TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

ITI40LT
Berta Härsing 134918 IAPB

MEDICENCY- MEDITSIINILINE ABIMEES SINU NUTISEADMES

bakalaureusetöö

Juhendaja: Roger Kerse
Üldinformaatika
õppetool
lektor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Berta Härsing

13.05.2015

Annotatsioon

Käesoleva lõputöö “Medicency- meditsiiniline abimees Sinu nutiseadmes” põhieesmärgiks on kiirabitöötaja esmase abi parandamine abivajajale ennem sündmuspaigale jõudmist. Teiseks eesmärgiks on tuua ühiskonnale kergemini kättesaadavaks ja leitavaks põhilised esmaabivõtted ning arstid erinevates Eestimaa linnades, broneerida vastuvõtuaegu jms.

Töö tulemusena valmis MEAN tehnoloogia komplekti ja Ionic raamistikuga mobiilides kasutatav rakendus, mis võimaldab õppida erinevate esmaabivõtete kohta ja testida ennast hiljem, võtta hädaolukorras kiirabitöötajaga ühendust ning näha erinevate arstide kohta infot ja broneerida vastuvõtuaegu.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 5 peatükki, 11 joonist.

Abstract

Medicency- medical help in your smart device

The thesis “Medicency- medical help in your smart device” primarly aims to improve ambulance first aid before even arriving. Anothor objective is to bring first aid techniques closer to community and information about doctors in Estonian cities. Medicency app should make reserving appointment to doctor easier.

The result is application of MEAN stack and Ionic framework, that allows to learn about various techniques of first aid and test yourself later, stay in touch with ambulance, see the information about different doctors in Estonian cities and book appointments.

The thesis is in Estonian and contains 33 pages of text, 5 chapters, 11 figures.

Lühendite ja mõistete sõnastik

AJAX	Asynchronous JavaScript and XML, asünkroonne JavaScript ja XML
API	Application Programming Interface, rakendusliides
Ctrl	Controller, kontroller
CSS	Cascading Style Sheets, kaskaadlaadistik
DOM	Document Object Model, dokumendiobjektide mudel
GPS	Global Positioning System, globaalne positsioneerimissüsteem
HTML	HyperText Markup Language, hüpertekst- märgistuskeel
HTTP	HyperText Transfer Protocol, hüperteksti edastusprotokoll
JS	JavaScript
MIT	Massachusetts Institute of Technology, Masschuetts'i tehnoloogia instituut
MVC	Model-view-controller, mudel-vaade-kontroller
SDK	Software Development Kit, arendustarkvara
SMTP	Simple Mail Transfer Protocol, lihtne meiliedastus protokoll
SASS	Syntactically Awesome Stylesheets, kaskaadlaadistik
XML	Extensible Markup Language, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Sissejuhatus

Antud bakalaureusetöö “Medicency- meditsiiniline abimees Sinu nutiseadmes” ajendas kirjutama tuttav, kelle elukutseks on inimesi hädaolukorras aidata ehk kiirabitöötaja. Õnnetused on paratamatud juhtuma ning sageli tunnevad kiirabiarstid, et ennem sündmuskohale jõudmist oleks hea näha reaalses ajas ka kannatanu olukorda. Mõnikord aitaks see rohkem ette valmistada, oleks võimalus jälgida inimese seisundit kohale jõudmiseni, hoiaks aega kokku ja inimese elu saaks veel päästa. Teiseks positiivseks küljeks oleks võimalus kiirabitöötajal juhendada kannatanu kõrval olijat (kui sündmuskohal juhtub olema rohkem inimesi), mida teha, et kannatanul kergem oleks ning kuidas antud olukorras käituda.

Lõputöö põhieesmärgiks on muuta esmaabi andmine kannatanule meedikute poolt paremaks. Teine eesmärk oleks harida inimesi meditsiinivaldkonnas ja muuta erinevate esmaabivõtete kohta lugemise mugavamaks ning lihtsalt kättesaadavaks. Eesmärgiks oleks ka kaardistada erinevad perearstid Eestimaa linnades ning võimaldada antud rakenduse kaudu vastuvõtule broneerimise võimalikuks.

Antud lõputöö koosneb mobiilsest rakendusest kasutajale, veebirakendusest meditsiinitöötajale, veebirakendusest arstidele ning 37 lehest dokumendist, mis hõlmab 5 peatükki ning 11 joonist.

1. Ülevaade kasutatud tehnoloogiatest

Antud lõputöö raames kasutas autor MongoDB-d[9] andmebaasina, Angular.js[1] oli veebilehele ilmuva kasutajaliidese realiseerimise keeleks, Node.js[13] ning Express.js[5] taga rakenduse ehitamiseks. Kunas tegemist on hübriidse mobiilirakendusega, siis kasutas autor Ionic't [7] ning selleks, et oleks võimalus erinevaid riistvaralisi funktsionaalsusi rakendusele lisada, oli kasutusel Cordova[3]. Twilio[16] ja Google'i API-t kasutati selleks, et erinevaid funktsionaalsusi teostada. Järgnevalt on lahti seletatud erinevad tehnoloogiad.

1.1 MEAN tehnoloogia komplekt

Rakenduse arendamiseks kasutas autor MEAN tehnoloogiat. MEAN on vaba ja avatud lähtekoodiga JavaScripti tarkvara kooslus, millega on hea ehitada dünaamilisi veebilehti ja –rakendusi. MEAN tehnoloogia komplekt koosneb MongoDB-st, Express.js-ist, Angular.js-st ja Node.js-ist. Eesrakendus ja serveri pool on võimalik kirjutada JavaScript'iga. MEAN sai oma nime MongoDB arendaja Valeri Karpovi loomena, kus ta seda termi oma blogipostituses mainis. [6].

MEAN kasutab igal pool andmete haldamiseks JSON-i, mis hoiab kokku aega refaktoreerimisele ja andmete õigele kujule saamiseks. GET ja POST päringud on tänu sellele väga lihtne teha ja erinevate rakendusliideste ehk API-ide kasutamine on lihtsustatud (autor kasutas rakenduses nt Google'i ning Twilio API-t). [12]

1.2 MongoDB

Andmebaasiks kasutas autor antud bakalaureusetöö raames MongoDB-d. MongoDB on tasuta ja avatud lähtekoodiga andmebaas. MongoDB on mitterelatsiooniline andmebaas, dokument-orienteeritud ja seda kasutatakse peamiselt JSON-tüüpi dokumentide salvestamiseks, mis teeb andmete integratsiooni lihtsamaks. [9]

MongoDB on ehitatud eelkõige pilve jaoks [17], lubades automaatset horisontaalset partitsiooni andmetest andmebaasis. See tähendab seda, et andmebaasi read andmebaasi tabelis on hoitud eraldatuna ning pole veergudesse lisatud. Selline ehitus aitab parandada kiirust ja tootlikkust just ärirakendustel. [4]

Autor valis antud lõputöö raames MongoDB, kunas päringud on JSON formaadis ning skeem on võimalik defineerida täielikult koodi poole peal. Kogu projekt on võimalik kirjutada ühes keeles, mis muudab arendamise lihtsamaks. 2015. aasta seisuga on MongoDB populaarsuselt maailmas neljandal kohal. Näiteks LinkedIn kasutab MongoDB-d oma serveri poolse andmebaasina. [10]

1.3 Angular.js

Angular.js on avatud lähtekoodiga struktuurne JavaScript'i raamistik, mis on loodud Google'i poolt ning peamiselt ühe lehe rakenduste jaoks. Angular.js'i andmete sidumine ja sõltuvuse süstimine vähendab koodi, mida muidu peaks kirjutama. Angular.js töötleb AJAX-i ja DOM-i loogikat, mida muidu peaks koodina kirjutama. Disaini eesmärgid on siduda lahti DOM-i manipulatsioon raskest loogikast. See võimaldab arendada paralleelselt ja kasutada mõlemaid pooli samaaegselt. Angular.js rakendab MVC mustrit eraldada välimust, andmeid ja loogika komponente. Sõltuvuse süstimine võimaldab ka koormust serverile vähendada. [1]

Angular.js parandab koodi testitavust. Testimine on seda raskem, mida raske on koodi struktuur. Kui teha direktiive ja komponente, siis muudavad need koodi paremini loetavamaks ning ka kergemini testitavaks. Testida on võimalik Angular.js-i sellise raamistikuga nagu Jasmine. Jasmine on raamistik, millega on võimalik testida JavaScript'i koodi ning ei vaja DOM-i. [8]

Angular.js'l on stabiilseks versiooniks 1.5.5. Autor valis Angular.js-i, sest Angular.js'il on palju häid omadusi, mis on täpsemalt välja toodud 3. peatükis.

