TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND

ARVUTITEADUSE INSTITUUT

INFORMAATIKA

SIIM PRUULMANN

LOGISTITI KASUTAJALIIDES MOBIILSETELE SEADMETELE

BAKALAUREUSETÖÖ (6 EAP)

Juhendaja: Vambola Leping, MSc

Tartu 2014

LogistITi kasutajaliides mobiilsetele seadmetele

Lühikokkuvõte:

Selle töö eesmärk on uurida mobiilsetele seadmetele suunatud veebirakenduste loomise lahendusi. Töö käigus luuakse prototüüp mobiilsest rakendusest, mis oleks alternatiiv olemasoleva logistikasüsteemi kasutajaliidesele. Samuti tutvustatakse tehnoloogilisi lahendusi internetiühenduseta toimivate veebirakenduste loomiseks.

Võtmesõnad:

LogistIT, veebirakendus, internetiühenduseta veebirakendus

LogistIT User Interface for Mobile Devices

Abstract:

This paper researches creating web applications for mobile devices. A prototype is built as an alternative to an existing logistics system user interface. Also an introduction is given to technologies that can be used to create offline web applications.

Keywords:

LogistIT, web application, webapp, offline web application.

Sisukord

Sissejuhatus

2011. aastal alustas LogistIT transpordiettevõtetele suunatud teenuse pakkumisega. Teenuse keskmeks on veebirakendusena loodud töövahend logistikutele. Rakenduses ühendatakse sõidukite jälgimine planeerimisega. Selline lähenemine võimaldab automaatselt avastada probleeme, enne kui need jõuavad suureks paisuda.

LogistITi praegune kasutajaliides ei ole optimaalne kasutamiseks mobiilsetel platvormidel. Kaks peamist probleemi on täielik sõltuvus internetiühendusest ja piiratud mõõtmetega ekraanidega arvestamine. Selle töö eesmärk on uurida, mis oleks parim kasutajaliidese lahendus nutitelefonidele ja luua vastav kasutajaliidese prototüüp.

Esimeses peatükis kirjeldatakse olemasolevat LogistITi süsteemi, selle komponente ja ülesehitust. Samuti kirjeldatakse täpsemalt praeguse kasutajaliidese probleeme nutitelefonidel. Teine peatükk kirjeldab võimalikku lahendust ja põhjendab tehtud valikuid. Kolmas peatükk tutvustab tehnoloogiaid, mida kasutatakse internetiühenduseta toimivate rakenduste loomiseks. Ülevaade tehakse ka veel arenduses olevast tulevikutehnoloogiast.

Lisa 1 sisaldab viiteid prototüübi lähtekoodile ja töötavale näitele.

1. Olemasolev rakendus

LogistIT on üles ehitatud klient-server põhimõttel. Andmeid hoitakse ja töödeldakse serveris. Kasutaja kasutab spetsiaalselt loodud kasutajaliidest ehk UI'd, mis suhtleb serveriga läbi *Application Programming Interface*'i ehk API. API olemasolu võimaldab luua erinevaid kasutajaliideseid. Praegu on LogistITil kasutusel üks peamine kasutajaliides, mis töötab veebibrauserites. LogistITi serveripoolel on kasutusel PostgreSQL [1] andmebaas ja API on kirjutatud Pythonis [2]. Olemasolev kasutajaliides on kirjutatud JavaScriptis [3]. Järgnevalt tutvustatakse lähemalt andmemudelit ja peamisi selles eksisteerivaid objekte.

1.1 Serveripoole andmemudel

Süsteemi peamised objektid on sõiduk (*vehicle*), tellimus (*cargo*), vedu (*trip*), inimene (*person*), ettevõte (*company*), ala (*area*), silt (*tag*), relatsioon (*relation*) ja hoiatus (*alert*). Lisaks nendele on veel mitmeid objekte, mis aitavad siduda eelnevaid omavahel loogiliselt. Tähtsamad objektid on koos selgitavate näidetega allpool lahti kirjutatud. Näidete tegemiseks on kasutusel fiktiivne transpordifirma Transport AS, mis veab treileritega üle Euroopa sõiduautosid.

