Tallinna Ülikool
Informaatika Instituut

PHONEGAP RAKENDUS RALLY ESTONIA PILETIMÜÜGI NÄITEL

Bakalaureusetöö

Autor: Toomas Naaber

Juhendaja: Jaagup Kippar

Autor: ..…….”…….” ….……2015

Juhendaja:……..…..“…….” …….2015

Instituudi direktor:……..…..“…….” …….2015

Tallinn 2015

SISUKORD

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, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....

(autor)

LIHTLITSENTS LÕPUTÖÖ REPRODUTSEERIMISEKS JA LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS

Mina Toomas Naaber (sünnikuupäev: 14.08.1991) annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose PhoneGap rakendus Rally Estonia piletimüügi näitel, mille juhendaja on Jaagup Kippar, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

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

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

Tallinnas, *digitaalselt allkirjastatud*

Sissejuhatus

Bakalaureusetöö teemaks valis autor Rally Estonia piletimüügi rakenduse prototüübi loomise, mis aitaks piletimüügi koordineerimist paremini ning vähemate probleemidega korraldada. Töö lõppeesmärgiks on saada valmis esmane töötav prototüüp, mis toimib Android platvormil.

Töö valikul lähtus autor eelmise aasta kogemusest, olles ise piletimüügi juures ja nähes, kuidas vahepeal tekkisid mured ning probleemid. Mõnede probleemide lahendus viibis teatud aja, sest koordinaatoreid oli kaks ning pileti müügipunkte rohkem kui 20.

Töö algul toob autor välja põhilised võõrkeelsed mõisted, mida töö käigus rohkem kasutatakse. Teises peatükis võtab autor süvitsi ette rakenduse kavandamise etapid, kus käsitletakse valmiva prototüübi eelinformatsiooni, mille baasil saab hakata looma rakendust. Samuti tuuakse välja funktsionaalsed nõuded rakendusele, lühike tutvustus andmebaasile ning rakenduse kujundamise protsess.

Kolmandas peatükis tuuakse välja rakenduse arenduse protsessid, mis järgnevad PhoneGap rakenduse põhja seadistusele, mis said tehtud eeltööna seminaritöös. Tuuakse välja eraldi ka Angular scriptid ning templiidid ja Parse puhul andmebaasi näidispäringud ning Push funktsiooni seadistus.

Eesmärgi saavutamiseks viib autor läbi vestluse eelmise aasta koordinaatoritega, et selgitada välja funktsionaalsed nõuded. Seejärel uurib võimalike lahendusi ning võtab kasutusele MySQL’st erineva andmebaasi võimaluse ja kasutab veebirakenduse loomiseks mugavat veebiraamistikku Angulari. ,

Rakendusega tutvumiseks võib võtta ühendust e-posti teel rakenduse autoriga
e-post: toomas91@tlu.ee

Mõistete loetelu

JavaScript – JavaScript on kliendipoolne programmeerimiskeel, mis tähendab, et lähtekoodi töötleb kliendi veebilehitseja mitte veebiserver (TechTerms, 2015).

API ehk Application Program Interface – on kogum käskudest, funktsioone ja protokolle, mis programmeerijad saavad kasutada luues tarkvara konkreetsele operatsioonisüsteemile (TechTerms, 2015).

HTML ehk Hyper-Text Markup Language – programmeerimiskeel, millega kirjutatake veebilehti (TechTerms, 2015).

CSS ehk Cascading Style Sheet – kasutatakse veebilehe kujunduse vormistamiseks. Seda kasutades saab määrata näiteks teksti stiili, tabeli suurusi ja palju muid aspekte, mida varasemalt sai määrata ainult HTML lehel (TechTerms, 2015).

JSON ehk JavaScript Object Notation – tekstipõhine andmevahetusvorming, et edastada struktureeritud andmeid (TechTerms, 2015).

RGB ehk Red Green Blue – kolm põhitooni, mille kokku miksimisel saab moodustada mis tahes värvikombinatsiooni (TechTerms, 2015).

SDK ehk Software Development Kit – kasutatava tarkvara kogumik arendajatele loomaks rakendusi konkreetsetele seadmetele või operatsioonisüsteemidele (TechTerms, 2015).

GPS ehk Global Positioning System - on satelliitnavigatsiooni süsteem, mida kasutatakse määramaks maapinnal oleva objekti asukoht (TermTechs, 2015).

1 Rakenduse kavandamine

Rakenduse kavandamisel vaadeldakse funktsionaalseid nõudeid, tehakse ülevaade andmebaasi tabelitest ning kirjeldatakse lahti kujundusprotsessi etapid.

1.1 Funktsionaalsed nõuded rakendusele

Rakenduse loomiseks sai autor kokku Rally Estonia piletimüügi koordinaatoriga ning arutamise käigus leiti ühiselt neli erinevat funktsionaalset nõuet piletimüüjale ning üks lisafunktsionaalsus administraatori vaatesse. Kõik funktsionaalsused kategoriseeriti kolme gruppi.

1.1.1 Müüja piletite sisestamine

Arutelus selgus, et kõige olulisem funktsioon rakenduse juures on müügitulemuste sisestamine. Iga müügipunkt saab enda valdusesse teatud arvu rallipileteid, päevapasse ning rallipasse. Hulk, mis igasse punkti läheb, on erinev, sest teatud kohtades on vaatajate huvi tunduvalt suurem. Seega, nendes punktides peab olema piletite kogus suurem.

Praeguse info kohaselt ei ole piletitel mingeid eritunnuseid, millega peaks pileteid eristama. Piletite müüjad saavad enda kätte piletid, edasi koordinaator määrab ära hulga, mis vastavasse punkti sai antud. Sedasi on hea jälgida kui palju pileteid antud punktis järgi on peale müümisi.

Piletite sisestamisel on kindel eesmärk – müügi koordinaatoritel on selge ülevaade piletite müügi protsessist ning sellest, kuidas teatud punktides antud protsess on toimunud.

1.1.2 Müüja positsioneerimine

Positsioneerimine tuli nõuete analüüsis välja viimase ideena. Selleks, et müüjad jõuaksid õigesse punkti ilma suuremate takistusteta leiti, et antud võimalus annaks müüjale rakenduse toel kätte õige suuna müügipunktini. Kuna müügipunktide asukohad on enamjaolt külavaheteedel ning mis on kitsad, siis bussiga ei pääse igale poole ligi ja manööverdamine sealsetes teeoludes on väga keeruline. Seepärast viiakse müüjad teatud ristmikuteni, kust edasi tuleb neil minna jalgsi oma punktini.

Mõned müüjad peavad jalgsi minema rohkem kui kilomeeter ning teepeal olevad ristmikud võivad tekitada segadust ja müüja võib sattuda valesse kohta pileteid müüma. Hilisema avastuse

käigus võib sealt punktist läbi läinud olla suur hulk rahvast ning sealt tulenev piletite tulu jääb saamata. Kõige rohkem kahju tooksid need punktid, mis on vaatajate seas populaarseimad.

Piletimüügi koordinaatorite rakendusele sooviti võimalust vaadelda kõikide punktide asukohti, et olla kindel nende positsiooni õigsuses. Peale selle peaks olema võimalik vaadelda ka punktide müügiprotsessi kulgu. Viimast koguste kuvamiseks valitud punktile vajutades.

