistiku üks komponentidest.

3.1.4 Node.js

Apache Cordova installimiseks on eelnevalt vajalik installida arvutisse Node.js, mis on esimene nõue Cordova dokumentatsioonis. Node.js on serveripoole raamistik mis on ehitatud Google Chrome JavaScript mootori peale ning mida enamjaolt kasutavad arendajad tema käsurealiidese NPM (Node Package Manager ehk Node pakkide haldur), mis on rikkalik pakkide või moodulite kogu, tarbeks. (Tutorialspoint, kuupäev puudub) Üks sellistest pakkidest on ka Apache Cordova mida on võimalik installida kasutades NPM käsurealiidest.

3.1.5 Firebase Cloud Messaging (FCM)

Gurmeeteater mobiilirakenduse üheks soovitud funktsionaalsuseks oli *push notification'ite* (serveripoolsed teavitused) saatmine ning saamine. Selleks on olemas Google all olev keskkond nagu Firebase, mis on mobiili- ja veebirakenduste arenduskeskkond ning üks nende tasuta teenustest on Firebase Cloud Messaging (Firebase pilvesõnumid) mis võimaldab kergelt serveripoolsete teavituste lisamist arendatavale rakendusele. Firebase toetab erinevatele platformidele arendamist mis sobib Gurmeeteater hübriidrakendusele ideaalselt. (Firebase, 2017)

3.1.6 Bootstrap

Kujunduse poole pealt on Gurmeeteater mobiilirakenduse arendamisel kasutatud Bootstrapi raamistikku. Bootstrap on võimas *front-end* raamistik mis on loodud kiirendama ning lihtsustama veebiarendust just kujunduse poole pealt. See sisaldab HTML ja CSS malle tavapäraste kasutajaliideste loomiseks nagu vormid, nupud, tabeleid, häire teateid, väljad, animeeritud listid ja muud kujundamist lihtsustavaid ning atraktiivsemaks muutvaid funktsionaalsusi. (Tutorial Republic, kuupäev puudub)

3.2 Disaini kooskõlastamine kliendiga

Disaini kooskõlastamise jaoks valmistas ettevõtte disainer kolm vaadet mis illustreerisid kliendile rakenduse põhifunktsionaalsusi (vt. Lisa 1, Lisa 2 ja Lisa 3). Disaini kooskõlastamisega probleeme ei tekkinud. Kujundus, mille disainer valmis meisterdas, meeldis kliendile väga ning muudatusi disaini osas õnneks tegema ei pidanud. Edasi liikusid disaini failid kujunduse lahtilõikaja kätte ning sealt juba HTML, CSS ja osaliselt JavaScript kujul arendaja kätte.

3.3 Rakenduse arendus

Nagu eelnevates peatükkides märgitud sai oli Gurmeeteater mobiilirakenduse arendusprojekti raames vaja valmis meisterdada kaks rakendust (Gurmeeteater rakendus ning Administreerimise rakendus). Käesoleva töö autor tegeles peamiselt Gurmeeteater rakenduse arendamisega mistõttu keskendutakse järgnevas peatükis just selle rakenduse arenduse kirjeldamisele.

3.3.1 Apache Cordova eelseadistamine

Enne rakenduse arendamise kallale minemist tuleb eelnevalt teha arvutis eelseadistusi, mis on vajalikud Apache Cordova raamistiku kasutamiseks. Kuna käesoleva bakaureusetöö eesmärgi alla kuulub vaid arendusprotsessi kirjeldus, siis toob autor välja vaid eelseadistamiseks vajalikud põhitegevused ning lisab juurde viited põhjalikumatele juhenditele.

Esimese sammuna tuleb allalaadida ning installida Node.js seda saab teha külastades nende veebilehte[1]. Edasi saab Node.js käsurealiidese kaudu allalaadida ning installida Apache Cordova. Selleks tuleb avada Windows operatsioonisüsteemiga seadme puhul CMD (Command Prompt) või OS X operatsioonisüsteemiga seadme puhul *Terminal* ning trükkida järgmine käsk: *npm install -g cordova* (vt. Joonis 1), kus *-g* tähendab seda, et Cordova installitakse globaalselt, et oleks võimalik ükskõik mis kausta navigeerides alati kutsuda välja Cordova käsklusi.

```
C:\Users\Slav>npm install -g cordova_
```

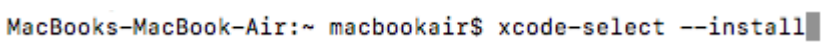
Joonis 2 Käsurealiidese kaudu Cordova installimine

Vastavalt sellele millisele mobiiliplatvormile plaanitakse mobiilirakendust arendada tuleb veel nende platformidele omased seadistused ära teha. Androidi puhul tuleb allalaadida ning installida Java Development Kit (JDK ehk Java arenduspakk) ning Android Studio, lisaks sellele Android Studio kaudu installida lisa pakke mis on seotud Androidi operatsioonisüsteemi versioonidega. Täpsemad juhised leiab Cordova kodulehelt[2]. OS X puhul tuleb App poest alla laadida Xcode ning käsurea võimaluste käivitamiseks tuleb lisaks

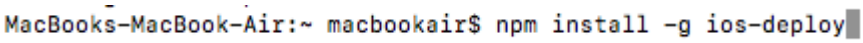
[1] https://nodejs.org/en/download/

[2] https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html

veel allalaadida mõned lisavahendid, selleks tuleb trükkida käsureale järgnevad käsklused: *xcode-select --install* (vt Joonis 3) ning *npm install -g ios-deploy* (vt Joonis 4). Täpsemad juhised vahednite kohta, kui ka seadistamise kohta leiab Cordova kodulehelt[3].



Joonis 3 OS X Cordova eelseadistamise käsurea vahendi lisamine



Joonis 4 OS X käsurealt ios-deploy vahendi installimine

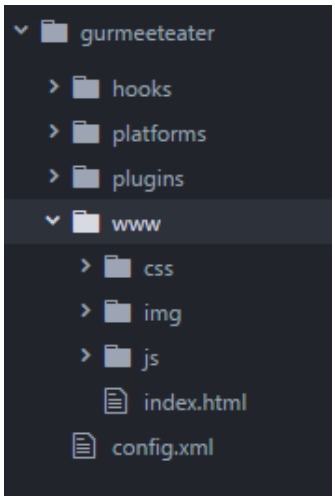
3.3.2 Algrakenduse loomine

Rakenduse loomisel tuleb esimese asjana luua Cordova algrakendus ehk tühirakendus mis on iga Cordova projekti aluseks. Seda saab teha kasutades käsurealiidest järgneva käsuga: *cordova create gurmeeteater com.websystems.gurmeeapp gurmeeapp* (vt. Joonis 5), kus *"gurmeeteater"* on projekti kausta nimi, *"com.websystems.gurmeeapp"* on rakenduse identifikaator ning *"gurmeeapp"* on rakenduse pealkiri.

Joonis 5 Cordova algrakenduse loomine

Sellega genereerib Cordova raamistik uue projekti kausta (gurmeeteater) ning uue rakenduse failistruktuuri (vt Joonis 6). Kaust kus toimub hübriidmobiilirakenduse peamine arendus on "www" kaust kus hoitakse kõiki rakenduse kujundusfaile, kontrollereid ning mallifaile.