Angular.js 2 on väljas beeta versioonis. Angular.js 2 on paranenud mobiilse arenduse poolest, palju põhi loogikat on liigutatud moodulitesse ning näiteks dünaamiline lehe laadimine iseloomustavad hetkel beetas olevat Angular.js 2-te. Angular.js 2 peaks

muutma ka suuri rakendusi kiiremaks ja parandama lehe laaditavust märkimisväärselt. Angular.js 2 on arendatud Google'i poolt ning koostööd on tehtud ka React.js-i arendajatega ja React.js-i parimad palad on samuti sisse toodud. [1]

1.4 Express.js

Express.js on Node.js serveri raamistik, mida on võimalik kasutada erinevate veebirakenduste raames. Autor valis Express.js'i, sest kunas lahendus on hübriidrakendusena tehtud, muudab see rakenduse arendamise lihtsaks ja rakenduse kergesti kasutatavaks. Expressi kasutatakse ka serveri poolse osana MEAN tehnoloogias. Express muudab http ja marsruutimise kergesti teostatavaks ning lihtsaks. Expressi kasutab näiteks Netflix. [5]

1.5 Node.js

Node.js on vabavaraline käituskeskkond, arendamaks serveripoolselt veebirakendust, ta on ühelõimeline ning sündmussilmusega. Node.js ei ole JavaScript, kuid peamised moodulid on JavaScript'is kirjutatud ning arendajad saavad samuti uusi mooduleid JavaScript'is kirjutada. [13]

Node.js'il on sündmuspõhine arhitektuur, mistõttu on võimalik asünkroonne I/O. Selleks, et Node.js'iga rakendust käivitada on mitu võimalust- Node.js'iga käivitada server.js fail või npm-iga installida Nodemon[13], mis käivitab ise rakenduse, kui seal muudatusi teha. Pole vaja iga kord rakendust taas käivitada, vaid npm-iga alla laadida Nodemon[13], mis teeb selle kasutaja eest ise ära ja muudab arendamise kergemaks. [13]

Nodemailer[14] on lisa Node.js'ile, et oleks võimalik saata e-maile. Kõige lihtsamalt on võimalik see installida npm-iga. Nodemailer'iga on võimalik saata e-maile kasutades lihtsat meiliedastusprotokolli või kasutades näiteks transpordi lisa. Autor kasutas Nodemailer'it bakalaureusetöös e-mailide saatmiseks arstidele. Autor valis Nodemaileri, sest Node.js'i kasutades soovis hoida ühtset joont ning proovida erinevaid võimalusi, mida Node.js pakub. [14]

1.6 npm

npm on pakettide haldus JavaScript'i käitluskeskkonnale. npm jookseb käsurea abil ja haldab sõltuvusi rakenduse jaoks. Selle abil saab alla laadida Node.js rakendusi, mis on npm-is olemas. Antud töö raames laadis alla autor npm-i abil Twilio[16], Express.js, MongoDB ning selleks, et serveri poolt ehitada veel ka Express-session. [15]

1.7 Ionic

Ionic on HTML5 SDK, mis võimaldab ehitada mobiilirakendusi, kasutades tehnoloogiaid nagu HTML, CSS, JavaScript, positiivne on see, et täiesti tasuta. Ionic lihtsustab eessüsteemi. Ionic on põhiliselt välimus ja ilu rakenduse juures, välja on töötanud selle MIT. Ionic võimaldab korraga arendada nii Androidi kui IOS operatsioonisüsteemidele. [7]

Ionicu kasutamiseks on vaja Angular.js-i, mis hetkel autori lõputöö raames oli olemas. Angular.js'iga on võimalik luua erinevaid animatsioone ja teisi liideseid, mis muudavad rakenduse kasutaja jaoks ilusaks ning kergesti kasutatavaks. Ionicul on väga palju erinevaid seemneprojekte, millega alustades on juba kõik vajalik esialgseks arendamiseks olemas. Ionicu alla laadides on olemas juba Sassi ja JS raamistiku lisad. [7]

Üks peamisi põhjusi, miks autor valis Ionicu, oli ka väga ilus ja mugav kasutajaliides. Javascript-i ja CSS-i komponendid katsid enamiku rakenduse jaoks vajaliku ära. Juba sisse ehitatud $ionicPopup teenus lubab kergesti välja kutsuda teateid kasutajale, muutes rakendust palju kasutajasõbralikuks [11].

1.8 Cordova

Apache Cordova (varasemalt PhoneGap) on mobiilirakenduste arendamise raamistik. Cordovaga on võimalik CSS3, HTML5 ja JavaScript'i abiga ehitada mobiilirakendusi. Cordova lubab sisse pakkida CSS-i, JavaScript'i ja HTML-i sõltuvalt seadme

platvormist. Tulemuseks on hübriidne mobiilirakendus, mis tähendab, et nad ei ole puhatad/algsed mobiilirakendused aga mitte ka puhtalt veebi põhised. [3]

Apache Cordova rakendused kasutavad CSS3 ja HTML5-t välimuse jaoks ning funktsionaalus/loogika on JavaScript'is. HTML5 abil saab ligipääsu näiteks GPS-ile, sammulugejale ja kaamerale, mida autor lõputöös kasutas. [2]

1.9 Twilio

Twilio on pilve kommunikatsiooni firma USA-s, Californias. Twilio globaalse pilve API kaudu on võimalik muuta keerulised telekommunikatsiooni riistvaravahendid minevikuks. Twilio abil on võimalik veebi, lauaarvuti ja mobiili tarkvara arendada. [16]

Antud lõputöös kasutas autor kiirabitöötaja ja abivajaja videokõne loomiseks Twilio API-t. Twilio API võimaldab video –ja tavakõnesid, sõnumeid, autentimist, kõnede jälgimist jms. Twilio on suurfirmade (Uber, Coca Cola) jaoks väga populaarne ning seetõttu otsustas autor kogemuse saamiseks seda kasutada. Samuti on Twiliol olemas Authy, mis vastutab mõlema poole autentimise eest. [16]

2. Medicency rakenduses kasutatavad Angulari elemendid

2. 1 Kontroller

Kontrollerid on Angular.js'is defineeritud kui JavaScript'i konstruktorfunktsioonid, mida kasutatakse, et defineerida Angular.js'i käsitlusala objekti.