1.1.3 Müüja ning koordinaatori teadete saatmine

Teadete saatmise olulisus tuleneb vajadusest müüjate ja koordinaatorite vahelisest suhtlemisest. Koordinaatoreid oli 2014 aastal kaks tükki ja müügipunkte rohkem kui 20. Esines juhtumeid kus telefoni liin müügipunkti ja koordinaatori vahel oli hõivatud ning probleemi lahendus viibis. Rakendus annaks võimaluse müüjal saata teade rakenduse andmebaasi, mis jõuab koordinaatori rakendusse nähtavaks. Koordinaatoril on võimalus antud teatele vastata, kas läbi telefonikõne või anda vastu kirjalik teade, lähtudes vajadusest.

1.2 Rakenduse andmebaasi kirjeldus

Andmebaasi valikuna võtab töö autor kasutusele ParseJS, millest sisu küsimiseks ning lisamiseks kasutatakse JavaScrpti funktsioone. Rakenduse andmebaasi luuakse kuus erinevat andmetabelit. Käesolevas peatükis tehakse lühike tutvustus põhilistest tabeliridadest, mis omavad suuremat vajalikkust, kuid kõiki tabeli kirjeid ei tooda välja rakenduse turvalisuse kaalutluste pärast.

Esimene andmetabel on kasutajate haldamiseks, kus hoitakse kasutajanime, parooli, positsiooni rakenduse kasutamisel ning rallipunkti.

Järgmiseks tabeliks on piletite müügi tabel, kuhu lisatakse piletimüügist tulenevad müüginumbrid. Selles tabelis hoitakse järgmised andmed – kasutaja ID, pileti/passi ID tuvastamaks, millise sisestusega tegu on. Lisaks on rallipunkti ID ning viimaseks tabeli tulbaks on kustutatud piletid.

Kolmandaks tabeliks loodi positsioonide tabel, kuhu saabuvad iga punkti asukohad koheselt peale positsioneerimist ning piletimüüjate poolt lisatud kindlad asukohad, kus vastavad piletimüüjad olema peavad. Tabelitesse määrab koordinaator punkti ID ning vajalikud koordinaadid. Piletimüüja positsioneerimise käigus lisatakse andmebaasi piletimüüja tegelikud asukoha koordinaadid.

Neljandaks tabeliks loodi kommentaaride ja teadete tabel. Tabelisse saabuvad kõik müügipunktide teated kui ka koordinaatorite teated. Tabeli kirjetena on kommentaari lahter, kuhu lisatakse kommentaar. Lisaks on veel tähtis teate lahter, millega määratakse, kas teade on tähtis ning kas seda peab nägema avakuva vaates.

Viiendaks tabeliks loodi logimise andmetabel, kuhu jäädvustatakse kasutajate sisselogimise ajad. Tabel on oluline, et näha millal viimati müüja rakendust kasutas. Kui teatud aja jooksul ei ole rakendust kasutatud, siis on võimalik võtta ühendust punktiga ning uurida, kas midagi on juhtunud.

Viimane tabel on seadme installatsiooni andmete hoidmiseks. Tabelis hoitakse seadme käest installatsiooni käigus saadud seadme infot. Seadmele antakse kindel ID, mille järgi on võimalik seda tuvastada. Vastavasse tabelisse lisatakse veel rallipunkt ning seadme operatsioonisüsteem.

1.3 Rakenduse kujundamine

Rakenduse prototüübi esialgsele kujundamisele suurt rõhku ei pööratud. Kujundusprotsessis alustati algusest ning käesolevas peatükis kirjeldatakse kogu protsessi värvide valikust kuni rakenduse endani.

1.3.1 Rakenduse esmane värvivalik

Rakendusele tuli leida algsed värvide kombinatsioonid, mis sobivad antud imagoga kokku. Värvi valikuid oli mitmeid, kuid lõpuks sai valitud värvideks toonid RGB värvikoodis #699E99, RGB #00352F ning RGB #000E0D.

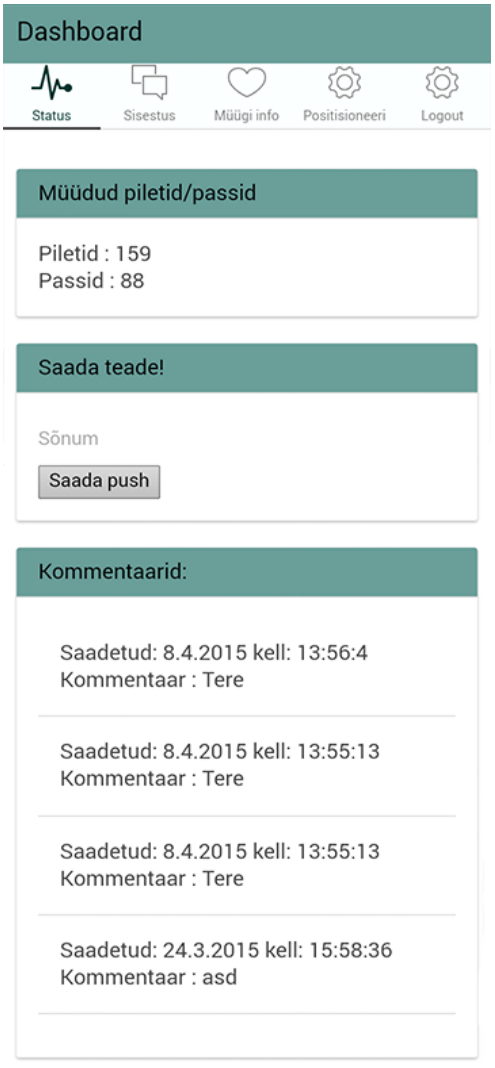
1.3.2 Login ja avakuva

Arutelu käigus tuli välja, et sisse logimine peab olema lihtne ja arusaadav (Joonis 3). Kujunduses lähtuti eelmainitust ning lisatud sai rakenduse logimisel kaks tekstikasti (input text, input password) ja login nupp. Vea teate puhul kuvatakse pop-up teade. Ekraanil asub login vorm sisuala rakenduse keskel.

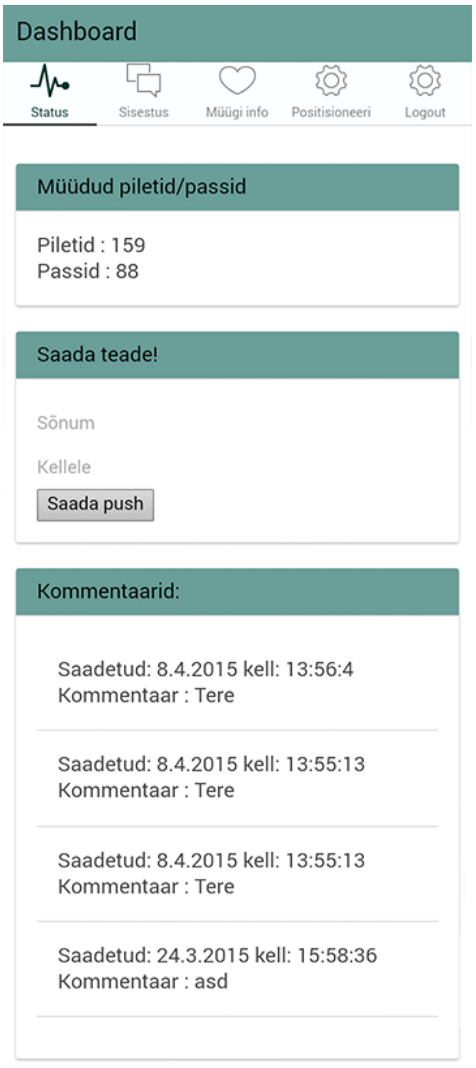
Avakuva kujundamise käigus tuli luua vaateid rohkem kui üks. Lisaks põhivaatele leiti lahendus ka navigaatorile. Avakuvale tehti kaks erinevat lahendust, üks piletimüüjale ja teine koordinaatorile.