1.1.1 Sõiduk

Sõiduk on kliendi kasutuses olev auto, buss, veoauto või mõni muu sõiduvahend. Sõiduk võib olla varustatud jälgimisseadmega, mis edastab infot sõiduki asukoha, liikumiste ja üldise olukorra kohta. See info võib olla reaalajas või ka pikema hilinemisega. Sõiduki üldise olukorra all saab mõista näiteks kütusepaakides oleva kütuse kogust või näiteks ka mõne sõidukil oleva lisaseadme staatust. Sõiduki kirjelduses peab olema registreerimisnumber.

1.1.2 Tellimus

Tellimus on logistikafirma kliendi soov. Eelnevalt alustatud näite puhul oleks tellimus Mati Maasika soov saada 1 sõiduauto Berliinist Tartusse. Tellimusega käivad kaasas ka ajad. Sama näidet jätkates, Mati autot saab Berliinist peale võtta 1. juunil kella 10st hommikul kuni kella 5ni õhtul ja ta soovib, et auto jõuaks hiljemalt 4. juuniks Tartusse. Logistik lepib kokku kindlad kellaajad, millal auto Berliinist peale võetakse ja millal ta Tartusse kohale peab jõudma. Seega on tellimusel klient, kaup, lähtekoht ja sihtkoht ning peale- ja mahalaadimise ajad.

1.1.3 Vedu

Vedu määrab sõiduki mingiks ajavahemikuks kasutusse. Üldiselt on vedu tellimusele auto lisamine. Tellimuse peale- ja mahalaadimise kohtadest saavad veo peatused ning nende ajad määravad veo kestvuse. Näiteks kui Transport ASi logistik määrab Mati Maasika tellimuse autole 111AAA tekib uus vedu. See vedu kestab 1. juunist kuni 4. juunini ja sellel on kaks peatust: esimene peatus on Berliinis ja teine Tartus. Veole saab lisada ka teisi tellimusi, kui näiteks Berliinist oleks samas ajavahemikus vaja veel mõni teine auto tuua. Veol võib olla ka peatusi, mis ei ole seotud tellimustega. Nii-öelda vabad peatused lahendavad olukorra, kui sõiduk peab käima näiteks teeninduses. Minimaalselt moodustavad veo seega sõiduk ja kindel ajavahemik. Kui veol on vähemalt kaks peatust planeerib süsteem automaatselt nende vahele teekonna. Teekond on kirjeldus, milliseid teid pidi kahe punkti vahel liigutakse. Rohkem kui kahe peatuse korral planeeritakse teekond iga kahe järjestikuse punkti vahele. Logistikul on võimalik teekondi vastavalt vajadusele muuta. Planeeritud teekondade põhjal teeb süsteem aja- ja distantsihinnanguid. Samuti võrreldakse neid sõidukil oleva jälgimisseadme saadetava infoga. Planeeritud teekonnast kõrvalekaldumiste avastamisel luuakse hoiatus.

1.1.4 Persoon

Kõige selgemad näited on LogistIT süsteemi kasutava ettevõtte töötajad. Eelkõige sõidukite juhid ja logistikud. Persoonidena peaks olema kirjeldatud kõik kelle kohta on vaja hoida mingit informatsiooni (näiteks kontaktandmeid). Muuhulgas peab ka igal LogistIT süsteemi kasutajal olema omanimeline persoon. Igal persoonil ei pea küljes olemas kasutajat. Näiteks logistikul on süsteemi sisenedes persoon, kellena ta paistab teistele kasutajatele. Persoonide üks peamisi omadusi on, et neid saab siduda teiste objektidega. Kui logistik loob uue tellimuse märgitakse automaatselt tema persoon selle tellimusega seotuks. Selline sidumine võimaldab kohe näha, kes tegeleb tellimusega. Samuti saavad kasutajad luua seoseid persoonide ja teiste objektide vahel. Näiteks saab sõidukiga registreerimisnumbriga 111AAA siduda autojuhi Raul Rool. Nüüd on Raul Rooli persooni vaadates näha, et ta on parasjagu sõiduki 111AAA juhiks määratud. Sellise suhte loomisel saab määrata algus- ja lõpuaja.