Kontrollereid on võimalik defineerida erinevat moodi, üks võimalus on mallis, mis kasutab ng-controller direktiivi: nt ng-controller = "DoctorsCtrl", või siis näiteks teine võimalus on nt app.js failis malli korral defineerida, millist kontrollerit ta kasutab.

```
.state('tab.learn-detail', {
  url: '/learn/:id',
  views: {
    'tab-learn': {
      templateUrl: 'templates/learn-detail.html',
      controller: 'LearnDetailCtrl'
    }
  }
});
```

Joonis 1. Kontrolleri defineerimine app.js failis

Kui kontroller lisatakse näiteks "Controller as" süntaksiga, siis kontrolleri instants määratakse omandiks uuele käsitlusala objektile.[1]

2. 2 Andmete sidumine

Andmete sidumine on Angular.js rakendustes automaatne andmete sünkroniseerimine mudeli ja vaadete vahel. Vaade on projektsioon mudelist. Kui mudel muutub, muutub ka vaade. Angular.js'i mallid töötavad erinevalt. Esiteks mall kompileeritakse brauseris. Kompileerimise samm loob tõelise vaate. Kõik muudatused vaatel muudavad ka mudelit ja vastupidi. Kunas vaade on ainult projektsioon mudelist, siis kontroller on täielikult eemaldatud vaatest. See muudab testimise kergemaks, sest on lihtne testida kontrollerit, mis on isoleeritud vaatest ja sõltuvustest [1].

2.3 Teenused

Angular.js'i teenused on asendamatud objektid, mis on ühendatud kokku kokkukuuluvuse süstimise abil. Teenused on vajalikud selleks, et kood oleks organiseeritud ning kergesti hallatav ja kasutatav rakenduses. Angular.js'i teenus initsialiseeritakse ainult siis, kui rakenduse komponent on sellest sõltuv. Angular.js'il on ka sisse-ehitatud teenuseid. Sisse-ehitatud teenus algab alati $ märgiga, nt: $http. [1]

```
.controller('LearnCtrl', function($scope, $http, $location) {
  $scope.treatments = [];
$http.get('/medic/findTreatments').then(function (result) {
  $scope.treatments = result.data;

});
```

Joonis 2. GET-päring sisse ehitatud $http teenusega.

2.4 Käsitlusala objekt

Käsitlusala objekt on objekt, mis viitab rakenduse mudelile. Käsitlusala objektid on pandud arhitektuurilisse struktuuri, mis matkib DOM-i struktuuri.

Miks on käsitlusala objektide kasutamine hea:

- Käsitlusala objektid saavad algatada sündmusi ning varustada API-t, et jälgida mudeli mutatsiooni.
- Käsitlusala objekt võimaldab API-il levitada igat mudeli muutust süsteemis vaatesse.
- Neid saab kasutada, et limiteerida ligipääsu rakenduse komponentide seadistustele, lubades ligipääsu jagatud mudeli seadetele. Pesa käsitlusala objektid on "laps käsitlusala objektid" või "isoleeritud objektid".

Käsitlusala objekt andmemudelina on liim rakenduse kontrolleri ja vaate vahel. [1]

```
.controller('DoctorDetailCtrl', function($scope, $stateParams, $http, $state, $location) {
  $http.get('/medic/findDoctor/' + $state.params.id).then(function (result) {
    $scope.details = result.data;
    console.log($scope.details);
  });
```

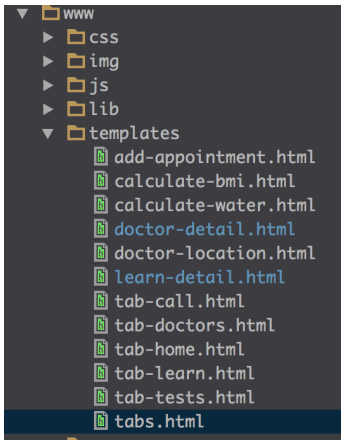
Joonis 3. Näide käsitlusala objektist $scope.details

```
<ion-view>
  <ion-nav-title>{{details.name}} {{details.lastname}}</ion-nav-title>
  <ion-content class="padding">
```

Joonis 4. HTML-is arsti detailide kuvamine

2.5 Mallid

Angular.js-s on mallid kirjutatud HTML-is ja koosnevad Angular.js-spetsiifilistest elementide atribuutidest. Angular.js kombineerib malli informatsiooniga mudelist ja kontrollerist ja muudab dünaamilise info nähtavaks sellisena, nagu kasutaja seda näeb brauseris. [1]



Joonis 5. Mallide kausta näide

```
<ion-tabs class="tabs-icon-top tabs-color-active-assertive">


  <!--- Home --->
  <ion-tab title="Home" icon-off="ion-ios-home-outline" icon-on="ion-ios-home" href="#/tab/home" style="...">
    <ion-nav-view name="tab-home" style="..."></ion-nav-view>
  </ion-tab>

  <!-- Learn board -->
  <ion-tab title="Learn" icon-off="ion-ios-book-outline" icon-on="ion-ios-book" href="#/tab/learn">
    <ion-nav-view name="tab-learn"></ion-nav-view>
  </ion-tab>

<!--- Call doctor --->
  <ion-tab title="Call" icon-off="ion-ios-telephone-outline" icon-on="ion-ios-telephone" href="#/tab/call">
    <ion-nav-view name="tab-call"></ion-nav-view>
  </ion-tab>

  <!-- Doctors Tab -->
  <ion-tab title="Doctors" icon-off="ion-ios-people-outline" icon-on="ion-ios-people" href="#/tab/chats">
    <ion-nav-view name="tab-chats"></ion-nav-view>
  </ion-tab>


  <!-- Testing Tab -->
  <ion-tab title="Tests" icon-off="ion-ios-lightbulb-outline" icon-on="ion-ios-lightbulb" href="#/tab/tests">
    <ion-nav-view name="tab-tests"></ion-nav-view>
  </ion-tab>


</ion-tabs>
```

Joonis 6. tabs.html malli ülesehitus

Autor kasutas erinevaid malle, et kood oleks kergesti hallatav. Süsteemne arhitektuur võimaldab mahukatest failidest kiiresti andmeid üles leida ning ka teistele arendajatele arusaadavaks muuta.

3. Medicency rakenduse kirjeldus

Medicency rakendus on suunatud kõigile, kes soovivad oma teadmisi meditsiinilises valdkonnas täiendada või saada infot, kuidas käituda, kui on näiteks mesilaselt nõelata saanud. Kõige põhiliseks eesmärgiks antud rakenduse puhul on võimalus hädaolukorras kiirabitöötajaga kontakti saada videokõne teel, et kannatanuga kaasas olijal oleks võimalik kiirabitöötaja juhendamisel esmaabi osutada. Kiirabi teab olla ka rohkem ettevalmistunud, olles näinud eelnevalt kannatanu olukorda.

Rakendus on jaotatud 5 erinevasse vaatesse, millest igaühel on oma funktsionaalsus, kontroller, mall jms. Viide vaatesse jaotas autor tingituna eesmärkidest, mis töö arendades endale valis.

3.1 Arendusprotsess

Rakenduse Medicency arendamist alustas autor kõigepealt märkmete tegemisega. Pani kirja, mis igas vaates olema peaks ning tegi endale paberile ülesanded iga vaate kohta, et oleks ülevaade suurest pildist ning kõik ülesanded oleks väikesteks juppideks jagatud. Joonistas üles ka erinevad vaated. Kunas tahtis rohkem õppida MEAN tehnoloogia kohta ning Angular.js meeldis autorile väga, siis tehnoloogiate valikul otsustas MEAN tehnoloogia kasuks. Rakenduse kasutamine peab olema autori soovil võimalik nii Androidi kui IOS-i operatsioonisüsteemi peal, seega otsustas Ionicu kasuks.

Arendamist alustas autor sellest, et lõi viie erineva vaate mallid ning muutis navigeerimise erinevate vaadete vahel võimalikuks. Tegi eraldi kaustad JavaScript'i, HTML-i ja CSS-i jaoks. Kohe alguses ühendas ka oma projekti versioonihaldustarkvaraga, et oleks võimalik jagada koodi juhendajale või kui töö arendamisel tekkis probleeme, oli versioonihaldustarkvaras olemas töötav lahendus. Edasi arendas iga vaate kaupa ning üritas iga kord, kui üks vaade valmis sai, teha versioonihaldustarkvarra muudatustest üles laadimise.