Piletimüüja avakuvasse sooviti paigutada kolm tulemit või vaadet. Esiteks pidi iga piletimüüja nägema tema poolt müüdud piletite hulka, et veenduda müüdud piletite kogustes. Teiseks peab müüja saama avakuvast saata koordinaatoritele probleemist teate, juhul kui telefoni kaudu pole koordinaator kättesaadav. Kolmandaks tuli soovina koordinaatoritelt, et avakuvas peab olema nende poolt saadetud teated nähtavad (Joonis 2).

Avakuva koordinaatorite rakenduses tuleb mõningal määral erinev (Joonis 1). Avakuvas ühesuguseks jääv element on sõnumite saatmine, kuid väikse erinevusega. Rakenduses saab koordinaator valida, kas teade saadetakse konkreetsesse punkti või kõikidesse punktidesse korraga.



Joonis 1

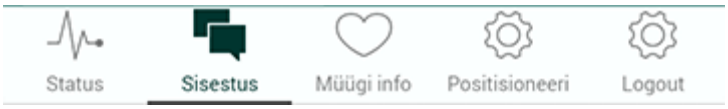


Joonis 2

Joonis 3

1.3.3 Navigaator ehk menüü

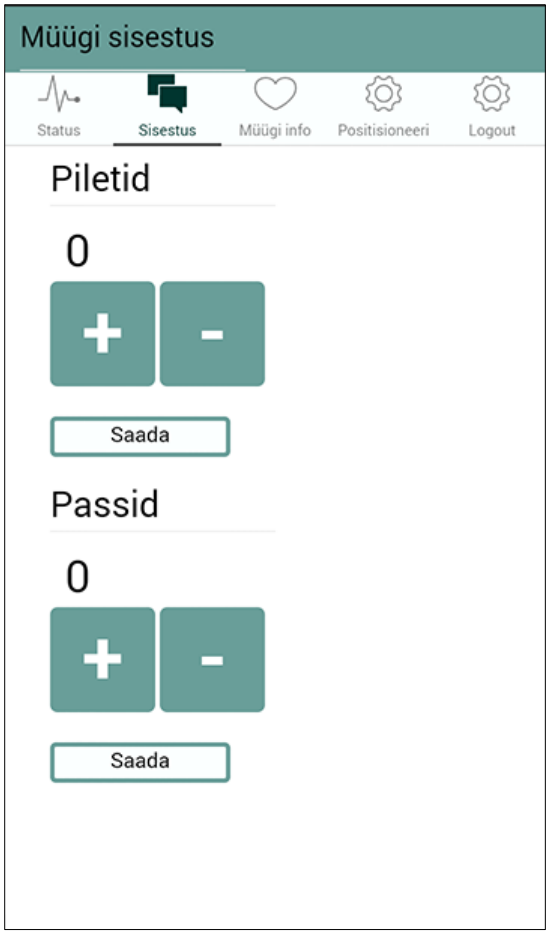
Menüüriba kujunduses sai paika pandud, nii et see paigutuks rakenduse päisesse ning fikseeritakse (Joonis 4). Menüü reale tuleb viis erinevat linkimisnuppu – avakuva, piletite sisestus, müügiajalugu, positsioneerimine ja logi välja. Koordinaatoritel tuleb piletite sisestuse asemel teadete ajalugu.



Joonis 4

1.3.4 Piletite sisestamine

Piletite sisestuse vaatesse (Joonis 5) tuleb kaks erinevat välja ning nupud. Nupud lähevad tekstiväljade alla. Väljade sisse kirjutatakse või lisatakse nuppude abil piletite/passide kogused, mitu piletit müüdi. Piletite müügitulemus kajastub koheselt ka müügiajaloo vaates.

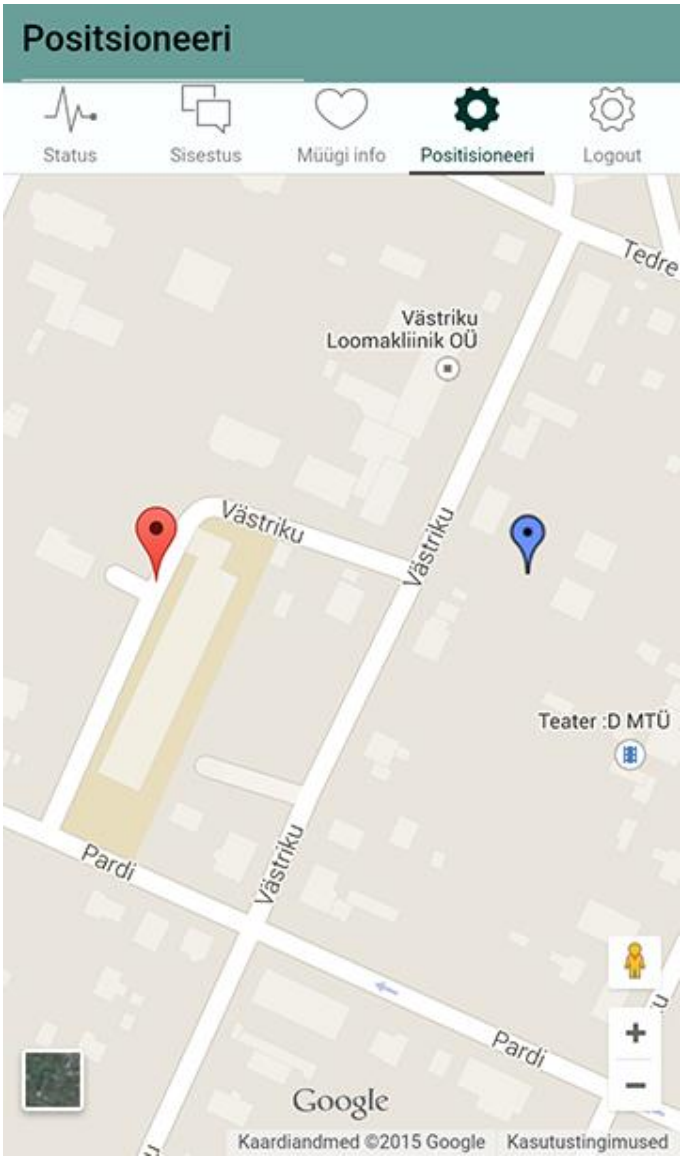


Joonis 5

1.3.5 Positsioneerimine

Positsioneerimise vaates (Joonis 6) kuvatakse müüjale Google Maps kaart, kus on määratud kaks punkti. Üheks punktiks on koht, kuhu peab liikuma piletimüüja ning teiseks on punkt, kus müüja asub. Punktid määratakse erinevate värvi ikoonidega, kus sinine on sihtkoht ning punane müüja asukoht.

Koordinaatori kaardi vaatesse tulevad kõikide punktide asukohad, kus müüjad hetkel on ning kus nad olema peaks. Eristuseks kasutatakse juhuslikke värvikoode persooni tuvastamiseks ning sinist ikooni sihtkohana. Lisaks tuleb igale punktile juurde lisada ka piletimüügi tulemused, mis tulevad nähtavaks alles müüja asukohale vajutades.