1.1.5 Ettevõte

Ettevõte kirjeldab juriidilist isikut. Transport AS puhul oleks neil ettevõtetena kirjeldatud süsteemis firmad, kellele nad pakuvad oma teenust ja firmad kelle teenuseid nad ise kasutavad. Sisuliselt kõik, kellega neil on äriline suhe. Peamine kasutus süsteemis on kirjeldada tellimuse klienti. Ettevõtetel on süsteemis nende nime põhjal määratud lühikood, mida kasutatakse tellimuse koodide kokkupanekul. Näiteks Klient OÜ lühikood on „KL” ning kõik nende tellimused saavad endale ainulaadse koodi, mis algab selle lühikoodiga. Näiteks „KL-1001”.

1.1.6 Ala

Ala kirjeldab geograafilist kohta. Selleks on ta määratud vähemalt kolme koordinaadiga. Kõik kohad on süsteemis kirjeldatud aladena – tellimuste siht- ja lähtekohad, vedude peatused. Lisaks ala kirjeldusele, mis on jada geograafilisi punkte, on alal ka nii-öelda

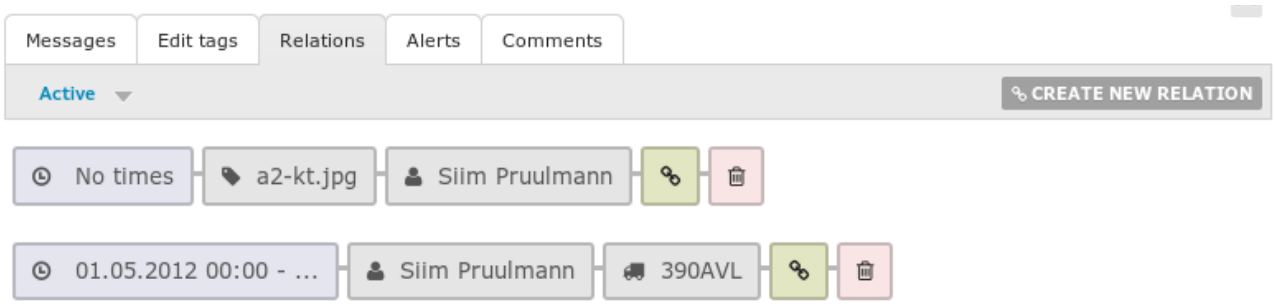
sisenemispunkt (*entry point*), mis on oluline teekondade määramisel. Sisenemispunkt määrab alast algava või seal lõppeva tee otspunkti. Ala võib katta väga suure piirkonna, sisenemispunkt tagab, et planeeritav tee viiks näiteks platsi värava juurde, mitte kuhugi suvalisse kohta platsi taga. Alad pakuvad võimaluse automatiseerida mitmeid logistiku tegevusi. Üks selliseid on sõiduki kohalejõudmise äramärkimine. Kui süsteem näeb sõiduki jälgimisseadme saadetavate andmete põhjal, et sõiduk on jõudnud oma teos oleva veo mingisse peatusse märgitakse automaatselt sinna jõudmise aeg. Samuti tehakse alade põhjal analüüsi, kui kaua sõiduk on kindlates alades viibinud. Võimalik on ka otse teada anda logistikule, näiteks SMSiga, kui sõiduk jõuab ettemääratud alasse.