Arendades üritas jälgida head tava ja võimalikult palju oma koodi tükeldada, et koodi hallatavus oleks parem ning kogu kood ei oleks ühes failis. Kogu koodi hoidis autor ka inglise keeles ning üritas võimalikult arusaadavalt oma funktsioonile nimesid panna, et ka teistele oleks arusaadav, mida mingi meetod teeb või peaks tegema.

3. 2 Pealehe vaade

Pealehe vaatel on kasutajat tervitav tekst ning võimalus 2 erinevale nupule vajutades teha arvutusi oma tervise heaks. Loogika juhtimiseks on HomeCtrl- kontroller, millega on võimalik edasi navigeerida kahte erinevasse vaatesse, kus ühes on võimalik arvutada kehamassiindeksit ning teises soovituslikku veekogust päeva jooksul.

3.2.1 Päevase veekoguse arvutamine

"Sinu päevane veekogus" vaates on võimalik kasutajal arvutada päevane soovituslik veekogus, kui ta sisestab kaalu ning päevas tehtud trenni minutites. Antud vaate calculate-water.html andmete välja kuvamiseks on loodud kontroller WaterCtrl, kus on

järgnev funktsioon:

$scope.calcWater()- funktsioon arvutab soovitatava veekoguse arvestades treeningminuteid ning inimese kaalu. Funktsioon tagastab liitrite arvu kahe komakohaga ning kuvab selle calculate-water.html mallis.

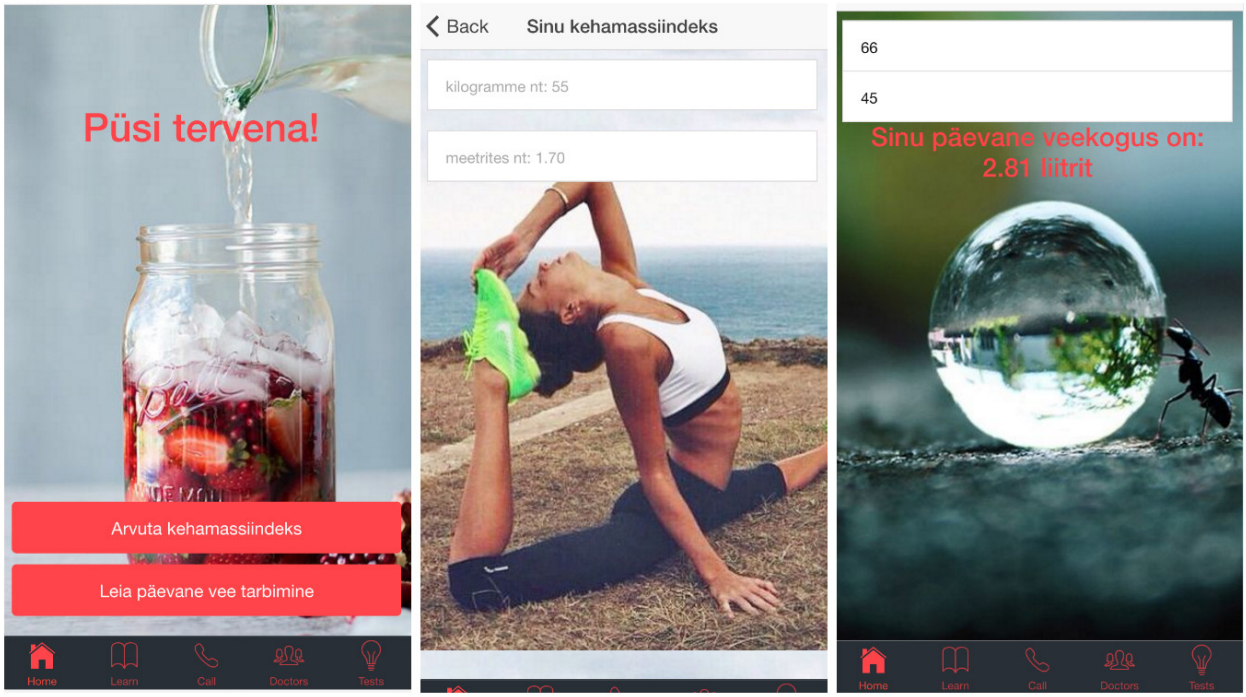
3.2.2 Kehamassiindeksi arvutamine

Antud vaates on võimalik inimesel sisestada kaal kilogrammides ning pikkus meetrites ning peale mõlema välja sisestamist kuvatakse talle kehamassiindeks ning millises kategoorias tema keha on. Kategooriaid, mille alusel jagada on 4: "rasvunud", "ülekaaluline", "ideaalkaalus" ning "alakaaluline".

Funktsioonid bmiCtrl'is:

$scope.calcBMI() – funktsioon arvutab kehamassiindeksi kasutades pikkust ning kaalu. Funktsioon tagastab kehamassiindeksi. Parameetrid, mis funktsioon sisse saab, on funktsiooniga kaasa antud ning kasutaja sisestatud sisendi ja ng-model väärtused.

$scope.categorize()- funktsioon võtab arvesse kehamassiindeksi ja tagastab vastavalt sellele kategooria, millises kaalus kasutaja on. Näiteks kui kehamassiindeks on suurem kui 30, tagastab funktsioon kategooria "rasvunud" ja kuvab selle calculate-bmi.html mallis.