Joonis 6

1.3.6 Müügiajalugu

Piletimüüjatel ja koordinaatoritel tulevad erinevad müügiajaloo vaated. Piletimüüjatel (Joonis 7) tulevad vaated üksteise järel ning need on sorteeritud alates viimasest kuni esimese sisestuseni. Piletimüüjad saavad piletiajaloos olevaid sisestusi kustutada. Kuvamisel näidatakse kellaaega, piletite kogust ning selle järel kustutamise nupp. Vajutades nuppu kustuta, küsitakse üle, kas müüja on kindel, et soovib vastavat ajaloo sisestust eemaldada.

Koordinaatori vaates avanevad esimesena piletimüügi punktid, kus ühele reale paigutatakse kaks müügipunkti nime nuppudena. Igale punktile vajutades näevad koordinaatorid kogu infot sisestuste kohta, ka kustutatud pileteid valitud punktis.

Joonis 7

1.4 Tehnoloogilised valikud

Rakenduse loomiseks võtab autor kasutusele hiljuti tutvutud võimalustega, kuidas luua rakendus HTML, CSS ning JavaScripti toel. Valituks osutus PhoneGap, mis on tasuta ja avatud lähtekoodiga raamistik ning võimaldab luua mobiiltelefoni rakendusi kasutades standardiseeritud veebipõhiseid API platvorme (Adobe Systems Inc, 2015). Rakenduse arendamisel kasutatakse koodi kirjutamisel HTML standardeid, kujunduslikult antakse ilu juurde CSS sisendiga ning funktsionaalsust lisatakse JavaSriptiga. Valmivale rakendusele lisatakse juurde Parse andmebaas JavaScripti liidestusega.

PhoneGap rakendusele on võimalik juurde lisada Java põhjal töötavaid liideseid, mille küsimiseks kasutatakse Cordova poolt loodud JavaScripti. Eelmainitud võimaluse kaudu võtab autor kasutusele Parse Push meetodi, mis võimaldab saata teateid teistesse seadmetesse ning seda rõhutatult.

1.4.1 Parse andmebaas

Autor valis uudsema lähenemise, võttes kasutusele Parse andmebaasi lahenduse enam levinud MySQL andmebaasi asemel. Parsel on nii JavaScripti kui ka Java abil kasutatav Androidi dokumentatsioon, mis annab arendajale võimaluse kasutada rohkelt Parse võimalusi Android seadmes. Lisaks on olemas teised võimalused, kuidas Parse andmebaasi kasutada erinevatel operatsioonisüsteemidel.

Parse on pilveserveril töötav rakenduslik liides, mis muudab andmete vahendamise kiireks ning turvaliseks. Parse'l on ka sisse ehitatud analüüsi võimalus, mis annab igapäevase ülevaate rakenduse kasutusest link.

1.4.2 Parse Push

Parse Push loob rakendusele otsetee kasutajateni jõudmiseks (Parse, 2015). Parse Push'i kasutab autor oma töös teadete saatmise eesmärgil. See võimaldab saata teateid otse soovitud seadmesse ning seejärel jääb sellele märge, et keegi on seadmele teate saatnud. (Joonis 9). Funktsionaalsuse poolest töötab samal põhimõttel nagu Google Mail uue kirja teade või rohkem tuntud Facebooki teade.

1.4.3 Angular JS

AngularJS on struktuurne raamistik dünaamilistele veebirakendustele. See võimaldab kasutada HTML'i kui loodava malli keelt ja võimaldab laiendada HTML süntaksit, et väljendada oma rakenduse komponente selgelt ja lühidalt. Angular andmete sidumine ja sõltuvuse tekitamine võimaldab eemaldada liigse koodi kirjutamist mida muidu kirjutatakse. Ja see kõik juhtub veebilehitsejas, mistõttu on ideaalne partner mistahes serveripoolsele tehnoloogiale (Angular JS, 2015). Angulari kasutab autor töö ühtse veebirakenduse kuvamiseks.

2 Rakenduse arendus

Rakenduse arendust algustati emase tühja rakenduse loomisega, mille autor tegi ära seminaritöö käigus. Tühjale rakendusele tuli hakata ehitama sisu sellele peale. Käesolevas peatükis kirjeldab autor samm sammulist rakenduse ehitamist. Põhjaks kasutab autor seminaritöös loodud rakenduse põhja (Naaber, 2015).

2.1 Angular raamistiku paigaldus.

Etapi esimese asjana tuli luua Angular põhi. Selleks tuli luua kolm uut javascripti faili – app.js, services.js ning controller.js. Lisaks JavaScriptidele loodi ka Angulari tarbeks templiidid. Angular paigalduse käigus lähtutakse vajalikust osast ning ei kasutata kõiki Angular funktsioone ega võimalusi.

2.1.1 Angular Javascrpit

Esialgu tuleb luua app.js faili uus *angular.module* (koodinäide 1), milleks saab sisuala, mis hoiab teisi meetodeid ühtsena (Angular JS, 2015). Moodulile lisatakse juurde nimi näite puhul ’starter’ ning see järel alammoodulid, mis kasutavad ’starter’ moodulit. Näidise juures lisatakse juurde hiljem vajalikud *starter.controllers* ning *starter.services* (Koodinäide 1).

```
angular.module('starter', ['starter.controllers', 'starter.services'])
```

Koodinäide 1

Peale module loomist lisatakse juurder *.config* funktsioon, millele antakse kaasa väärtused $stateProvider (Brewer, 2015), mis toimib nagu marsruuter, kuid keskendub kindlale olekule (state) ja $urlRouterProvider (DeKrey, 2015), mis vastutab kuvatava vaate eest.

Navigeerimiseks lisame .config funktsioonile juurde .state meetodid (Koodinäide 2), mis määravad ära iga navigaatori kaardil oleva elemendi. State meetodeid tuleb lisada nii palju, kui on erinevaid alamlehti mõeldavas navigeerimispuus. Loodavale rakendusele tuleb lisada kaht tüüpi *state* meetodit. Esimene neist on *abstract* tüüpi, mida loetakse kui esimese astme menüüna. Esimesele tüübile lisatakse juurde väärtused ülem menüü nimi, milleks näites on *tab, url* ehk sihtaadress, *abstract* tõene (true) ehk määratakse ülem menüüks ning *templateUrl* ehk templiidi asukoht.

Teiseks *non-abstract*, mis on juba vaikimisi määratud vääraks (false), mida arvestatakse kui teise astme menüüks (child) *abstract* tüübile (Knight, 2015). Teisele tüübile määraratakse alammenüü nimi, milleks näites on *tab.dash*, mis on eraldatud punktiga. Punktist eespool on määratud ülemmenüü nimi. Lisaks antakse kaasa väärtused url ehk sihtaadress ning siis vaadete (views) andmed. Vaadete andmete hulka kuuluvad vaate nimi, näites tab-dash, mida kasutatakse navigeerimisel tundmaks ära, millist vaadet vajutamisel soovitakse kuvada. Sellele antakse kaasa templiidi link ning kontroller, mis hakkab valitud vaate infot kuvama.

```
.state('tab', {
    url: "/tab",
    abstract: true,
    templateUrl: "templates/tabs.html"
})
.state('tab.dash', {
    url: '/dash',
    views: {
      'tab-dash': {
        templateUrl: 'templates/tab-dash.html',
        controller: 'DashCtrl'
      }
    }
 })
```

Koodinäide 2

Config funktsiooni lõppu lisatakse esmase vaate url *$urlRouterProvider* abiga ($urlRouterProvider.otherwise('/tab/login');
). Loodava rakenduse puhul on selleks vaateks login. Selle lisamisega on app.js fail valmis.

```
$urlRouterProvider.otherwise('/tab/login');
```

Koodinäide 3

Järgmiseks lisatakse controller.js faili kontrollerid, mida kasutatakse, et laiendada Angulari ulatust (Angular JS, 2015). Kontroller kasutab funktsioonis lisaväärtust *$scope*, mis kutsub välja *state* vaikeväärtusena esitatud HTML vaate.