[3] https://cordova.apache.org/docs/en/latest/guide/platforms/ios/index.html



Joonis 6 Cordova algrakenduse strukuur

Selleks, et saaks rakendust soovitud platvormile ehitada tuleb eelnevalt lisada kõik vajalikud platvormid (iOS platvormile saab rakendust ehitada ainult OS X süsteemiga seadmetelt). Lisamiseks tuleb rakenduse projekti kaustast trükkida käsureale järgnev käsk: *cordova platform add android --save* (vt. Joonis 7), lisades lõppu *--save* salvestab lisatud platvormi andmed "config.xml" faili.

```
C:\Users\Slav\Workspace\gurmeeteater>cordova platform add android --save
```

Joonis 7 Cordova projektile platvormi lisamine

Gurmeeteater rakenduses soovib autor kasutada *splashscreen'i* ehk laadimislehte mida kuvatakse rakenduse käivitamisel seda saab teha plugina abil mille nimi on *cordova-plugin-splashscreen*. Selleks tuleb trükkida käsureale pluginate lisamise käsklus: *cordova plugin add cordova-plugin-splashscreen --save* (vt Joonis 8).

```
C:\Users\Slav\Workspace\gurmeeteater>cordova plugin add cordova-plugin-splashscreen --save
```

Joonis 8 Cordova projektile plugina lisamine

Cordova rakenduse ehitamiseks tuleb trükkida käsureale, kindla platvormile ehitamisel näiteks *cordova build android* (vt. Joonis 9) või kõikidele platvormidele ehitamisel lihtsalt *cordova build* (vt Joonis 10). Õnnestumise korral tuleb lõpus teade *"Build successful"* (ehitamine õnnestus) ning ehitatud rakenduse faili asukoht (vt Joonis 11).

```
C:\Users\Slav\Workspace\gurmeeteater>cordova build android
```

Joonis 9 Cordova rakenduse ehitamine kindlale platvormile

```
C:\Users\Slav\Workspace\gurmeeteater>cordova build
```

Joonis 10 Cordova rakenduse ehitamine kõikidele platvormidele

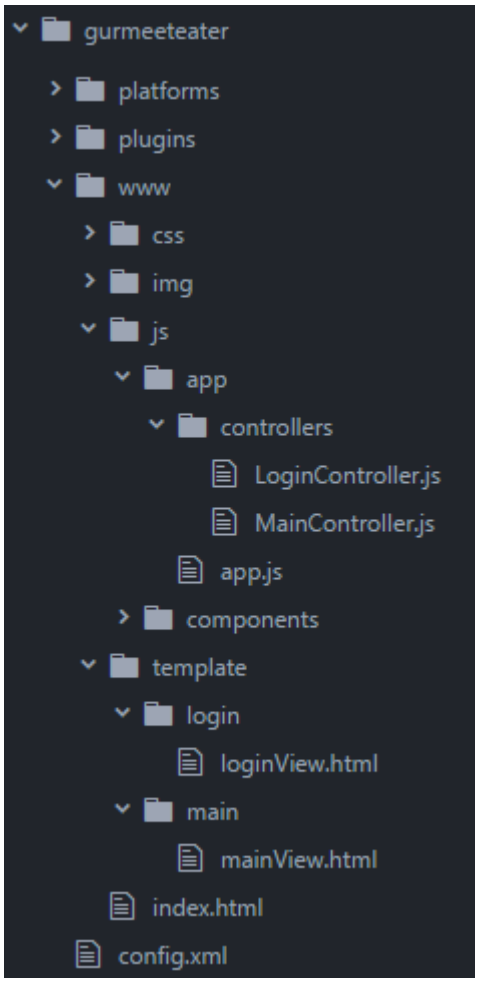
```
BUILD SUCCESSFUL

Total time: 1 mins 16.748 secs
Built the following apk(s):
        C:/Users/Slav/Workspace/gurmeeteater/platforms/android/build/outputs/apk/android-debug.apk
```

Joonis 11 Cordova rakenduse ehitamise õnnestumise teade

3.3.3 AngularJS ning Gurmeeteater rakenduse struktuuri lisamine

Järgnevalt oli vaja paika panna Gurmeeteater rakenduse struktruur ning vajalikud mallid ja kontrollerid selleks tuli kasutusele võtta AngularJS ning luua selle abil rakenduse põhi. Struktuurselt oli paika pandud, et Gurmeeteater rakendusel tuleb kaks põhivaadet - "Login View" ehk sisselogimise vaade ning "Main View" ehk põhivaade. Selleks, et struktuur oleks puhas ning selge jagas autor vaated eraldi mallifailidesse (loginView.html ning mainView.html) ja igale mallifailile lisas vastavad kontrollerid (LoginController.js ning MainController.js). Lisaks sellele muutis autor Cordova poolt vaikimisi loodud "index.js" fail "app.js" failiks, mis hakkab olema peamine Angular moodulfail (vt Joonis 12).



Joonis 12 Gurmeeteater struktuur peale moodulfaili, mallide ning kontrollerite lisamist

AngularJS rakenduse struktureerimisel tuleb esimese asjana paika panna "app.js" failis *angular.module* ning *router* ehk ruuter mille abil saab navigeerida ühelt vaatelt teisele, selleks on Angular'il olemas lisamoodul nimega *ui.router*. Ruuteri määramiseks tuleb lisada "app.js" faili lõppu konstruktorfunktsiooni *.config*, millesse on süstitud Angular komponendid *$stateProvider* ning *$urlRouterProvider*, ja defineerida funktsioonis *$stateProvider'i* abil olekud (*.state'd*). Olekutele tuleb määrata nimed mille abil on võimalik hiljem nende poole pöörduda ning vastavalt vajadusele lisaväärtusi nagu vastava oleku ehk vaate mallifaili asukoht (vt. Koodinäide 1).

```
"use strict";

var app = angular.module('GurmeeApp', ['ui.router']);

app.run(['$rootScope', function ($rootScope) {

    var initialize = function () {};

    initialize();
}]);

app.config(function ($stateProvider, $urlRouterProvider) {
    //Here you define states
    $stateProvider
        .state('login', {
            url: '/login',
            templateUrl: 'template/login/loginView.html'
        })
        .state('main', {
            url: '/main',
            views: {
                '': {
                    templateUrl: 'template/main/mainView.html'
                }
            }
        });
    //For any unmatched url direct to loginView
    $urlRouterProvider.otherwise("/login");
});
```

Koodinäide 1 Angular moodulfaili põhja ning ruuterite määramine

Edasi tuleb muuta "index.html" fail ning sisustada mallifailid ning kontrollerid. "index.html" fail on kõikide mallide alusfail mis käivitatakse kõige esimesena ning selleks, et rakendus käivitaks Angular mooduli tuleb see "index.html" failis välja kutsuda. Seda saab teha lisades, Angular mooduli poole pöördumise, käskluse *ng-app="MooduliNimi"*, Gurmeeteater rakenduse puhul vastavalt siis *ng-app="GurmeeApp"*. Vaated kuvatakse eraldi * * konteinerites kasutades Angular käsklust *ui-view* (vt. Koodinäide 2).

```
<!DOCTYPE html>
<html ng-app="GurmeeApp">
    <head>
        <!--<meta http-equiv="Content-Security-Policy" content="default-
src 'self' data: gap: https://ssl.gstatic.com 'unsafe-eval'; style-src
'self' 'unsafe-inline'; media-src *; img-src 'self' data: content:;">-->
        <meta name="format-detection" content="telephone=no">
        <meta name="msapplication-tap-highlight" content="no">
        <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width">
        <link rel="stylesheet" type="text/css" href="css/index.css">
        <title>GurmeeTeater</title>
    </head>
    <body>
        <div style="height: 100%;" ui-view>
        </div>
        <!-- Angular.js and Bootstrap components -->
        <script type="text/javascript"
src="js/components/angular.js"></script>
        <script type="text/javascript" src="js/components/angular-ui-
router.min.js"></script>

        <!-- Cordova -->
        <script type="text/javascript" src="cordova.js"></script>
        <!-- App files -->
        <script type="text/javascript" src="js/app/app.js"></script>

        <!-- Controllers -->
        <script type="text/javascript"
src="js/app/controller/LoginController.js"></script>
        <script type="text/javascript"
src="js/app/controller/MainController.js"></script>
    </body>
</html>
```

Koodinäide 2 "index.html" faili muutmine

Viimase sammuna tuli sisustada ning omavahel siduda mallifailid ja vastavad kontrollerid. Kontrollerifailis tuli lisada eelnevalt määratud *angular.module* muutujale konstruktorfunktsioon *.controller*, määrata sellele nimi ja süstida kaasa vajalikud komponendid mida kontrolleris hiljem kasutama hakata (vt. Koodinäide 3). Selleks, et mallifail siduda loodud kontrolleriga tuleb kasutada Angular'i käsklust *ng-controller="MainController"* (vt. Koodinäide 4).

```
"use strict";
/**
 * Main controller
 */
app.controller('MainController', ['$scope', '$state', '$rootScope',
    function ($scope, $state, $rootScope) {
        this.initialize = function () {

        };

        this.initialize();
}]);
```

Koodinäide 3 "MainController" kontrolleri loomine

```
<!-- template/main/mainView.html -->
<div ng-controller="MainController">

</div>
```

Koodinäide 4 "mainView" mallifaili sidumine kontrolleriga

3.3.4 Kujunduse külgepanek

Selleks ajaks kui Angular põhja paika panemine valmis sai, jõudsid lahtilõigatud kujunduse failid töö autori kätte ning võis hakata rakendusele kujunduse külge panema. Lahtilõigatud failid on iseenesest juba HTML, CSS ning JavaScript failid mis tuli autoril sättida sedasi, et need Gurmeeteater rakenduse loogikat arvestades sama moodi toimiksid. Kuna "index.html" on vaid konteineriks vaadetele, siis kõik muudatused ja lisamised toimusid vaadete mallifailides.