Joonis 7. Pealehe näide

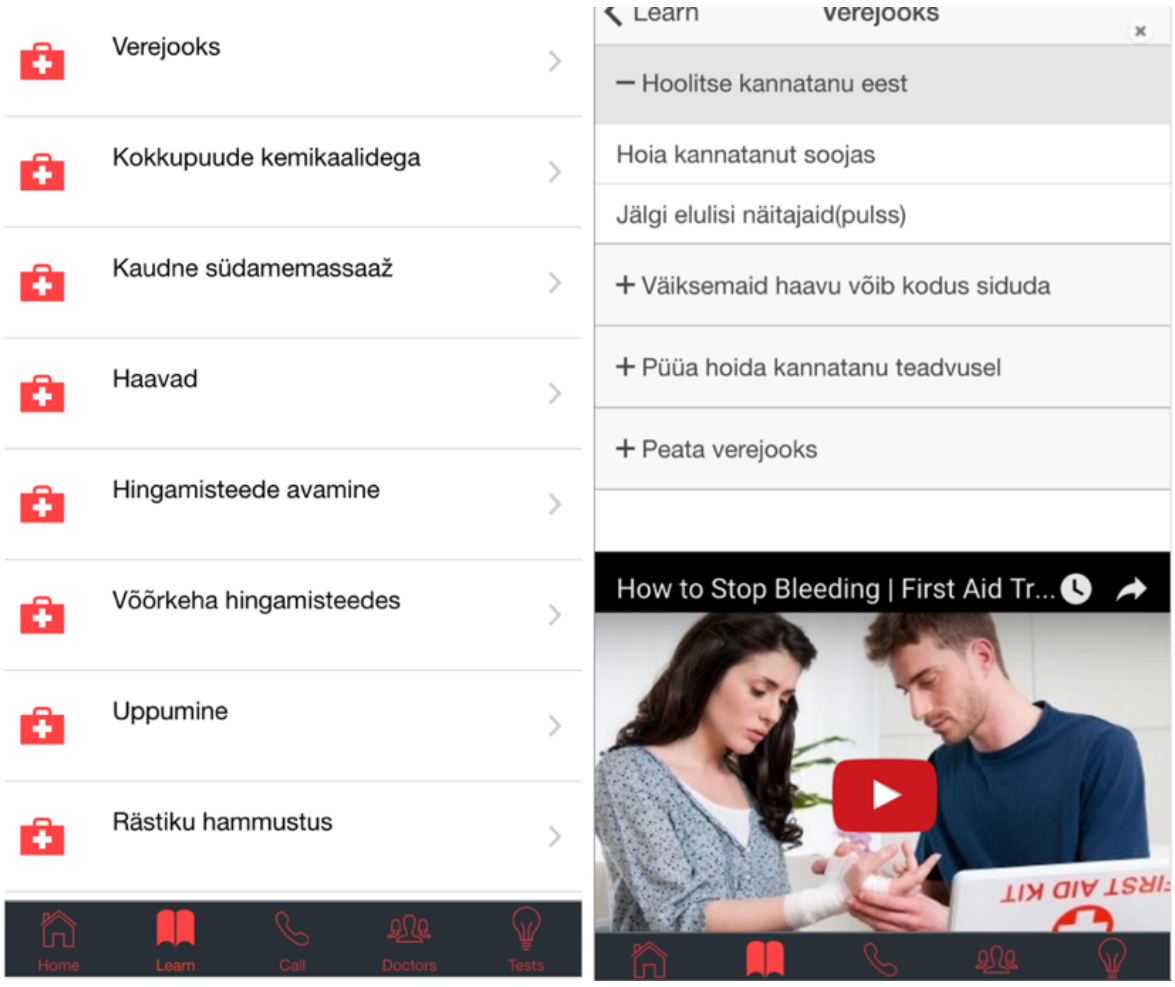
3.3. Esmaabivõtete vaade

Esmaabivõtete vaade on kuvatud Ionicu listvaatena, kus on kuvatud erinevad esmaabivõtted ning kui vajutada neist mõnele, avaneb detailvaade, kus akordioniga on võimalik täpsemalt lugeda konkreetse juhtumi kohta. Igal detailvaatel on ka video Youtube'i õpetusega.

Peamised funktsioonid antud vaadete jaoks:

$http.get('/findTreatments').then(); päring küsib andmebaasist kõik esmaabivõtted ja defineerib tulemuse $scope.treatments väärtuseks.

$scope.treatmentDetailView()- funktsioon, mis forEach tsükliga vaatab üle kõik esmaabivõtted ja tagastab detailid esmaabivõtte kohta ja suunab kasutaja esmaabivõtte detailvaatesse, kus kuvatakse esmaabivõtte kohta lisainfot (nõuanded + Youtube'i video).



Joonis 8. List esmaabivõtetest ning esmaabivõtte detailvaade

3.4 Kõne Twilio API-iga

```
var AccessToken = require('twilio').AccessToken;
var ConversationsGrant = AccessToken.ConversationsGrant;

app.get('/token', function(request, response) {
  var identity = randomUsername();
  process.env.TWILIO_ACCOUNT_SID = 'AC7b040471d000edea9f56558efad02a9c';
  process.env.TWILIO_API_KEY = 'SK9952eab61ab7e824122bd1a8fda782d6';
  process.env.TWILIO_API_SECRET = '20qa520JwjSznK0777vwdJqVyEqp13HV';
  var token = new AccessToken (
    process.env.TWILIO_ACCOUNT_SID,
    process.env.TWILIO_API_KEY,
    process.env.TWILIO_API_SECRET
  );

  // Assign the generated identity to the token
  token.identity = identity;

  //grant the access token Twilio Video capabilities
  var grant = new ConversationsGrant();
  process.env.TWILIO_CONFIGURATION_SID = 'VS6b5d504069532340999f28bb5aa0ca23';
  grant.configurationProfileSid = process.env.TWILIO_CONFIGURATION_SID;
  token.addGrant(grant);

  // Serialize the token to a JWT string and include it in a JSON response
  response.send({
    identity: identity,
    token: token.toJwt()
  });
});
```

Joonis 9. Twilio API kasutatavad võtmed

Twilio API kasutamine oli autori jaoks lihtne ning allpool erinevad funktsioonide kirjeldused:

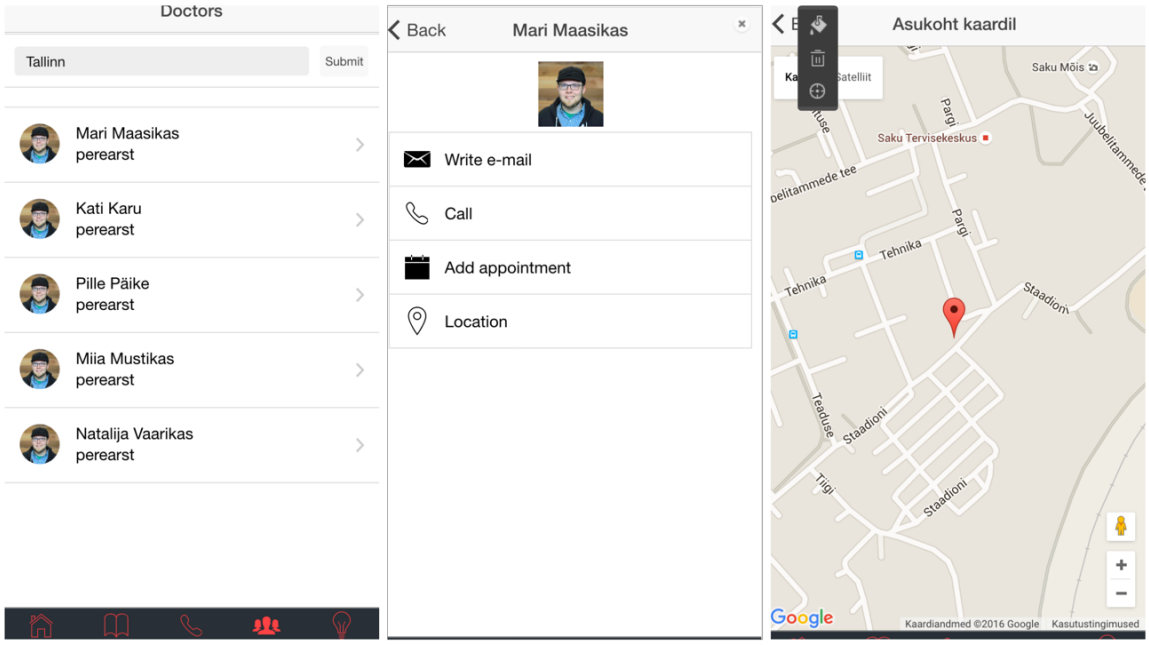
clientConnected() – Funktsioon loob peale kasutaja nupule vajutust ühenduse teise inimesega, antud juhul medicency-test-iga. Kui ühendust luua ei olnud võimalik, kuvab veateate: “Hetkel ühendust luua ei saa”.

conversationStarted(conversation)- Funktsioon, mis kuvab teise inimese ühinedes pildi temast. Kui ühendus katkeb kuvab vastavat teadet.

Window.onload()- lehe avamisel kuvatakse nupp, millega on võimalik vaadata eelvaadet veebikaameras ning nupp, mida vajutades saab meedikuga ühendust võtta. Sarnane loogika on ka Medicency meediku vaates, ainult puudub võimalus ise kõnet alustada.

Twilio API kasutades kaasnes ka turvalisuskontrolli probleem. Algselt üritas autor tööd üles panna serverisse, mis oli http protokolliga. Twilio API kasutamise eelduseks on aga ka see, et oleks https. Twilio lubab ainult turvalise ühenduse kaudu videokõnet luua. Lokaalselt näiteks *localhost:8080* on võimalik Twilio API-t kasutada, sest *localhost* on turvalisuse mõistes turvaline aadress.

3.5 Arstide vaade



Joonis 10.Arsti tuvastamine kaardilt

Arstide vaates on võimalik kuvada erineva linna järgi arste, kui antud linnas arstid puuduvad, kuvatakse kasutajale teade. Konkreetse arsti peale vajutades avaneb detailvaade, kus on erinevad võimalused. Kasutaja saab arstile e-maili kirjutada, kus ta suunatakse e-maili rakendusse ja arsti e-mail on eelkuvatud. Vajutades “Helista” nuppu, viiakse kasutaja telefoni vaatesse, kus arsti number on eelvalitud ning võimalus on helistada. Samuti on olemas võimalus näha arsti asukohta kaardil. Rakendus lubab ka kasutajal arsti aega broneerida. Kuupäeva ja kellaaja valides saab broneerida oma perearsti vastuvõttu, hiljem saadetakse arstile ning registreerunule e-mail täpsemate detailidega.