Alustuseks lisatakse uus moodul ning antakse talle nimi tulenevalt app.js tuletatud esimene alammoodul ehk *'starter.controllers'* (`angular.module('starter.controllers', [])`
Koodinäide 4).

```
angular.module('starter.controllers', [])
```

Koodinäide 4

Peale mooduli loomist hakatakse sellele lisama juurde *.controller* väärtusi. Neid väärtuseid tuleb lisada sama palju, kui app.js neid defineeriti. Igale kontrollerile antakse nimi ning funktsioon, millega hakatakse vaadet genereerima. Loodava rakenduse funktsioonile lisatakse kaasa vaikeväärtused *$scope* ning teenuse (service) nimi (

```
.controller('LogOut', function($scope , LogOut) {

})
```

Koodinäide 5).

```
.controller('LogOut', function($scope , LogOut) {

})
```

Koodinäide 5

Andmete kätte saamiseks teenusest tuleb väärtustada muutuja ning anda selle väärtuseks teenuse all tagastatav funktsioon (

```
.controller('SalesCtrl', function($scope, SalesInfo) {
    var $scope.salesinfo = SalesInfo.all();
})
```

Koodinäide 6).

```
.controller('SalesCtrl', function($scope, SalesInfo) {
    var $scope.salesinfo = SalesInfo.all();
})
```

Koodinäide 6

Kontroller vaatleb samuti nupuvajutusi, millele on lisatud silt *ng-click* ning väärtustatud selle kontrolleri all olev funktsioon koos vaikeväärtusega. Näiteks lisatakse nupule silt *ng-click='get_info_id(ID)'* (Koodinäide 7). Peale nupuvajutust käivitab kontroller vastavanimelise funktsiooni, mis oli *ng-click'le* juurde lisatud.

```
.controller('SalesCtrl', function($scope, SalesInfo) {
    var $scope.salesinfo = SalesInfo.all();
    $scope.get_info_id = function($ID){
        $scope.newinfo=SalesInfo.getInfobyID($ID);
    };
})
```

Koodinäide 7

Järgmiseks lisatakse services.js faili tehaseid (factory), mis aitavad luua uue teenuse kasutades selleks funktsiooni nulli või mitme argumendiga (

```
.factory('Dash', function() {
    var tickets = getalltickets();
    return {
        all: function() {
            return tickets;
        }
    }
})
```

Koodinäide 8). Selle funktsiooni tagastatud väärtus on koheselt kasutatud vastavas teenuses (Angular JS, 2015). Tagastatav väärtus teenusele peaks olema JSON massiivi vormis, sest vastav tulemus edastatakse kontrollerile, mis tulemi vaatesse kuvab.


```
.factory('Dash', function() {
    var tickets = getalltickets();
    return {
        all: function() {
            return tickets;
        }
    }
})
```


Koodinäide 8

2.1.2 Angular templiidid

Käesolevas peatükis kirjeldab autor 3 erineva templiidi loomist. Esimeseks templiidiks luuakse index.html, mis kogu rakenduse käigus saab olema põhjaks. Põhivaatele lisatakse juurde erinevaid väiksemaid kujundatud HTML tükke.

index.html tuleb lihtsa ülesehitusega tavaline HTML leht (Koodinäide 9), kus sisse imporditakse eelloodud Angular JavaScript failid ning body elemendile lisatakse juurde märgend *ng-app='starter'*. ng-app käivitab app.js failis Angular mooduli. Sealt edasi ehitatakse üles esimene vaade, milleks on eelmises peatükis $urlRouterProvider'is määratud vaade.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-
scalable=no, width=device-width">
    <title>RallyEstonia</title>
     <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.15/angular.min.js"
></script>
    <script src="js/app.js"></script> // lisame app.js faili
    <script src="js/controllers.js"></script> // lisame controllers.js
faili
    <script src="js/services.js"></script> // lisame services.js faili
  </head>
  <body ng-app="starter">
    <nav-view class="nav-container">
      <tabs class="tabs-icon-top tabs-color-active-positive">
        <tab title="Status" icon-off="ios-pulse" icon-on="ios-pulse-strong"
href="#/tab/dash">
          <nav-view name="tab-dash"></nav-view>
        </tab>
        <tab title="Sisestus" icon-off="ios-chatboxes-outline" icon-
on="ios-chatboxes" href="#/tab/entries">
           <nav-view name="tab-entries"></nav-view>
        </tab>
        <tab title="Müügi info" icon-off="ios-heart-outline" icon-on="ios-
heart" href="#/tab/sales">
          <nav-view name="tab-sales"></nav-view>
        </tab>
        <tab title="Positisioneeri" icon-off="ios-gear-outline" icon-
on="ios-gear" href="#/tab/position">
          <nav-view name="tab-position "></nav-view>
        </tab>
        <tab class="logout" style="display:none !important;" title="Logout"
icon-off="ios-gear-outline" icon-on="ios-gear" href="#/tab/logout">
          <nav-view name="logout"></nav-view>
        </tab>
        <tab class="login" title="Login" icon-off="ios-gear-outline" icon-
on="ios-gear" href="#/tab/login">
          <nav-view name="login"></nav-view>
        </tab>
      </tabs>
    </nav-view>
  </body>
</html>
```

Koodinäide 9

Index.html vaatele lisatakse juurde ka navigeerimis menüü *nav-view*. *nav-view'le* lisatakse sisubloki osa elemendina *tabs*, mis hoiab navigeerimis nuppe, elemente *tab*, ühes koos. Iga *tab* element omab endas elementi *nav-view* koos sildiga *name*, mis vajutamisel annab rakendusele teada, millist vaadet on vaja kuvada.

Teiseks kirjeldatakse avakuva loomist (

```
<view view-title="Dashboard">
  <content class="padding">
    <div class="list card">
      <div class="item item-divider">Müüdud piletid/passid</div>
      <div class="item item-body count" >
        <list>
          <item class="item-avatar  {{count.id}}" ng-repeat="count in
counts" type="item-text-wrap" >
            <div class="count">{{count.name}}: {{count.total}}</div>
          </item>
        </list>
      </div>
    </div>
    <div class="list card">
      <div class="item item-divider">Saada teade!</div>
      <div class="item item-body">
        <div>
          <input type="text" class="message" placeholder="Sõnum">
          <button ng-click="sendpush()">Saada push</button>
        </div>
      </div>
    </div>
    <div class="list card">
      <div class="item item-divider">Kommentaarid:</div>
      <div class="item item-body commentarea" >
        <list>
          <item class="item-avatar  {{comments.type}} {{comment.id}}" ng-
repeat="comment in comments" type="item-text-wrap" >
            <div class="count">{{comment.title}}</div>
            <div class="count">{{comment.created}}</div>
          </item>
        </list>
      </div>
    </div>
  </content>
</view>
```

Koodinäide 10), kus kuvatakse müüdud piletite koguseid, tähtsamaid teateid, mis on tulnud koordinaatoritelt ning võimalus saata teadet koordinaatorile.