Esimene vaade, mida kasutaja rakenduse käivitamisel näeb on sisselogimise vaade, kus vastavalt kliendi soovidele (kasutajad logivad sisse piletikoodi alusel, mille nad saavad kui ostavad ürituse pileti) on põhi elemendid ekraani ülemisel poolel Gurmeeteatri logo, piletikoodi sisestamise *input* väli, selle all sisselogimise nupp ning kõige all link mis viib Gurmeeteater veebilehele. Selleks ei pidanud tegema muud kui tõsta HTML kood *"loginView"* mallifaili, määrata õigetele *div*'dele õiged klassid (vt. Koodinäide 5), tõsta kujunduse failidega kaasa antud sõltuvused (nagu "bootstrap.js", "jquery.js") projekti sobivasse kausta ning süstida need sõltuvused "index.html" faili (vt. Koodinäide 6).

```
<!-- template/main/loginView.html -->
<div class="wrap login-page" ng-controller="LoginController">
    <div class="page-wrapper footer-padding">
        <div class="login-wrapper">
            <div class="login-logo">
                <img src="img/logobig.png" />
            </div>
            <div class="login-form">
                <form>
                    <input type="code" placeholder="Sisesta
piletiparool" class="ws-input top-input" />
                    <input type="button" value="Logi sisse" class="ws-
input login-btn bot-input"/>
                </form>
            </div>
            <a href="http://www.gurmeeteater.ee/" class="official-web-
link">Ava Gurmeeteatri veebileht</a>
        </div>
    </div>
</div>
```

Koodinäide 5 Sisselogimise mallifaili kood peale kujunduse lisamist

```
<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script type="text/javascript" src="js/components/jquery.js"></script>
<script type="text/javascript"
src="js/components/bootstrap.min.js"></script>

<!-- Class -->
<script type="text/javascript" src="js/components/flowtype.js"></script>
<script type="text/javascript"
src="js/components/jquery.viewportchecker.min.js"></script>
<script type="text/javascript" src="js/components/animations.js"></script>
```

Koodinäide 6 Kujundusega kaasas olevate sõltuvuste süstimine "index.html" faili

Teise vaate puhul muutusid asjad juba keerulisemaks kuna jällegi tuli arvestada kliendi soovi milleks oli, et üritusevaade ehk *"mainView"* pidi olema võimalikult muutuva sisuga, vastavalt sellele mis ürituse piletikoodiga sisse logiti. Seega kujunduse paika panemine toimus paralleeleselt funktsionaalsuste lisamisega.

3.3.5 Funktsionaalsuste kirjeldus

Arvestades kliendi konfidentsiaalsuspoliitikaga tohib käesoleva töö autor vaid kirjeldada põhifunktsionaalsusi. Koodijuppe lisab autor näidistena osade funktsionaalsuste kohta ning vajadusel kokkuleppel tööautoriga on võimalik kindlate funktsionaalsuste koodiga tutvuda.

Nagu eelnevas peatükis mainitud sai soovis klient, et rakenduse sisu oleks muutuv. Sisuliselt tähendas see, et võimalikult palju sisust tuleks andmebaasist, seega esimese asjana oli vaja *back-end* arendajaga kokku leppida loogika mis moodi see ühelehelise rakenduse puhul

toimiks. Selge oli see, et kliendi soov oli midagi CMS (Content Management System ehk sisuhaldussüsteem) sarnast kus saaks sisu hallata vastavalt ürituse vajadusele. Kahjuks arvestades paika pandud ajafaktorit polnud esimese ürituse raames seda võimalik teostada, samas eeltööd oli võimalik õiges suunas liikumiseks siiski teha. Peale mõningat *back-end* arendajaga läbirääkimist jõudsime järeldusele, et üldine kujundus peab jääma samaks kuna täielikult dünaamiliseks isegi kui on võimalik mobiilirakendust teha, siis on vähe tõenäoline, et see vastaks Android Play poe ja iOS App poe tingimustele. Mida saab teha on muuta värve, tausta, logosid, teksti, menüüvalikute arvu, pealkirju ning samuti ka menüüvalikute sisu. Nüüd kui strateegia oli paigas tuli hakata funktsionaalsuseid lisama.

Esimese asjana oli vaja valmis teha sisse logimise ning kasutaja autentimise funktsionaalsus. Kuna sisse logimine pidi toimima piletikoodi alusel, siis oli vaja Gurmeeteater veebilehe andmebaasist kätte saada piletite ning üritustega seotud andmed. Selleks tegi Gurmeeteater veebilehe arendaja eksport kontrolleri mille pihta oli võimalik päringuid teha ning mis tagastas JSON massivi kujul kõik vajalikud andmed ja Gurmeeteater mobiilirakenduse *back-end* arendaja tegi omakorda import kontrolleri mis teostab eksport kontrolleri pihta päringuid ning tagastatud andmed paigutab õigetesse andmebaasi tabelitesse laiali. Nüüd, kui andmed olid olemas, oli võimalik luua piletikasutajate kontroller "TicketUserCtrl.php" ning lisada sisselogimise funktsioon (vt. Koodinäide 7) mis tagastab massiivi, milles on piletiomaniku info, ürituse info, üritusega seotud valikumenüü ja muu info. Põhimõtteliselt kõik mis on "mainView" vaate sisustamiseks vajalik.

```
public function login(Request $request)
{
	if($request->has('ticketCode') && !$request->input('ticketCode'))
return $this->error('No Ticket code');

	$user = TicketUser::where('ticketCode', '=', $request-
>input('ticketCode'))->first();

	if(count($user) == 0) return $this->error('No ticket found');
	$token = TicketUser::createToken();

	if(isset($token['error'])) return $this->error($token['error'] . '
Please try again.');

	$user->token = $token;
	if($request->has('pushToken')) $user->pushToken = $request-
>input('pushToken');
	$user->save();
	TicketUser::where('pushToken', '=', $request->input('pushToken'))-
>where('id', '!=', $user->id)->update(array('pushToken' => null));

	session([
		'token' => $user->token,
		'ticketCode' => $user->ticketCode,
		'eventId' => EventTicket::find($user-
>eventTicketId)['eventId'],
		'userId' => $user->id
	]);
	return Response::json($this->success(array('user' => $user)));
}
```

Koodinäide 7 Back-end kasutaja sisselogimise funktsioon

Front-end ehk rakenduse poole pealt tuli samuti luua sisselogimise kontrollerisse funktsioon mis pöördub *back-end* funktsiooni poole. Selleks tuli lisada "LoginController"'isse *scope* funktsioon nimega *login* kus toimuks Angular *$http POST* pöördumine *back-end login* funktsiooni poole, andes kaasa *ticketCode* ning *pushToken* (tuleb veel käesolevas peatükis teemaks) (vt. Koodinäide 8). "loginView" vaates input väljale tuli lisada Angular käsklus *ng-model* ning anda sellele *scope* muutuja mille abil toimub suhtlus kontrolleri ja mallifaili vahel ja nupu väljale lisada käsklus *ng-click* ning anda sellele *scope* funktsiooni nimi (vt. Koodinäide 9).

```
$scope.login = function () {
	$http.Post(service.url + 'public/TicketUserCtrl/login', {
			ticketCode: $scope.form.code,
			pushToken: storage.localToken
		})
		.then(this.loginSuccess, this.loginSuccess);
}.bind(this);
```

Koodinäide 8 "LoginController"'i sisselogimise funktsioon

```
<input type="code" placeholder="Sisesta piletiparool" ng-model="form.code"
class="ws-input top-input" />
<input type="button" value="Logi sisse" class="ws-input login-btn bot-
input" ng-click="login()"/>
```

Koodinäide 9 "loginView" vaate kood peale Angular ng-model ning ng-click käskluste lisamist

Back-end poolt sisselogimise pöördumisega tagastatud andmed salvestatakse vastavalt *window.localStorage*'sse (veebilehitseja lokaalne ruum mida ei tühjendata ka, siis kui rakendus läheb kinni, hea kasutada näiteks autentimisel) ning Angular *$rootScope* (globaalne *scope*, mida saab kasutada kõikides vaadetes ja kontrollerites) muutujatena, et saaks edaspidi väärtuseid kasutada rakenduse toimimiseks ning suunatakse kasutaja edasi järgmisesse vaatesse kasutades Angular käsklust *$state.go*.