3.5.1 Funktsioonide kirjeldused DoctorsCtrl-is.

$http.get('findDoctors).then()- päring mis tagastab kõik arstid, kes baasis olemas on.

$scope.findDoctors()- funktsioon, mis võtab sisse kasutaja sisestatud linna, forEach tsükliga vaatab läbi kõik päringust leitud arstid, kui arst leitakse, kelle linn on sama, mis kasutaja soovitud linn, siis lisatakse $scope.doctors maatriksisse ja kui tsükkel lõppeb, siis tagastatakse maatriks arstidest soovitud linnaga. Kui linnad puuduvad, siis sisse ehitatud teenuse $ionicPopup'iga kuvatakse kasutajale teate, et kahjuks tema otsitud linna kohta meie süsteemis arstid puuduvad. Kasutajamugavuse kaalutlustel on seda kasulik teha, sest muidu kasutaja mõtleb, et süsteem on katki ning ei kasuta antud rakendust enam.

$scope.doctorDetailView() – funktsioon, mis forEach tsükliga käib läbi kõik arstid ja leiab id järgi arsti ja väärtustab $scope.details'i arsti väärtustega ning muudab asukohaks arsti detailvaate.

3.5.2 Funktsioonide kirjeldused DoctorDetailCtrl-is

$http.get('/findDoctor').then();- tagastab kõik andmed valitud arsti kohta id alusel

$scope.doctorLocation()- funktsioon, mis suunab kaardivaatesse kasutaja, kus tal on võimalik näha Google Maps'is arsti asukohta.

$scope.appointment()- funktsioon, mis suunab vaatesse, kus on võimalik arstiaega broneerida.

$scope.callMobile()- funktsioon, millele vajutades jõuab kasutaja numbrivaliku vaatesse, kus tal on number juba ees ja võimalik helista nuppu vajutades arstile kõne teha.

3.5.3 Funktsioonide kirjeldused MapControlleris

$http.get('findDoctor' + $state.params.id).then(); - interneti aadressis oleva id parameetri kaudu pärib serveri poolt kohale antud arsti laiuskraadi ja pikkuskraadi, mille abil Google Maps leiab asukoha kaardil.

Initialize()- Funktsioon, mis väärtustab ära pikkuskraadi ja laiuskraadi, mis serveri poolt tulid arsti kohta. Kasutades Google Mapsi API-t initsialiseerib uue kaardi, ning tagastab selle, et saaks kasutajaliideses kasutajale välja kuvada. Initialize() kutsutakse välja kohe, kui antud vaatesse sisenetakse, sest vastasel juhul vaade oleks tühi ja kuvataks kaart alles peale lehe värskendamist.

3.5. 4 Funktsioonid AppointmentCtrl-is

$http.get('/findDoctor/' +$state.params.id).then();- Päring, mis tagastab internetiaadressilt saadud id järgi arsti andmed.

$scope.isTrue()- funktsioon, mis otsib antud arsti kohta kõik päevad, mis tal on aegu, et kasutaja saaks neid broneerida. Funktsioon tagastab "tõene", kui vabu päevi leidub, vastasel juhul kuvatakse teade, et antud hetkel sellele arstile vabu aegu pole.

$scope.saveAppointment()- salvestab arsti vastuvõtu broneeringu.

- **$http.post('/submitAppointment').then();-** Salvestab baasi vastava broneeringu.
- **$http.post('/updateDoctorTime').then();-** Muudab kasutaja aja baasis vääraks, et enam keegi teine seda aega broneerida ei saaks.
- **$http.post('/sendEmailToDoctor').then();-** saadab serveri poolt andmed, mis tuleb saata e-mailiga arstile broneeringu kohta.
- **$http.post('/sendEmailToPatient').then();-** saadab serveri poolt andmed, mis tuleb saata e-mailiga broneerijale meeldetuletuseks.

3.6 Testide sooritamine

Testide vaadet juhib TestCtrl ning antud vaates on järgmised funktsioonid:

$scope.reset()- funktsioon, mis kustutab kõik varasemad tulemused ja kuvab kasutajale küsimustikku nullist.

$scope.getQuestion()- funktsioon, mis pärib quizFactory kaudu kohale küsimused, kui küsimusi rohkem pole, siis *quiz.isOver* saab tõese väärtuse.

$scope.checkAnswer()- funktsioon, mis võtab kasutaja sisestatud vastuse ja kontrollib selle õigsust. Õige korral skoori suurendatakse, vale korral jääb skoor samaks ja kuvatakse, et vastasid valesti.

$scope.nextQuestion()- Antud funktsioon suurendab id väärtust ning kutsub välja $scope.getQuestion() funktsiooni.

quizFactory- Kasutusel selleks, et $http.get('/findQuiz').then(); päringu tagastamisel tagastaks küsimuse, kui neid baasis eksisteerib.

Tehas võimaldab Angular.js'is kasutada teisi teenuseid, teenuseid initsialiseerida ning viivitatud, laiska initsialiseerimist. Tehas konstrueerib uue teenuse kasutades 0 või enamat argumenti. Tagastatav väärtus on teenuse instants [1].

3.7 Serveri pool

3. 7. 1 Nodemailer'i funktsionaalsus

Nodemailer'iga saadab rakendus broneeringu tegijale ning arstile e-maili. Luuakse transport, kuhu saab defineerida teenuse- antud bakalaureusetöö raames on e-maili teenuseks Gmail, autenditakse kasutaja, kes saadab e-mail. Transport.sendEmail() sisuks on kellelt kellele saadetakse, pealkiri ning kirja sisu.

```
var nodemailer = require('nodemailer');

app.post('/sendEmailToDoctor', jsonBodyParser, function (req, res) {
  var transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'medicency@gmail.com',
      pass: 'Medicency1994'
    }
  });
  transporter.sendMail({
    from: 'medicency@gmail.com',
    to: "Contact Us from <"+req.body.email+">",
    subject: 'Arsti vastuvõtuaeg',
    text: "Nimi: "+req.body.name+", Perekonnanimi: "+req.body.lastName+", Telefon: "+req.body.phone+", " +
    "Email: "+req.body.email+", Kuupäev: "+req.body.date+", kellaaeg: "+req.body.time+", Comments: "+req.body.comments+""
  });
  res.end("E-mail sent");
});

app.post('/sendEmailToPatient', jsonBodyParser, function (req, res) {
  var transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'medicency@gmail.com',
      pass: 'Medicency1994'
    }
  });
  transporter.sendMail({
    from: 'medicency@gmail.com',
    to: "Contact Us from <"+req.body.userEmail+">",
    subject: 'Arsti vastuvõtuaeg',
    text: "Arsti aeg broneeritud: kuupäev: "+req.body.date+", kellaaeg: "+req.body.time+""
  });
  res.end("E-mail sent");
});
```

Joonis 11. Kirja saatmise loogika Nodemailer'iga

3.7.2 Serveri pool ning ühendus andmebaasiga MongoDB

Serveri pool on kirjutatud Node.js'is ning kõigepealt on defineeritud, millised sõltuvused peavad olemas olema.

Alljärgnevalt toob autor välja serveri poole peal olevad funktsioonid:

- **app.get('/findDoctors').then();-** Loob ühenduse andmebaasiga. Kollektsioon nimega "doctors" hoiab endas kõik arste ning iga objekt lisatakse listi ning tagasihelistamisega tagastatakse tekstilisel kujul.
- **app.get('/findTreatments').then();-** Loob ühenduse andmebaasi kollektsiooniga "emergency" ning pärib sealt kohale kõik nõuanded erinevate hädaolukordade korral. Tagasihelistamisega tagastatakse list.
- **app.get(/'findTreatment/:id').then();-** Loob ühenduse andmebaasiga ning leiab "emergency" kollektsioonist kõik esmaabivõtted ja otsib id järgi kasutaja soovitud esmaabivõtte.