```
<view view-title="Dashboard">
  <content class="padding">
    <div class="list card">
      <div class="item item-divider">Müüdud piletid/passid</div>
      <div class="item item-body count" >
        <list>
          <item class="item-avatar  {{count.id}}" ng-repeat="count in
counts" type="item-text-wrap" >
            <div class="count">{{count.name}}: {{count.total}}</div>
          </item>
        </list>
      </div>
    </div>
    <div class="list card">
      <div class="item item-divider">Saada teade!</div>
      <div class="item item-body">
        <div>
          <input type="text" class="message" placeholder="Sõnum">
          <button ng-click="sendpush()">Saada push</button>
        </div>
      </div>
    </div>
    <div class="list card">
      <div class="item item-divider">Kommentaarid:</div>
      <div class="item item-body commentarea" >
        <list>
          <item class="item-avatar  {{comments.type}} {{comment.id}}" ng-
repeat="comment in comments" type="item-text-wrap" >
            <div class="count">{{comment.title}}</div>
            <div class="count">{{comment.created}}</div>
          </item>
        </list>
      </div>
    </div>
  </content>
</view>
```

Koodinäide 10

Esimese kui ka kolmandas elemendis klassiga list kasutatakse silti ng-repeat, mis annab võimaluse korrata vastava elemendi sisu nii mitu korda, kui on vastavas massiivis elemente. Vastavad massiivid, mida kasutatakse jõuavad vaateni läbi kontrolleri, mis vastavalt käsitlevale vaadet korrigeerib. Nupp sildiga ng-click annab võimaluse käivitada kontrolleri funktsiooni, mis omakorda teeb ära küsitava pärigu ning tagastab seejärel teate või vastuse. Selleks võib olla massiiv, tõeväärtus (boolean)

2.2 Parse funktsioonid

Parse platvorm pakub täielikku serveripoolset lahendust mobiilsele rakendusele. Eesmärgiks on kaotada täielikult vajadus kirjutada server põhist koodi.

Parse JavaScript SDK põhineb populaarsel Backbone.js raamistikul. See vastab olemasolevatele Backbone rakendustele minimaalsete muudatustega looja koodis. Parse eesmärk on vähendada konfiguratsiooni ja võimaldada arendajal kiiresti alustada ehitamist JavaScripti ja HTML5 rakendust Parse'l (Parse, 2015).

2.2.1 Parse andmebaasi

Loodavas prototüübis kasutatakse kolme liiki Parse objekti funktsioone – salvestamine, päringu esitamine ning olemasoleva uuendamine. Parsega alustamiseks tuleks luua uus konto või kasutada juba olemasolevat. Sisselogimisel luuakse uus tühi rakendus, millele antakse kaasa genereeritud API võtmed (
Joonis 8). Neist seitsmest võtmest kasutatakse rakenduses JavaScript võtit (JavaScript Key) ning Rakenduse ID(Application ID). Selleks tuleks JavaScripti failis luua uus funktsioon näiteks välja databaseQuerys() ning seejärel kutsuda initialize funktsioon (

```
function databaseQuerys(){
      Parse.initialize("Application ID ", "JavaScript Key");
}
```

Koodinäide 11), mis võimaldab JavaScriptil saada ühedust Parsega .


```
function databaseQuerys(){
      Parse.initialize("Application ID ", "JavaScript Key");
}
```


Koodinäide 11

Loodid funktsiooni hakatakse kasutama uute funktsioonide prototüüpide loomiseks. Loodavale rakendusele luuakse 11 funktsiooni, mis kasutavad eelmainitud Parse objekti funktsioone. Autor käsitleb peatükis igast funktsioonist ühte näitena.

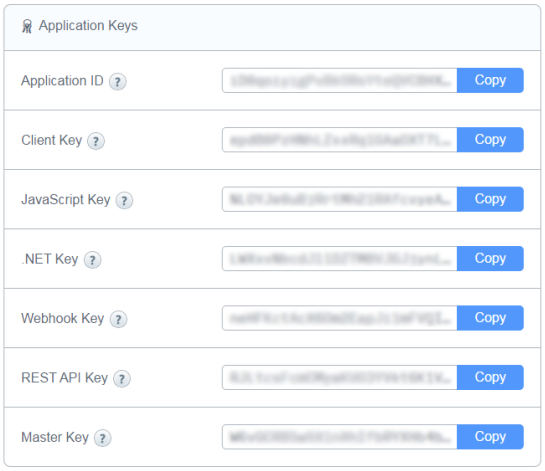
Andmete salvestamiseks luuakse prototüüp funktsioon *databaseQuerys.prototype .addComment*, mis salvestab kommentaaride tabelisse kasutaja ID, kommentaari, määrab ära vaikimisi seadistatud tähtsuse ning loetud staatuse. Funktsioonile antakse kaasa parameeter *_comment* ehk teksti alasse kirjutatud jutt. Funktsiooni siseselt defineeritakse ära kolm muutujat *currentUser*, mis pannakse võrduma sisse logitud kasutaja andmetega, *newComment*, mis pannakse võrduma Parse andmetabeli objektiga ning *newCommentObj*, mis on võrdne uue *NewComment* andmetabeli objektiga. *.set('tabeli tulba nimi', väärtus)* käsuga määratakse ära tulba nimi ning väärtus. *.save()* funktsioon salvestab eelnevalt määratud väärtused andmebaasi. Peale õnnestunud salvestamist antakse teada, et kommentaar on saadetud. Esinenud vea puhul antakse kasutajale teada, et saatmine ebaõnnestus ning palutakse proovid uuesti

```
databaseQuerys.prototype.addComment = function (_comment){
   var currentUser = Parse.User.current();
   var newComment = Parse.Object.extend("kommentaaride tabel");
   var newCommentObj = new newComment();
    newCommentObj.set("user_id", currentUser.id);
    newCommentObj.set("comment", _comment);
    newCommentObj.set("important",0);
    newCommentObj.set("read", 0);
    newCommentObj.save(null, {
    success: function(){
      alert("Kommentaar on saadetud!");
    },
    error: function(error){
      alert("Saatmine ebaõnnestus, palun proovi uuesti"");
      return false;
    }

  })
  return true;
}
```

).

```
databaseQuerys.prototype.addComment = function (_comment){
   var currentUser = Parse.User.current();
   var newComment = Parse.Object.extend("kommentaaride tabel");
   var newCommentObj = new newComment();
    newCommentObj.set("user_id", currentUser.id);
    newCommentObj.set("comment", _comment);
    newCommentObj.set("important",0);
    newCommentObj.set("read", 0);
    newCommentObj.save(null, {
    success: function(){
      alert("Kommentaar on saadetud!");
    },
    error: function(error){
      alert("Saatmine ebaõnnestus, palun proovi uuesti"");
      return false;
    }

  })
  return true;
}
```

Koodinäide 12



Joonis 8

Andmete pärimiseks luuakse prototüüp funktsioon *databaseQuerys.prototype.getUserPostition*, mis küsib Parsest kaustaja positsiooni andmeid. Valitud funktsioon on vajalik, teadmaks kuhu punkti piletimüüja minema peab ning koordinaatoritele nägemaks, kas piletimüüjad on oma õigetes punktides. Funktsiooni siseselt defineeritakse ära kolm muutujat *currentUser*, mis pannakse võrduma sisse logitud kasutaja andmetega, *Position*, mis pannakse võrduma Parse andmetabeli objektiga ning *positionExistsQuery*, mis on võrdne uue Position andmetabeli objektiga. *.equalTo('tabeli kirje', 'kasutaja_ID')* paneb positsiooni tabelis kasutaja ID võrdseks sisse logitud kasutaja ID'ga ning .find funktsiooniga leitakse andmetabelist üles vastav kasutaja, juhul kui valitud seal olemas on. Tagastus väärtuseks Pos on *.find()* funktsioonist tulenev teave JSON kujul ning see omistatakse muutujale *realuser*. Kätte saadud tulemus *realuser* muutujana tagastatakse küsitatavasse kohta b.Promse kujul (Koodinäide 13).