Kasutades globaalselt salvestatud ürituse andmeid oli vaja järgmise sammuna põhivaate kujundus kokku monteerida. Põhivaate kujunduse loomisel kasutati Bootstrap *Collapse* plugini *Accordion* näidet mis juba tühja näidisena toimimiseks kasutab omaette loogikat (vt. Lisa 4). Nüüd oli vaja sama loogika ja funktsionaalsus tööle saada Angular *ng-repeat* käsklusega ekraanile tekitatud korduvate elementide vahel ning igale elemendile külge panna vastav sisu (vt. Koodinäide 10). Selleks võetakse ükshaaval ette *menuItems.user.event.menu* objektiga seotud massiivid, täidetakse kood vastavalt massiivides olevate väärtustega ning kuvatakse ekraanil nii mitu *div* elementi ehk menüüvalikut mitu massiivi on *menuItems.user.event.menu* objektis. Menüüvalikule vastava sisu kuvamise seosed on paika pantud *back-end* poole peal seega toimub sisu kuvamine samuti kasutades massiivis olevat sisu muutujat (*items.content*). Kuna menüüvaliku sisu on HTML kood mis on salvestatud andmebaasi, siis tagastatakse see stringi kujul mida HTML kujul kuvamiseks on vaja muuta see tagasi õigesse vormingusse. See ei osutunud, aga kergeks ülesandeks. Autor proovis kasutada Angular SCE (Strict Contextual Escaping ehk ranged kontekstipõhised pääsud) mooduli funktsiooni nagu *$sce.trustAsHtml*, kuid muudetud vorminguga ei tulnud kaasa *scope* muutujate väärtused ning funktsionaalsus. Lõpuks õnnestus leida interneti avarustest kellegi poolt loodud Angular direktiivi *"compileHtml"* (vt. Lisa 1) mis tegi täpselt seda mida vaja oli.

```
<div class="panel-group" id="accordion" role="tablist" aria-
multiselectable="true" style="margin-bottom: 0;">
  <div class="panel panel-default" ng-repeat="items in
menuItems.user.event.menu | orderBy : 'order'">
    <div class="panel-heading collapsed" role="tab" id="menuTab-
{{items.id}}">
      <h4 class="panel-title">
        {{items.name}}
        <div class="abs-link" role="button" data-toggle="collapse" data-
parent="#accordion" data-target="#{{items.code}}" aria-expanded="false"
aria-controls="{{items.code}}" ng-click="mapLoading()">
        </div>
      </h4>
    </div>
    <div id="{{items.code}}" class="panel-collapse collapse"
role="tabpanel" aria-labelledby="menuTab-{{items.id}}">
      <div class="panel-body" style="text-align: center; border-top:
0px;">
        <div compile="items.content"></div>
      </div>
    </div>
  </div>
</div>
```

Koodinäide 10 "mainView" vaate korduvate menüüvalikute ja neile vastava sisu lisamine kasutades Angular ng-repeat käsklust

Kliendi soovil oli vaja lisada rakendusele ka serveripoolsete teavituste funktsionaalsuse. Selleks on olemas selline hea keskkond nagu Firebase mis pakub tasuta Firebase Cloud Messaging teenust. Selleks on vaja sisse logida nende keskkonda kasutades Google kontot, avada Firebase konsool, lisada konsooli alt uus projekt, lisada uus rakendus, valida platvorm (iOS, Android või Web), ära täita rakenduse info (kõige tähtsam osa on "Android package name" (vt. Joonis 13), rakenduse identifikaator mis määrati Cordova algrakenduse loomisel ning kätte saadav "config.xml" failist) mille järel pakutakse ning tuleb alla laadida fail "google-services.json". iOS puhul toimub sarnane protseduur, kuid lisaks faili ("GoogleService-Info.plist") allalaadimisele on vaja lisada arendusmeeskonna sertifikaadid ilma milleta ei hakka serveripoolsed teavitused kohale jõudma.

Joonis 13 Firebase uue rakenduse lisamisel vajalik väli

Järgmise sammuna tuli Cordova poole pealt installida plugin *cordova-plugin-fcm* ja *cordova-plugin-velda-devicefeedback* ning tõsta allalaaditud "google-services.json" fail Cordova

rakenduse *root* kausta. Peale seda tuli kasutada fcm plugina dokumentatsioonis olevaid koodinäiteid, et genereerida seadmespetsiifiline token (*pushToken*) mille kaudu on võimalik serveri poolt saata konkreetsele seadmele teavitusi. (Github, 2017) Teavituste saatmiseks tuleb *back-end* poole peal luua funktsioon millele lisada cURL (ehk vahend mida saab kasutada andmete saatmiseks serveri poole või serverist) serveripoolsete teavituste saatmise kood (hetke seisuga on õnnestunud ainult cURL kaudu teavituste saatmise tööle saada) (vt. Koodinäide 11).

```
$headers = array(
      'Authorization:key = firebase-keskkonnast-rakenduse-serverivõti',
      'Content-Type: application/json'
);
$fields = array(
      'registration_ids' => $this->tokens,
      'notification' => $this->message,
      'time_to_live' => 1200,
      'data' => $this->data,
      'priority' => 'high'
);

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, 'https://fcm.googleapis.com/fcm/send');
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));
$result = curl_exec($ch);
if ($result === FALSE) {
      die('Curl failed: ' . curl_error($ch));
}

return $result;
```

Koodinäide 11 Serveripoolsete teavituste saatmise cURL kood

Gurmeeteater mobiilirakenduse üheks eesmärkidest on külalise paberkandjal pileti asendamine. Seda võimaldab qrkoodi funktsionaalsuse lisamine rakendusele, selleks on autor kasutanud *angular-qrcode* kogu mis on võimalik NPM käsurealiidese abil alla laadida kasutades käsklust *npm install angular-qrcode*. Kogu kasutamiseks tuleb lisada *angular.module*'le juurde sõltuvus *monospaced.qrcode* (vt. Koodinäide 12) ning qrkoodi genereerimine toimub HTML vaates (vt. Koodinäide 13).

```
var app = angular.module('GurmeeApp', ['ui.router', 'monospaced.qrcode']);
```

Koodinäide 12 Angular.module'le qrkoodi kogu sõltuvuse lisamine

```
<qrcode version="2" data="{{menuItems.user.ticketCode}}" size="160">
</qrcode>
```

Koodinäide 13 QR koodi genereerimine

Viimaseks funktsionaalsuseks toob autor välja Google kaardiliidese lisamise ürituse asukoha kuvamiseks. Selleks kasutas autor jällegi Angulari kogu mille abil on võimalik rakendusele lisada Google kaardi funktsionaalsus. Seda on samuti võimalik allalaadida kasutades NPM käsurealiidest käsklusega *npm install angular-google-maps.* Kogu kasutamiseks, nagu qrkoodi kogu puhul, tuleb lisada *angular.module*'le juurde sõltuvus *uiGmapgoogle-maps* (vt. Koodinäide 14).

```
var app = angular.module('GurmeeApp', ['ui.router', 'uiGmapgoogle-maps',
'monospaced.qrcode']);
```

Koodinäide 14 Angular.module'le GoogleMaps kogu sõltuvuse lisamine

3.3.6 Probleemide kirjeldus ning nende lahendamine

Peamiseks probleemiks hübriidrakenduste loomisel on WebView kasutus ning sellest tulenevad iseärasused. Nimelt on turul meeletult palju erinevaid Android seadmeid ning arvestada kõikide seadmete iseärasustega ei ole võimalik seega tihtipeale võib rakendus ühel seadmel toimida ideaalselt, kuid teisel mitte.