- **app.post(‘/submitAppointment’, jsonBodyParserer, function()).then();-** Eesrakendusest saadetud andmed jõuavad antud päringuga serveri poolele ning päring salvestab need andmeid “appointments” kollektsiooni.

- **app.post(‘/updateDoctorTime’).then();-** Antud päring teeb eesrakendusest saadud arsti id, kuupäeva ja kellaaja kohta “doctors” kollektsiooni muudatuse, *value* väärtuse muudab tõese asemel vääraks, sest vastav aeg on broneeritud.

- **app.get(‘/findDoctor/:id’).then();-** Antud päring leiab vastava id-ga arsti ja tagastab kõik andmed tema kohta.

4. Medicency rakendus kiirabitöötajatele

Antud bakalaureusetöö raames võimaldamaks kasutajal videokõnet meditsiinitöötajaga, tuli teha veebirakendus ka kiirabile. Rakendusega saab kiirabitöötaja ennast kasutajateks registreerida ning seejärel sisse logida. Kasutajale avaneb pealeht, kus tal on võimalik oodata sissetulevaid kõnesid ning kui kellelgi on hädaabiolukord anda talle nõu ning üritada võimalikult palju teda aidata kuni sündmuspaigale jõudmiseni.

4.1 Ülesehitus ja teostus

Antud veebilehe jaoks kasutas autor MEAN tehnoloogiat nagu Medicency üldise rakenduse jaoks. Angular.js kattis kasutajaliidese poole, Node.js serveri poole ning kiirabitöötajate jaoks lõi autor eraldi kollektsiooni *"medics"* MongoDB andmebaasi, kuhu salvestas registreerinud kasutaja kohta info.

4.1.1 Registreerimine ja sisse logimine

Registreerimisvaates küsitakse kasutaja kohta nime, e-maili ning parooli, millega ta edaspidi saab sisse logida. Õnnestunud registreerimise korral suunatakse kasutaja sisse logimise lehele ja kuvatakse ka teadet õnnestunud registreerimise kohta.

Sisse logimisel küsitakse kasutajalt e-maili ning parooli ja õnnestumise korral suunatakse vaatesse, kus on võimalik kõnedele vastata. Ebaõnnestumisest teavitatakse kasutajat teatega. Sisse logimisel salvestatakse kasutaja andmed sessiooni külge ning tänu sellele elab lehekülg üle ka värskendamise ja kasutaja ei pea uuesti sisse logima.

4.1.2 Twilio API-ga kõne

Videokõnet saab sarnaselt Medicency mobiilirakendusele kasutada ka kiirabitöötaja. Erandiks on vaid see, et tal puudub võimalus ise kõnet sooritada. Kõne vaates on võimalik arstil kuvada oma kaamera eelvaadet ning kui keegi ühendab ennast kõnega, siis on näha teise inimese pilti.

5. Medicency rakendus arstidele

Medicency rakendus arstidele on veebileht, millega arstidel on võimalus ennast süsteemi registreerida ning sisse logides avaneb neile vaade, kus on võimalik lisada vabu aegu, kuhu Medicency mobiilirakenduse kasutajatel on võimalik aegu broneerida. Arstidel on võimalus ka aegu kustutada, kui broneerija mingil põhjusel kohale tulla ei saa. Arstile kuvatakse ka broneeritud aegu ning talle on näha, kes tema juurde tulevad.

5.1 Ülesehitus ja toestus

Antud veebilehe jaoks kasutas autor MEAN tehnoloogiat nagu Medicency üldise rakenduse jaoks.

5.1.1 Sisse logimine ja registreerimine

Registreerimisvaates küsitakse kasutaja kohta nime, aadressi, e-maili ning parooli, millega ta edaspidi saab sisse logida. Õnnestunud registreerimise korral suunatakse kasutaja sisse logimise lehele ja kuvatakse ka teadet õnnestunud registreerimise kohta.

Sisse logimisel küsitakse kasutajalt e-maili ning parooli ning õnnestumise korral suunatakse vaatesse, kus on võimalik aegu broneerida ja vaadata. Ebaõnnestumisest teavitatakse kasutajat teatega. Sisse logimisel salvestatakse kasutaja andmed sessiooni külge ning tänu sellele elab lehekülg üle ka värskendamise ja kasutaja ei pea uuesti sisse logima.

5.1.2 Aegade broneerimine

Broneerimiseks peab arst valima sobiva kuupäeva ja kellaaja, mille peale avaneb mudel, kus ta peab täitma vajalikud väljad broneeringu kohta. Broneeringu kinnitamisel salvestatakse “appointments” kollektsiooni vastava broneeringu andmed ning muudetakse vastava aja väärtus vääraks.

Kokkuvõte

Bakalaureusetöö raames valmis mobiilirakendus, millega on kasutajal võimalik võtta videokõne teel ühendust kiirabitöötajaga. Meedikutele valmis veebirakendus, millega ta saab videokõne teel juhendada hädasolijat või tema kõrval olijat. Mobiilirakendusega on võimalik kasutajal lugeda erinevaid esmaabivõtteid ning hiljem ennast testida lugemise põhjal. Rakendus võimaldab otsida erinevaid perearste Eesti linnades ning broneerida vastuvõtuaegu oma perearstile. Arst ja kasutaja saavad broneeringu kohta e-maili teel kinnituse detailidega. Eraldi veebileht valmis ka arstidele, et nad saaksid lisada broneerimiseks aegu ning vaadata, kes neile on aegu broneerinud.

Töö positiivsetest külgedeks on reaalne töötav lahendus, mis võimaldab kõnet kiirabitöötajaga ning muudab esmaabi andmise kergemaks. Rakendusega on võimalik kergelt broneerida vastuvõtu aegu ja vältida arsti juures pikki ootejärjekordi. Bakalaureusetöö positiivseks küljeks on ka rakenduse kasutamise võimalus nii IOS kui Androidi operatsioonisüsteemis.

Antud lõputöö nõrgaks küljeks jäi reaalsete kasutajate käest arendamise käigus tagasiside küsimine. Oleks võinud rohkem uurida inimeste käest nende soove ja huvisid. Võimalik edasiarendus oleks võtta ühendust kiirabitöötajatega ning proovida neil testida antud rakendust ja võtta kuulda nende soovitusi, mis rakenduses veel olema peaks. Soov on võtta ühendust ka mõne perearstiga Tallinnas ning proovida neil arstiaegade broneerimise süsteemi kasutada ja saada neilt tagasisidet.

Bakalaureusetöö eesmärk sai täidetud. Valmis reaalselt töötav lahendus IOS-i ja Androidi operatsioonisüsteemi peale ning võimalik on kõne kahe osapoole vahel. Tänu MEAN tehnoloogia lahendusele valmis täielikult JavaScript'is kirjutatud lahendus ning kasutajal on võimalik oma teadmisi esmaabivõtetega täiendada.