```
databaseQuerys.prototype.getUserPostition = function(){
      var currentUser = Parse.User.current();
      var Position = Parse.Object.extend("positsiooni tabel");
      var positionExistsQuery = new Parse.Query(Position);
      positionExistsQuery.equalTo("user_id", currentUser.id);
      var realuser = positionExistsQuery.find({
        success: function(Pos) {
           return Pos;
        }
      })
      return realuser;
}
```

Koodinäide 13

Andmete uuendamiseks kasutatakse mõlemal eelmainitud funktsioone *.set(), .find()* ning *.save()*. Näitena luuakse funktsioon *databaseQuerys.prototype.setPosition*, kus määratakse positsiooni tabelisse, müüja asukoha koordinaadid. Funktsioonile antakse kaasa xcoord ja ycoord väärtused, mis tulevad GPS'st. Funktsioonis defineeritakse ära kolm muutujat *currentUser*, mis pannakse võrduma sisselogitud kasutaja andmetega, *newPos*, mis pannakse võrduma Parse andmetabeli objektiga ning *getuser*, mis on võrdne uue *newPos* andmetabeli objektiga. *.equalTo('tabeli kirje', 'kasutaja_ID')* paneb positsiooni tabelis kasutaja ID võrdseks sisse logitud kasutaja ID'ga ning *.find()* funktsiooniga leitakse andmetabelist üles vastav kasutaja, juhul kui valitud seal olemas on. Peale *find* funktsiooni lõpetamist kutsutakse välja funktsioon *.then()*, mis on eelnevale järgmine samm. Sellele antakse kaasa *find()* funktsioonist saadud andmed. Seejärel *.set('tabeli tulba nimi', väärtus)* käsuga määratakse ära tulba nimi ning väärtus ning *.save()* funktsioon uuendab eelnevalt määratud väärtused andmebaasis (Koodinäide 14).

```
databaseQuerys.prototype.setPosition = function (xcoord,ycoord){
   var currentUser = Parse.User.current();
   var newPos = Parse.Object.extend("positisiooni tabel");
   var getuser = new Parse.Query(newPos);
   getuser.equalTo("user_id", currentUser.id);
    getuser.find().then(function(data){
        data[0].set("xacoords", String(xcoord));
        data[0].set("yacoords", String(ycoord));
        data[0].save();
      });
}
```

Koodinäide 14

2.2.2 Parse Push

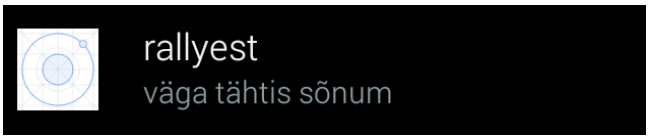
Valitud võimalus võimaldab saata teateid otse seadmesse, kuhu soovitakse ning seejärel sihtkohale jääb maha märge, et keegi on seadmele teate saatnud. Parse Push installimiseks tuli

lisada PhoneGap MainActivity java faili juurde lisada funktsioon, mis aktiveeriks seadmes Parse. Selleks tuli *onCreate* funktsiooni lisada Koodinäide 15 olev koodilõik.

```
Parse.initialize(this, "App ID", "Client Key");
ParsePush.subscribeInBackground("Urho", new SaveCallback() {
    @Override
    public void done(ParseException e) {
        if (e == null) {
          Log.d("com.parse.push", "successfully subscribed to the broadcast
channel.");
        } else {
          Log.e("com.parse.push", "failed to subscribe for push", e);
        }
      }
    });
```

Koodinäide 15

Peale java koodi lisamist tuli avada AndroidManifest.xml faili ning sinna juurde panna parse poolt staatilised koodijupid (Koodinäide 16 ning Koodinäide 17), kus tuli ära muuta !IMPORTANT teatega väärtused. Need tuli asendada rakenduse pakettidega. Uuel installimisel käivitus Parse funktsioon ning lisas andmebaasi juurde uue seadme. Rakenduse töötamise ajal teate saatmisel tuli seadme ekraanile märguande märgend (Joonis 9).



Joonis 9

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"
/>

<!--
  IMPORTANT: Change "com.parse.starter.permission.C2D_MESSAGE" in the lines
below
  to match your app's package name + ".permission.C2D_MESSAGE".
-->
<permission android:protectionLevel="signature"
    android:name="com.parse.starter.permission.C2D_MESSAGE" />
<uses-permission android:name="com.parse.starter.permission.C2D_MESSAGE" />
```

Koodinäide 16

```
<service android:name="com.parse.PushService" />
<receiver android:name="com.parse.ParseBroadcastReceiver">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED" />
    <action android:name="android.intent.action.USER_PRESENT" />
  </intent-filter>
</receiver>
<receiver android:name="com.parse.ParsePushBroadcastReceiver"
    android:exported="false">
  <intent-filter>
    <action android:name="com.parse.push.intent.RECEIVE" />
    <action android:name="com.parse.push.intent.DELETE" />
    <action android:name="com.parse.push.intent.OPEN" />
    </intent-filter>
</receiver>
<receiver android:name="com.parse.GcmBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND">
  <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE" />
    <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
    <!--
      IMPORTANT: Change "com.parse.starter" to match your app's package
name.
    -->
    <category android:name="com.parse.starter" />
  </intent-filter>
</receiver>
```

Koodinäide 17

3 Analüüs

Veebipõhise rakenduse loomine oli autorile uus väljakutse ning püüti kasutada hiljuti omandatut. Esimeste katsetustega ei saadud kuidagi vajalikke koode sedasi tööle, et poleks midagi parandada. Rakenduse loomise käigus tuli ette mitmeid õnnestumisi kui ka valusaid tagasilööke.

PhoneGapi puhul muutus häirivaks faktoriks testimine, kuna koguaeg tuli Node.js'ga rakendust ehitada ning seadmele installida. Iga selline tegevus võttis ajast 10 – 20 sekundit. Esimestel päevadel tuli installida rakendust kümneid kordi ning ajaliselt kaotati kokku päevas keskeltläbi ühe tunni, mis arenduses on päris arvestatav aja kadu. Vahest ei leidnud Node.js ka seadet üles, millele installitavat rakendust lisada, mille lahendamine võttis omakorda palju aega. Hilisemas faasis, kui rakenduse funktsioonid said töökorda, ei tekkinud enam nii palju seisakuid ning installimisrohkus langes märgatavalt. Tunni jooksul tuli teha maksimaalselt 10 installi.