Cordova raamistikul arendades on väga palju katse-eksituse meetodi kasutamist, kuna rakenduste arendamisel ning funktsionaalsuste lisamisel on väga tähtsal kohal pluginate kasutamine. Nendes ei saa aga alati kindel olla ning alles peale suuremat rakenduse testimist võib selguda, et plugin siiski ei toimi korralikult ning vaja leida mingi alternatiiv mis on kokkuvõttes suur ajaraiskamine. Näiteks Gurmeeteater rakenduse arendamise käigus lisas arendaja, kes tegeles administreerimise rakenduse arendamisega, rakendusele QR koodilugemise plugina millel oli hea tagasiside ning suur kasutajaskond. Peale testimist ei olnud, aga klient nõus sellist funktsionaalsust kasutama kuna kindlates olukordades oli QR koodilugeja liiga aeglane, lisaks sellele polnud võimalik lihtsasti muuta QR koodi lugeja vaadet mis avanes Android aktiivsusena. Sama pluginat kasutas ka käesoleva töö autor Gurmeeteater rakenduses qrkoodi genereerimiseks, kuid jällegi avanes genereeritud qrkood Android aktiivsuse vaates, kus oli üleliigseid elemente mis ei sobinud rakenduse kujundusega, seega otsustas autor võtta kasutusele *angular-qrcode* kogu. Autori kogemus näitab, et võimalikult palju funktsionaalsuseid võiks üritada lahendada ilma, et kasutada pluginaid, see

tagab stabiilsema funktsioneerimise eri seadmetel. iOS puhul seda probleemi ei esine nii palju kuna Apple'l on ainult enda seadmed ning süsteemid erinevad vaid mõne versiooni võrra mis tagab stabiilsema tulemuse.

iOS'iga seoses tekkisid probleemid serveripoolsete teavituste kättesaamisega ning App poodi rakenduse üleslaadimisega. Apple'l on väga ranged reeglid seoses nende platvormile arendamisega. Esiteks peab sul olema OS X operatsioonisüsteemiga seade, et saaksid üldse iOS'ile rakendusi luua, teiseks pead looma või olema mõne Apple arendusmeeskonna liige muidu ei ole võimalik rakendust allkirjastada. Nagu eelnevates peatükkides mainitud sai tuleb serveripoolsete teavituste saatmiseks lisada Firebase keskkonda Apple arendaja sertifikaadid selleks, et teenus iOS'i peal tööle hakkaks. App poodi üleslaadimisel tuleb tähelepanelikult läbi lugeda tingimused või nõudmised rakendustele. Sellega seoses lükati Gurmeeteater rakendus App poe poolt tagasi. Nimelt põhjuseks oli, et koodi alusel sisselogimine või sisu kuvamine on App poe tingimuste vastane. Sisselogimine peab olema standartsel kasutajanimi (e-mail) ja parool kujul. Seetõttu oli töö autor sunnitud Gurmeeteater rakenduse sisselogimise funktsionaalsuse spetsiaalselt iOS'i jaoks ümber tegema.

Suureks probleemiks Cordova raamistikul arendamisel on ka see, et rakenduse ehitamine võtab tüütult palju aega ning arendamise käigus toimub seda rakenduse ehitamist väga palju. Seoses sellega on näiteks PhoneGap ja Ionic raamistikel *serve* funktsionaalsus, kuid see on mõeldud ainult läbi veebibrauseri arendamisel mille kaudu ei ole võimalik kõiki rakenduse funktsionaalsuseid testida.

3.3.7 Valminud rakendus

Valminud rakendus koosneb kahest põhivaatest milleks on sisselogimise "loginView" (vt. Lisa 7) vaade ning põhivaade "mainView" (vt. Lisa 6). Sisselogimine toimub piletikoodi alusel ning vastuseks tagastatakse andmebaasist piletiomaniku info, ürituse info, menüüvalikud, menüüvalikute sisu, taustapilt ning üritуselogo. Menüüvalikute avamine toimib akkordeoni meetodil (vajutades menüüvalikule lohistatakse sisu lahti). Gurmeeteater mobiilirakenduse menüüvalikud:

- Ürituse info, kus on lavastuse tutvustus ning lavastusega seotud inimeste nimed.
- Asukoht, kus on Google kaardi liides mis juhatab ürituse toimumiskohale (vt. Lisa 9).

- Piletiinfo, kus on qrkood mis asendab paberkandjal piletit ning konkreetse piletiga seotud info (vt. Lisa 8).

- Toidumenüü, kus on toidumenüü ning joogikaart.

- Tagasiside, tagasiside vorm (vt. Lisa 10).

Rakenduse alumises osas on nupp millega saab teenindaja lauda kutsuda ning üleval paremas nurgas on peidetud menüü mille kaudu on võimalik ürituselt välja logida.

3.4 Andmebaasi struktuur

Andmebaas on loodud kasutades Laraveli migratsiooni faile ning phpMyAdmin keskkonda. Tabeleid on andmebaasis kokku 16 ning baasiga suhtlemiseks ja päringute tegemiseks on loodud Laraveli raamistikul back-end mudelid ja kontrollerid. Tabeli struktuuri ning seostega tutvumiseks on töö autor lisanud lisadesse andmebaasi diagrammi milles osad tabelid on konfidentsaalsuspoliitika tõttu ära varjatud (vt. Lisa 11).

3.5 Testimine reaalsetes tingimustes (Gurmeeteater eelproov)

Esimene Gurmeeteater mobiilirakenduse tõsisem testimine toimus Gurmeeteater ürituse eelproovi ajal kus paluti kohal viibida ka Gurmeeteater mobiilirakenduse arendajatel. Testimine koosnes kolmes etapis. Esimene oli välis ukse juures piletikontroll, kus testiti administreerimise rakenduse qrkoodi lugejat ning tutvustati Gurmeeteater rakenduse kasutamist külalistele kes ei olnud jõudnud endale rakendust veel seadmesse installeerida. Teine etapp oli valvelauas uuesti qrkoodi lugemise testimine (teisel korral tagastatakse külalise laua andmed, et saaks lauda suunata) ning Gurmeeteater mobiilirakenduse poole pealt vastavalt teisest qrkoodi lugemisest avaneva "kutsu teenindaja lauda" nupu funktsionaalsuse toimimine ning serveri poolt saadetud teavituste kohale jõudmine. Kolmas etapp oli ürituse ajal peidetud menüüvalikute aktiveerimise ning "kutsu teenindaja lauda" nupu kasutuse testimine. Ürituse lõpus tagasiside vormi täitmise testimine.

Gurmeeteater mobiilirakenduse arendajatel olid kaasas ka sülearvutid, et saaks kohe tekkinud probleemidele jälile. Üks probleem ilmnes suhteliselt kohe ning see oli seotud sisselogimisega kus kasutajatel ei olnud võimalik sisse logida. Probleem oli seotud sellega, et kliendi esialgne soov oli mitte võimaldada samaaegselt ühe ja sama koodiga sisselogimise võimalust. Juhtus aga see, et mingites olukordades ei kustunud välja logides andmebaasist kasutajat

identifitseerivad tokenid mis jättis mulje, et kasutaja on sisse logitud ning ei lasknud enam uuesti sama piletikoodiga sisse logida. Back-end arendaja muutis siiski võttis ära piirangu ning sisselogimine hakkas toimima nii, et kirjutatakse token üle kui token on andmebaasis veel alles. Rohkem eelproovi ajal probleeme ei esinenud, kuid hiljem selgus, et olid probleemid Gurmeeteater veebilehe eksport funktsioonis mis tagastas valesi andmeid. Kuna Gurmeeteater rakenduse back-end arendaja lisas impordi funksioonile külge kontrollid mis ei lubanud läbi lasta valede andmetega sisu, siis hiljem juhtus olukord kus import ei õnnestunud ning kasutajatel polnud võimalik piletikoodidega sisse logida, kuna neid piletikoode lihtsalt polnud andmebaasis olemas. Parandused viidi sisse Gurmeeteater veebilehe poole pealt ning olukord lahenes.

Kokkuvõte

Bakalaureusetöö eesmärgiks oli kirjeldada Gurmeeteater mobiilirakenduse arendusprotsessi alustades kliendi vajaduste välja selgitamisest kuni rakenduse valmimise- ning reaalsetes tingimustes testimiseni.