Kasutatud kirjandus

1. AngularJS [WWW] https://docs.angularjs.org (10.05.2016)
2. Apache Cordova [WWW] https://en.wikipedia.org/wiki/Apache_Cordova (10.05.2016)
3. Cordova [WWW] http://cordova.apache.org/docs/en/latest/ (10.05.2016)
4. Database sharding [WWW] http://www.agildata.com/database-sharding/ (14.05.2016)
5. ExpressJS [WWW] https://en.wikipedia.org/wiki/Express.js (10.05.2016)
6. Karpov, V. MongoDB blog, *The MEAN stack: MongoDB, ExpressJS, AngularJS and MongoDB*. [WWW], https://docs.angularjs.org/guide/introduction (10.05.2016)
7. Ionic [WWW] http://ionicframework.com/docs/ (14.05.2016)
8. Jasmine [WWW] http://jasmine.github.io/2.4/introduction.html (14.05.2016)
9. MongoDB [WWW] https://en.wikipedia.org/wiki/MongoDB (10.05.2016)
10. MongoDB [WWW] https://www.mongodb.com/ (10.05.2016)
11. Moronoy, J. HTML Mobile Josh Mornoy, 8 reasons why I'm glad I switched to the Ionic framework [WWW] http://www.joshmorony.com/8-reasons-why-im-glad-i-switched-to-the-ionic-framework (14.05.2016)
12. MEAN(software_bundle) [WWW] https://en.wikipedia.org/wiki/MEAN_(software_bundle) (10.05.2016)
13. Node.js [WWW] https://en.wikipedia.org/wiki/Node.js (10.05.2016)
14. Nodemailer [WWW] https://nodemailer.com/ (14.05.2016)
15. Npm (software) [WWW] https://en.wikipedia.org/wiki/Npm_(software) (14.05.2016)
16. Twilio [WWW] https://www.twilio.com/ (14.05.2016)
17. Wayner, P. InfoWorld, *MEAN vs LAMP for the future programming* [WWW] http://www.infoworld.com/article/2937159/application-development/mean-vs-lamp-your-next-programming-project.html (14.05.2016)

Lisa 1 – Twilio API kasutamine

```
// Successfully connected!
  function clientConnected() {
    document.getElementById('invite-controls').style.display = 'block';

    conversationsClient.on('invite', function (invite) {
      invite.accept().then(conversationStarted);
    });

    document.getElementById('button-invite').onclick = function () {
      var inviteTo = 'medicency-test';
      if (activeConversation) {
        activeConversation.invite(inviteTo);
      } else {
        // Create a conversation
        var options = {};
        if (previewMedia) {
          options.localMedia = previewMedia;
        }
        conversationsClient.inviteToConversation(inviteTo, options).then(conversationStarted, function (error) {
          console.error('Unable to create conversation', error);
        });
      }
    };
  }

// Conversation is live
  function conversationStarted(conversation) {
    activeConversation = conversation;
    if (!previewMedia) {
      conversation.localMedia.attach('#local-media');
    }

    conversation.on('participantConnected', function (participant) {
      participant.media.attach('#remote-media');
    });

    conversation.on('participantDisconnected', function (participant) {
      //log("Participant '" + participant.identity + "' disconnected");
    });

    conversation.on('disconnected', function (conversation) {
      conversation.localMedia.stop();
      conversation.disconnect();
      activeConversation = null;
    });
  }

  window.onload = function () {
    document.getElementById('button-preview').onclick = function () {
      if (!previewMedia) {
        previewMedia = new Twilio.Conversations.LocalMedia();
        Twilio.Conversations.getUserMedia().then(
          function (mediaStream) {
            previewMedia.addStream(mediaStream);
            previewMedia.attach('#local-media');
          },
          function (error) {
            console.error('Unable to access local media', error);
            //log('Unable to access Camera and Microphone');
          });
      }
    };
  };
});
```

Lisa 2- Arstide kontrollerid

```
.controller('MapController', function($scope, $ionicLoading, $compile, $http, $state) {
    $http.get('/findDoctor/' + $state.params.id).then(function (result) {
      $scope.details = result.data;

    });
    function initialize() {
    var myLatlng = new google.maps.LatLng($scope.details.lat, $scope.details.lon);

    var mapOptions = {
      center: myLatlng,
      zoom: 16,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("map"),
      mapOptions);

    //Marker + infowindow + angularjs compiled ng-click
    var contentString = "<div><a ng-click='clickTest()'>Click me!</a></div>";
    var compiled = $compile(contentString)($scope);

    var infowindow = new google.maps.InfoWindow({
      content: compiled[0]
    });

    var marker = new google.maps.Marker({
      position: myLatlng,
      map: map,
      title: 'Uluru (Ayers Rock)'
    });

    google.maps.event.addListener(marker, 'click', function() {
      infowindow.open(map,marker);
    });

    $scope.map = map;
  }

  $scope.centerOnMe = function() {
    if(!$scope.map) {
      return;
    }

    $scope.loading = $ionicLoading.show({
      content: 'Getting current location...',
      showBackdrop: false
    });

    navigator.geolocation.getCurrentPosition(function(pos) {
      $scope.map.setCenter(new google.maps.LatLng(pos.coords.latitude, pos.coords.longitude));
      $scope.loading.hide();
    }, function(error) {
      alert('Unable to get location: ' + error.message);
    });
  };

  $scope.clickTest = function() {
    alert('Example of infowindow with ng-click')
  };

  $scope.$on( "$ionicView.enter", function() {
    initialize();
    google.maps.event.trigger( map, 'resize' );
  });
})
```

Lisa 3. Perearsti aegade broneerimise kontroller

```
.controller('AppointmentCtrl', function($scope, $http, $state, $filter, $ionicPopup, $location) {

  $scope.currentItem = {};
  $scope.times = {};

  $http.get('/findDoctor/' + $state.params.id).then(function (result) {
    $scope.info = result.data;
    $scope.dates = $scope.info.dates;
    $scope.email = result.data.email;
  });

  $scope.isTrue = function () {
    var flag = false;
    var datey = $filter('date')(Date.parse($scope.currentItem.OpenDate), 'dd.MM.yyyy');
    angular.forEach($scope.dates, function (date) {
      if (datey === date.date) {
        $scope.options = date.options;
        $scope.date = date.date;
        flag = true;
      }
    });
    return flag;
  };

  $scope.saveAppointment = function () {
    $scope.currentItem.date = $scope.date;
    $scope.currentItem.doctorId = $state.params.id;
    $scope.currentItem.email = $scope.email;
    $http.post('/submitAppointment', $scope.currentItem).
    then(function(resp) {
    });
    $http.post('/updateDoctorTime', $scope.currentItem).
      then(function (resp) {
    });

    $http.post('/sendEmailToDoctor', $scope.currentItem).
      then(function (resp) {
    });
    $http.post('/sendEmailToPatient', $scope.currentItem).
    then(function (resp) {
      $location.path('/tab/home');
      var alertPopup = $ionicPopup.alert({
        title: 'Broneering tehtud',
        template: 'Hoia oma tervist!'
      });

      alertPopup.then(function(res) {
      });

    })
  };
})
```

Lisa 4. Node.js serveri pool

```
app.get('/findTreatments', function (req, res) {
  var sess = req.session;
  MongoClient.connect(url, function(err, db) {
    assert.equal(null, err);
    findTreatments(db, function(treatments) {
      console.log(treatments);
      db.close();
      res.end(JSON.stringify(treatments));
    });
  });
});


var findTreatments = function (db, callback) {
  var list = [];
  var cursor = db.collection('emergency').find();
  cursor.each(function(err, doc) {
    console.log(doc);
    assert.equal(err, null);
    if (doc) {
      var newObject = {};
      newObject = doc;
      list.push(newObject);

    } else {
      callback(list);
    }
  });
};




app.get('/findTreatment/:id', function (req, res) {
  MongoClient.connect(url, function(err, db) {
    assert.equal(null, err);
    if (req.params.id && Number(req.params.id) !== NaN) {
      findTreatment(db, Number(req.params.id), function(treatment) {
        console.log(treatment);
        db.close();
        res.end(JSON.stringify(treatment));
      });
    }
  });
});


var findTreatment = function (db, id, callback) {
  db.collection('emergency').findOne({"id": id}, function(err, treatment) {
    assert.equal(err, null);
    console.log(treatment);
    if (treatment) {
      callback(treatment);
    } else {
      callback();
    }
  });
};
```