Kõige keerulisem oli tegeleda vigade leidmisega. Kuna Node.js andis välja väga suurel hulgal erinevaid veateateid, mis on omavahel seotud, siis õige rea tuvastamine ning seejärel koodist otsimine oli tülikas ning aeganõudev. Vead tulid sisse uute ning tundmatute kohtade peal. Näiteks osutus esialgu võimatuks Parse Push lisamine rakendusele, sest Java ja Parse ühendamine läbi JavaScripti osutus keerulisemaks, kui juhendis näidati. Vahepeal idee külmutati, kuni muu osa sai valmis. Seejärel võeti Push uuesti päevakorda ning teisel katsel õnnestus see tööle saada.

Koheselt õnnestus autoril tööle saada Angular raamistik, mis on kogu rakenduse tuum. Sellel töötas kogu veebi ülesehitus ning funktsionaalsus. Lisaks Angularile sai kiirelt üles seatud ka andmebaasi funktsioonid. Parse Core kohapealt tekkis küll teatud probleeme, kuid need said probleemi lahenduse kiirelt. Veidi raskem osa oli saada üheskoos tööle Angular ja Parse, kuna viimane tagastas andmeid b.Promse kujul, siis jäi segaseks, mida järgmiseks teha, et andmeid kätte saada. Appi tuli Angular tugi, kus sai küsitud probleemse teema kohta täpsemat juhendit.

Seminaritöö ning bakalaureusetöö kogemustest võib autor välja tuua, et suuremate rakenduste programmeerimisel ei ole mõistlik kasutada aeganõudvaid installimisi läbiviivaid põhjasid. PhoneGap antud juhul sobiks küll väiksemate rakenduste loomiseks. Parse andmebaasina tundub olevat sobilik rakenduste programmeerimisel, sest liidestamine rakendusega on

mitmekesine ning võimalik on lisada andmebaasi paljudele platvormidele. Angular on autori silmis näis hea väljund, millega saab luua mitmekesist veebipõhja mugavate funktsioonidega.

Kokkuvõte

Bakalaureusetöö käigus viidi läbi intervjuu, mille käigus saadi teada rakendusele vajalikud funktsionaalsed nõuded ning arutati teatud määral ka kujunduslike aspektide üle. Vastavalt sai välja toodud põhifunktsioonid ning pikemalt sai kirjeldatud kujundusprotsessi.

Järgmise etapina pandi kokku andmebaasi tabelid, kuid töö käigus kirjeldati seda pindmisel määral, sest rakendatud andmebaas on loodud ärilisel eesmärgil. Andmebaasis hoitakse lisaks kirjeldatud valikutele ka tundlikke andmeid, mis ei tohiks sattuda kolmandatele osapooletele. Samas etapis valiti ka veebiraamistik Angular ja andmebaasi valik Parse ning lisaväärtusena Parse väljund Push.

Peale valikute tegemist asuti rakendust looma. Kuna rakenduse loomise algus sai tehtud seminaritöös, siis kasutas autor sama põhja ning alustas arendust alatest sealt. Arenduse käigus toodi välja esimesena Angulari JavaScripti funktsioonid, mis võimaldavad veebiraamistikku tööle panna ühtse tervikuna. Teisalt loodi Angular malli näidised, kus kirjeldati põhilisi elemente ning silte, mis arenduses käiku läksid.

Peale Angulari võttis autor ette Parse andmebaasi funktsioonid, mis võimaldavad andmebaasist küsida, sinna lisada andmeid ning sealseid kirjeid muuta. Lisaks kirjeldab autor Parse Pushi installatsiooni ning kasutuskohti ja toob näite kuidas rakenduse teated jõuavad seadmesse ning kuvab seda ekraanile.

Autor jääb valitud tehnoloogiliste valikutega rahule, kuigi töö käigus ilmnes mitmeid probleeme, kuid nendest saadi jagu.

Bakalaureusetöö eesmärgiks sai võetud prototüübi loomine ning seatud eesmärk sai täidetud. Valmis prototüüpi kasutab autor põhjana, et jätkata edasi arendust muutmaks rakendust täiuslikumaks.

Summary

PhoneGap Application: the Case of Rally Estonia Ticket Sales

For this bachelors thesis, an interview was contacted which specified the needed functionality and design for the application. After the interview the main functionality and design was documented.

The next step was building the database scheme, which is only descripted briefly in the thesis because the database was created for commercial use and already contained user data which needs privacy protection.

While planning the database, the final discussion on the wireframe and database engine was made. The main wireframe would be written on Angular and Parse database would be used. The author of the thesis already started the development on the application in his seminar work, so the same base was used.

While the application was developed, AngularJs-es functionality for creating web-wireframes was documented first. After that example theme templates were created describing the main elements used in the application. After the main wireframe as done with Angular, the author implemented Parse database. Parse allows to read, write and Update data in databases.

The author documented Parses Push installation uses, with examples on how the application receives data from the database and shows it on the screen.

The main goal of the bachelors thesis was to develop a prototype, which was successful. The prototype will be used in further development by the author.

Kasutatud kirjandus

Adobe Systems Inc. (2015). *PhoneGap | Home*. Allikas: http://phonegap.com/

Angular JS. (2015). *Angular Controller*. Allikas: https://docs.angularjs.org/guide/controller

Angular JS. (2015). *AngularJS: Developer Guide: Modules*. Allikas: https://docs.angularjs.org/guide/module

Angular JS. (2015). *AngularJS: Developer Guide: Providers*. Allikas: https://docs.angularjs.org/guide/providers

Angular JS. (2015). *What is Angular?* Allikas: https://docs.angularjs.org/guide/introduction

Brewer, M. (2015). *ui-router - Home*. Allikas: https://github.com/angular-ui/ui-router/wiki

DeKrey, M. (2015). *URL-Routing*. Allikas: https://github.com/angular-ui/ui-router/wiki/URL-Routing#urlrouterprovider

Knight, B. (2015). *Nested States & Nested Views*. Allikas: https://github.com/angular-ui/ui-router/wiki/Nested-States-%26-Nested-Views

Naaber, T. (2015). *PhoneGap analüüs - Eelseadistused bakalaureusetöö tarbeks*. Allikas: http://www.tlu.ee/~toomas91/seminarit%C3%B6%C3%B6/Seminari%20t%C3%B6%C3%B6%20Toomas%20Naaber.pdf

Parse. (2015). *Parse JavaScript Guide*. Allikas: https://parse.com/docs/js_guide

Parse. (2015). *Parse Push*. Allikas: https://parse.com/products/push

TechTerms. (2015). *API (Application Program Interface) Definition*. Allikas: http://techterms.com/definition/api

TechTerms. (2015). *CSS (Cascading Style Sheet) Definition*. Allikas: http://techterms.com/definition/css

TechTerms. (2015). *HTML (Hyper-Text Markup Language) Definition*. Allikas: CSS (Cascading Style Sheet) Definition

TechTerms. (2015). *JavaScript Definition*. Allikas: http://techterms.com/definition/javascript

TechTerms. (2015). *JSON (JavaScript Object Notation) Definition*. Allikas: http://techterms.com/definition/json

TechTerms. (2015). *RGB (Red Green Blue) Definition*. Allikas: http://techterms.com/definition/rgb

TechTerms. (2015). *SDK (Software Development Kit) Definition*. Allikas: http://techterms.com/definition/sdk

TermTechs. (2015). *GPS (Global Positioning System) Definition*. Allikas: http://techterms.com/definition/gps