Töö eesmärgi saavutamiseks jaotas autor töö kolme osasse. Esimeses osas toimus kliendiga suhtlemine, vajaduste ning idee väljaselgitamine ja soovitud funktsionaalsuste kooskõlastamine, kus selgus kliendi täpne soov ning ajaline piirang projekti valmimiseni. Töö teises osas sai paika pandud projektitöö korraldus kuhu alla kuulus arendajate vaheline suhtlus ja arenduse kooskõlastamine, vähestest kogemustest tingitud arendajate vaheline segadus ning projekti raames kliendiga suhtlemine. Viimases osas tegeles autor Gurmeeteater mobiilirakenduse arenduse käigus läbi käidud etappide kirjeldamisega, kus tõi välja projekti raames kasutatud arendusraamistikud ning tehnoloogiad, valminud disaini kooskõlastamise kliendiga, rakenduse arenduse sammud alates keskkonna eelseadistamisest, struktuuri loomisest, kujunduse külgepanekust kuni valmis rakenduse testimiseni.

Rakendust sai testida ka reaalsetes tingimustes, kui toimus Gurmeeteatri eelproov. Eelproovist tulid esile väiksed vead mis said kohe kohapeal ka korda tehtud. Suuremad vead tulid välja kahjuks hiljem, mis olid seotud Gurmeeteatri veebilehe andmete eksportimise funktsioonist, kuid need vead sai samuti üsna kiiresti lahendatud.

Arenduse käigus tulid välja rakenduse funktsionaalsuse loogikas üksikud probleemid. Põhiline oli seotud sisse logimise funktsionaalsusega mis sai tehtud piletikoodi alusel, kuid selgus, et Apple App poe tingimuste kohaselt peab sisse logimine toimuma kasutaja ja parooli alusel ning nad ei luba koodi alusel sisu kuvamist. Seda aspekti saakski edasiarendamise mõttes aluseks võtta ning muuta rakendus kliendipõhiseks mitte üritusepõhiseks nagu praegu.

Bakalaureusetöö autor sai väärtuslikud õppetunnid Gurmeeteater mobiilirakenduse arenduse käigus nii arendamises kui projektijuhtimises. Käesolev töö ning selle tulemused võivad huvi pakkuda inimestele, kes ei oma kogemust mobiilirakenduse arendamises ning sooviks tutvuda üldise protsessikirjeldusega, kui ka nendele kes soovivad suunata enda teadmisi kas hübriid- või kohalikumobiilirakenduste arendamise suunas.

Kasutatud kirjandus

Ziflaj, A. (2014). Native vs Hybrid App Development. Loetud 29.04.2017 aadressil: https://www.sitepoint.com/native-vs-hybrid-app-development/

Cordova. (kuupäev puudub). Cordova Tools. Loetud 29.04.2017 aadressil: http://cordova.apache.org/

Camden, K. R. (2016). Apache Cordova in Action. New York, USA. Manning Publications Co.

O'Brien, J. (2016). A brief History of Laravel. Loetud 29.04.2017 aadressil: https://medium.com/vehikl-news/a-brief-history-of-laravel-5d55970885bc

Google Trends. (kuupäev puudub). Google Trends Compare. Loetud 29.04.2017 aadressil: https://trends.google.com/trends/explore?cat=5&q=Phalcon,CodeIgniter,Zend,Cakephp,Laravel&hl=en-US

TutorialsTeacher.com. (kuupäev puudub). What is AngularJS? Loetud 29.04.2017 aadressil: http://www.tutorialsteacher.com/angularjs/what-is-angularjs

Tutorialspoint. (kuupäev puudub). Node.js - Introduction. Loetud 30.04.2017 aadressil: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Firebase. (2017). Firebase Cloud Messaging. Loetud 30.04.2017 aadressil: https://firebase.google.com/docs/cloud-messaging/

Tutorial Republic. (kuupäev puudub). Bootstrap Introduction. Loetud 30.04.2017 aadressil: http://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-introduction.php

Github. (2017). Google Firebase Cloud Messaging Cordova Push Plugin. Loetud 02.05.2017 aadressil: https://github.com/fechanique/cordova-plugin-fcm

Summary

Title: Gourmet Theatre Mobile Application Development

The aim of this Bachelor Thesis was to describe the process of developing Gourmet Theatre mobile application starting from understanding the client needs until completion of application and testing in real life scenario.

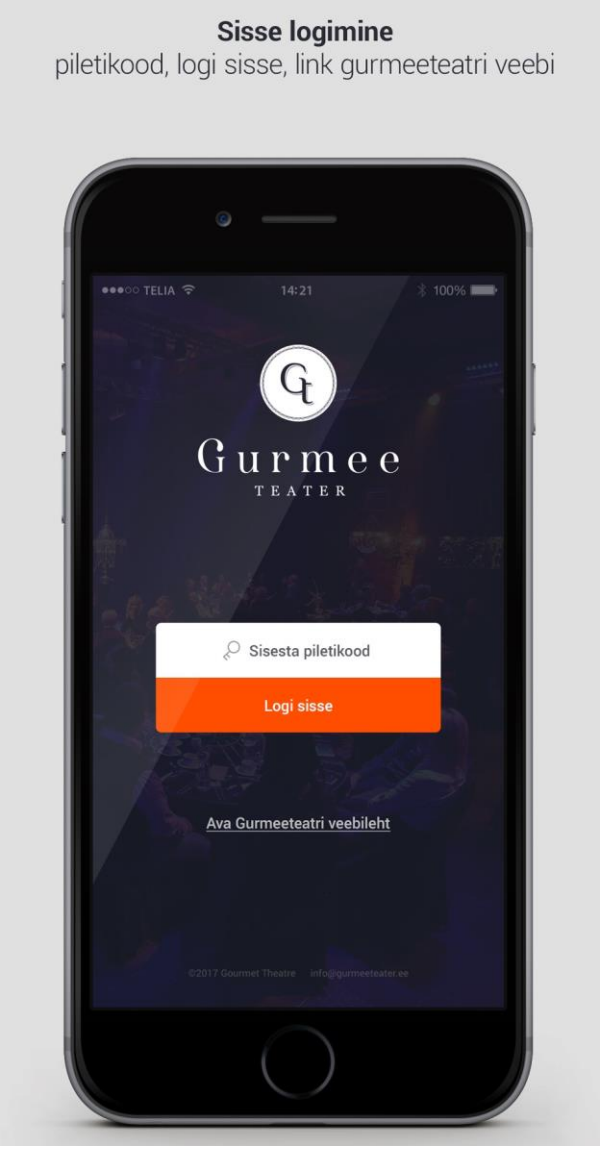
To achieve this aim author devided thesis into three chapters. First chapter describes the process of first communication with the client to understand the needs of a client, to write down application functionality needs and find out the project due date. In second chapter author establishes project management principles like communication between developers, describes emerged problems and steps taken to overcome them plus communication with client throughout the project. In the last chapter author concentrates on the coding stages of application development he brings up different frameworks and technologies that are used during the development stages, depicts the approval of the first application design and step-by-step description of the process of building the application from framework preparation to testing of the finished product.

The real life testing of Gourmet Theatre mobile application was done during the rehersal play where minor problems arose and got fixed on the spot. Bigger problems arose later during live plays because of the Gourmet Theatre website side import function but got fixed aswell very quickly.

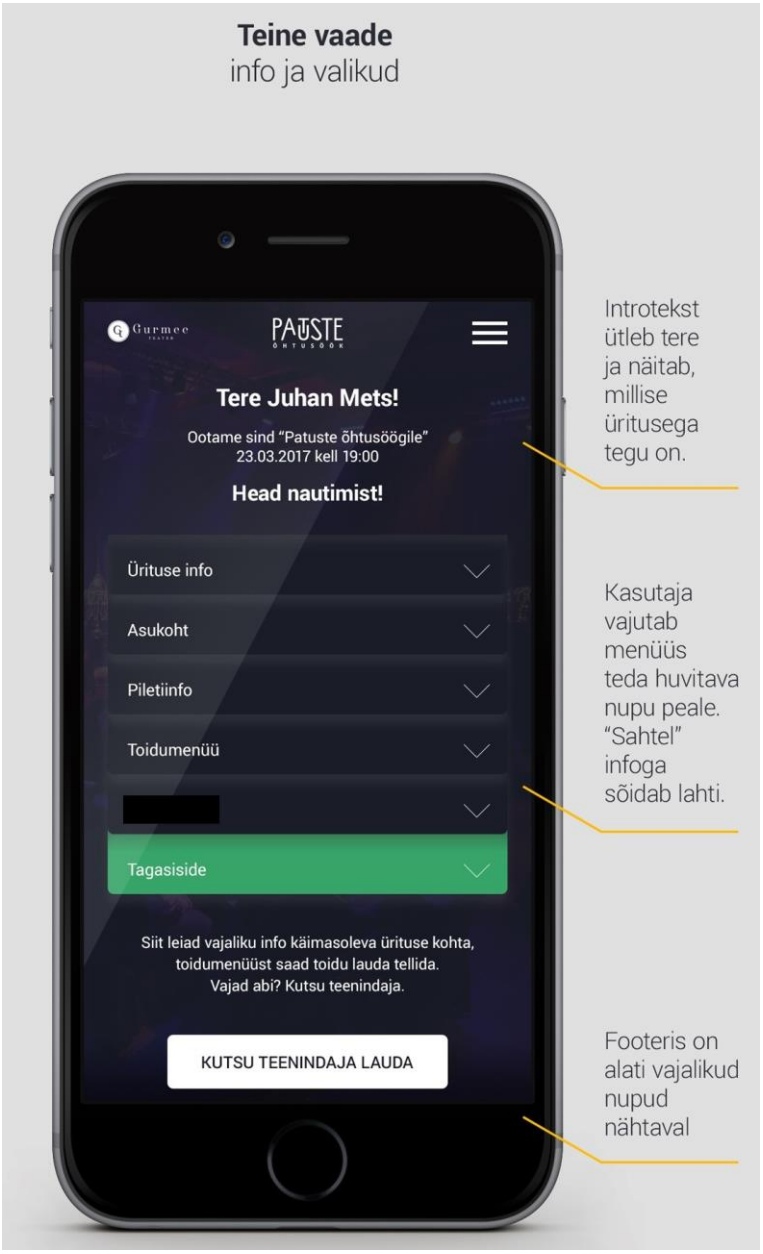
During the development of the application there were problems with the logics of the main functionality. Specifically problem with the login function of the application. Login was performed through ticketcode to show the content of the event but Apple App store does not permit the login through promotional codes. So because of this aspect the development of the application will continue with a direction to change the logic of the application to client centered.

Thesis author has aquired a lot of valuable information and knowledge during the process of developing Gourmet Theatre application and belives that the present thesis would interest readers who do not have experience in mobile application development or are searching for some guidance to direct them in the right direction.

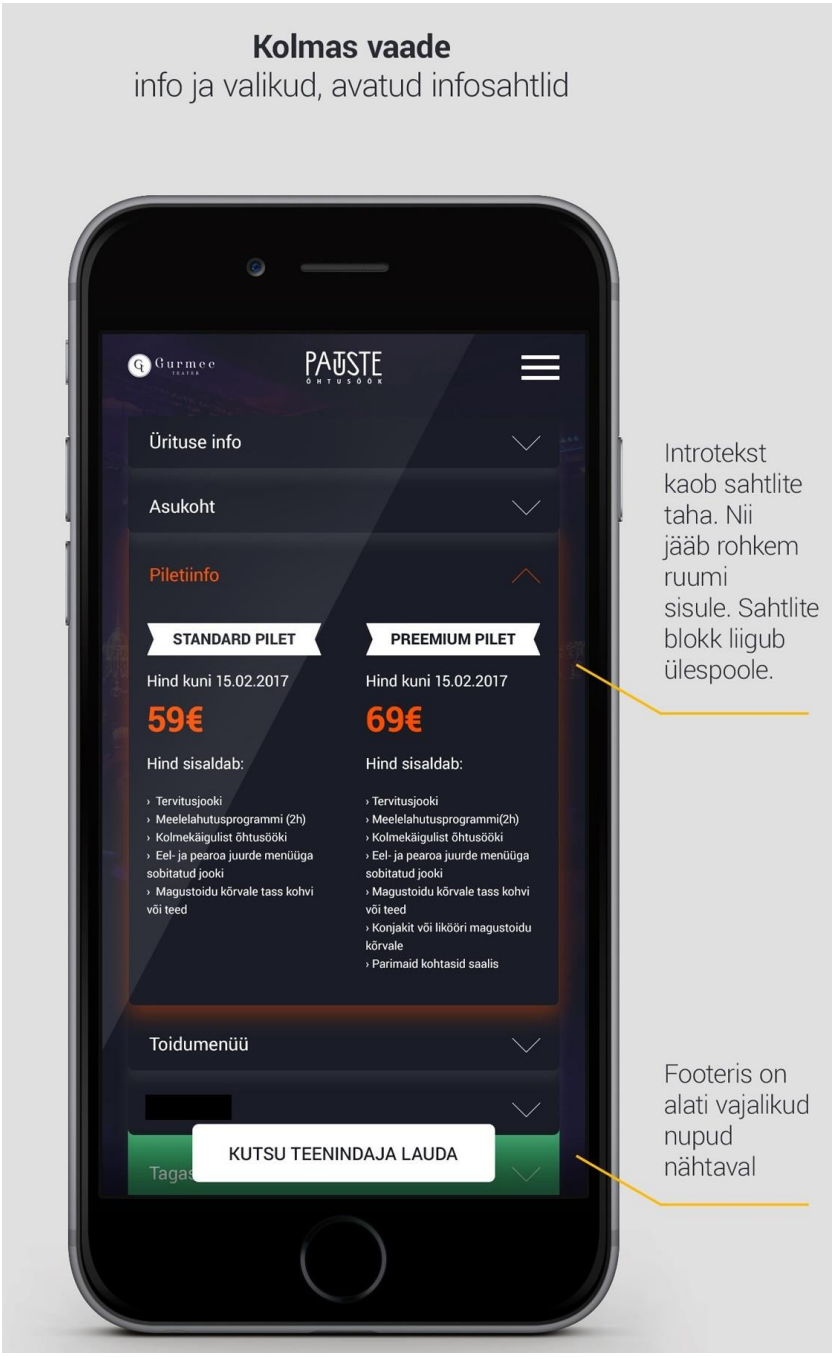
Lisad



Lisa 1 Disaini kooskõlastamine kliendiga, sisselogimise vaade



Lisa 2 Disaini kooskõlastamine kliendiga, teine vaade



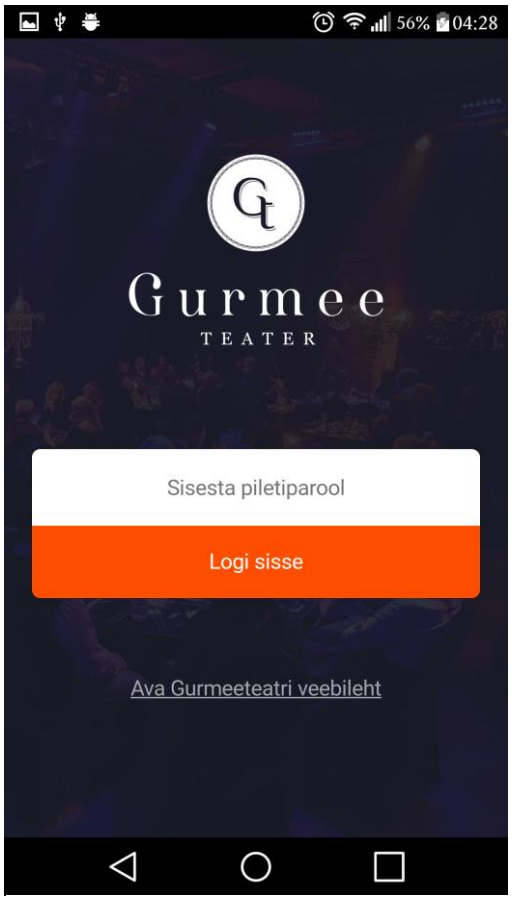
Lisa 3 Disaini kooskõlastamine kliendiga, kolmas vaade (avatud sahtliblokiga)

```
<div class="panel-group" id="accordion" role="tablist" aria-multiselectable="true">
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="headingOne">
      <h4 class="panel-title">
         <a role="button" data-toggle="collapse" data-parent="#accordion"
href="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
          Collapsible Item #1
        </a>
      </h4>
    </div>
    <div id="collapseOne" class="panel-collapse collapse in" role="tabpanel" aria-
labelledby="headingOne">
      <div class="panel-body">
        Body Text
      </div>
    </div>
  </div>
  <div class="panel panel-default">
    <div class="panel-heading" role="tab" id="headingTwo">
      <h4 class="panel-title">
        <a class="collapsed" role="button" data-toggle="collapse" data-parent="#accordion"
href="#collapseTwo" aria-expanded="false" aria-controls="collapseTwo">
          Collapsible Group Item #2
        </a>
      </h4>
    </div>
    <div id="collapseTwo" class="panel-collapse collapse" role="tabpanel" aria-
labelledby="headingTwo">
      <div class="panel-body">
        Body Text 2
      </div>
    </div>
  </div>
</div>
```

Lisa 4 Bootstrap Collapse Accordion koodinäide

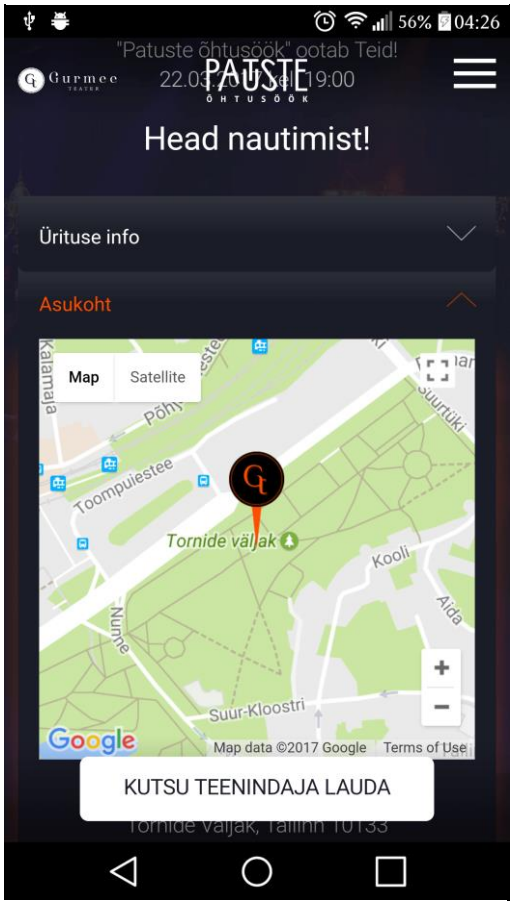
```
"use strict";
app.directive('compile', ['$compile', function ($compile) {
    return function (scope, element, attrs) {
        scope.$watch(
            function (scope) {
                // watch the 'compile' expression for changes
                return scope.$eval(attrs.compile);
            },
            function (value) {
                // when the 'compile' expression changes
                // assign it into the current DOM
                element.html(value);
                // compile the new DOM and link it to the current
                // scope.
                // NOTE: we only compile .childNodes so that
                // we don't get into infinite loop compiling ourselves
                $compile(element.contents())(scope);
            }
        );
    };
}]);
```

Lisa 5 Angular direktiiv mille abil saab Html stringi muuta Html vormingusse



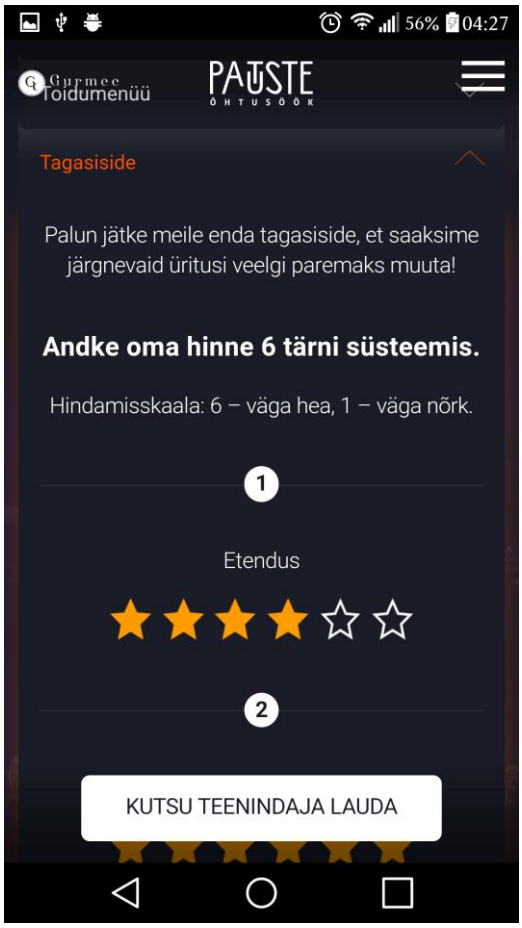
Lisa 7 Valmisrakenduse "loginView" vaade

Lisa 6 Valmisrakenduse "mainView" vaade

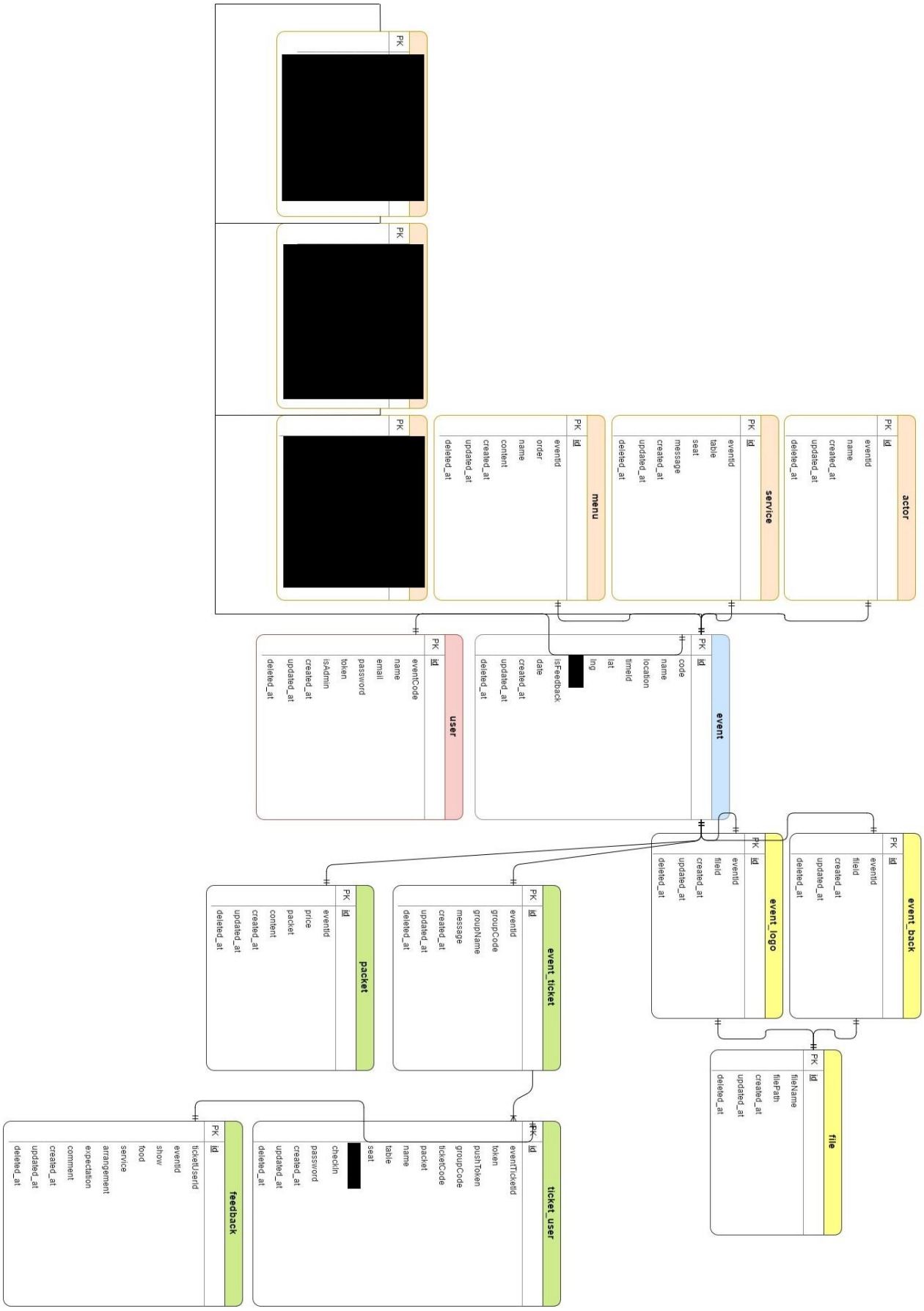


Lisa 9 Valmisrakenduse Asukoht sisu vaade

Lisa 8 Valmisrakenduse Piletiinfo sisu vaade



Lisa 10 Valmisrakenduse Tagasiside vormi vaade



Lisa 11 Andmebaasi struktuuri diagramm