1.1.7 Silt

Kõigile objektidele saab külge panna silte (*tag*). Silte on kahte tüüpi. Lihtsad, kus nad pole enamat kui märksõna, mis on objekti külge pandud. Tavapärane kollane kleebitav märkmepaber oleks parim kirjeldus. Sildid võivad olla ka niiöelda targad sildid. Sellised sildid omavad sildi kirjeldusega määratud efekti. Targad sildid saavad kanda endas rohkem informatsiooni, omada kehtivusaegu ja olla nõudvad sildid. Nõudvat silti on kõige lihtsam kirjeldada näitega. Veol võib olla silt Liikluskindlustus, mis on nõude tüüpi ja väärtusega -1. See tähendab, et selle nõude täitmiseks peab olema ka silt Liikluskindlustus, mis on täitvat tüüpi ja mille number on 1. Oletame, et seda veost vedaval sõidukil on selline silt olemas, mis ka parasjagu kehtib. Nende numbrite summa on 0 ja seega on nõue täidetud. Nõude silt on aegumatu ja nõuab, et koguaeg oleks Liikluskindlustuse silt olemas. Täitval sildil on ajad, mis peegeldavad reaalse liiklukindlustuspoliisi kehtivust. Kui sildi kehtivusaeg läbi saab, ei ole see enam aktiivsena sõiduki küljes. Sellises olukorras jääb nõudev silt rahuldamata ja süsteem tekitab hoiatuse, et sõiduk on ettenähtud sõitma aga tal pole selles ajas kehtivat liikluskindlustust. Tarkade siltide süsteem on laiendatav väga paljudele erinevatele võimalikele situatsioonidele.

1.1.8 Relatsioonid

Eelnevalt on juttu olnud objektide sidumisest. Üheks meetodiks objektide seostamisel on relatsioonid. Relatsioon on põhimõtteliselt algusaeg, lõpuaeg ja kaks või rohkem objekti. Relatsioonidega seotakse omavahel sõidukeid ja juhte, sõidukeid ja logistikuid, veoautosid ja treilereid ja palju muud. Pilt 1 demonstreerib kahte relatsiooni, nagu neid originaalses kasutajaliideses kuvatakse. Esimene kujutatud relatsioon on persooni ja manuse vahel. Teine relatsioon on persooni ja sõiduki vahel ning omab ka algusaega.



Pilt 1 – Relatsioonide näide. Persoon Siim Pruulmann on alates 1. maist 2012 seotud sõidukiga numbrimärgiga 390AVL

1.1.9 Muud objektid

Süsteemis on veel hoiatused (*alert*), sõnumid (*message*), manused (*attachment*), peatused (*stop*), tellimuse peatused (*cargo stop*), kommentaarid (*comment*) ja teekonnad (*route*).

Hoiatused tekivad nii-öelda lahendamata olekus. Enamasti väljendavad hoiatused mingit probleemi, näiteks planeeritud aja ületamist, planeeritud teest mööda sõitmist või midagi muud. Hoiatusi võib tekitada ka lihtsalt informatsiooni edastamiseks. Neid saab märkida jälgitavateks (logistik teeb hoiatusele märke, et ta tegeleb sellega) ja lahendada. Viimasel juhul tuleb märkida ka lahenduse põhjendus ning hoiatus eemaldatakse aktiivsete hoiatuste nimekirjast.

Sõnumid on hetkeseisuga SMSid ja e-mailid. Sõnumid on saavad olla nii süsteemi sissetulevad kui väljaminevad. Sissetulevatest sõnumitest antakse märku uue hoiatusega.

Manused on failid, mida hoitakse süsteemis. Relatsioonide abil saab neid siduda teiste süsteemis olevate objektidega.

Peatused on kooslus alast ja ajast, millal sõiduk peaks antud alas olema. Kahe järjestikuse peatuse vahele planeeritakse teekond. Teekond on jada koordinaate, mis kirjeldab teid, mida mööda liikuda esimesest alast teise.

Kommentaar on tekst, mis seotakse ühe kindla objektiga süsteemis. Kommentaaril on samuti kindel autor – persoon, kes kirjutas kommentaari.

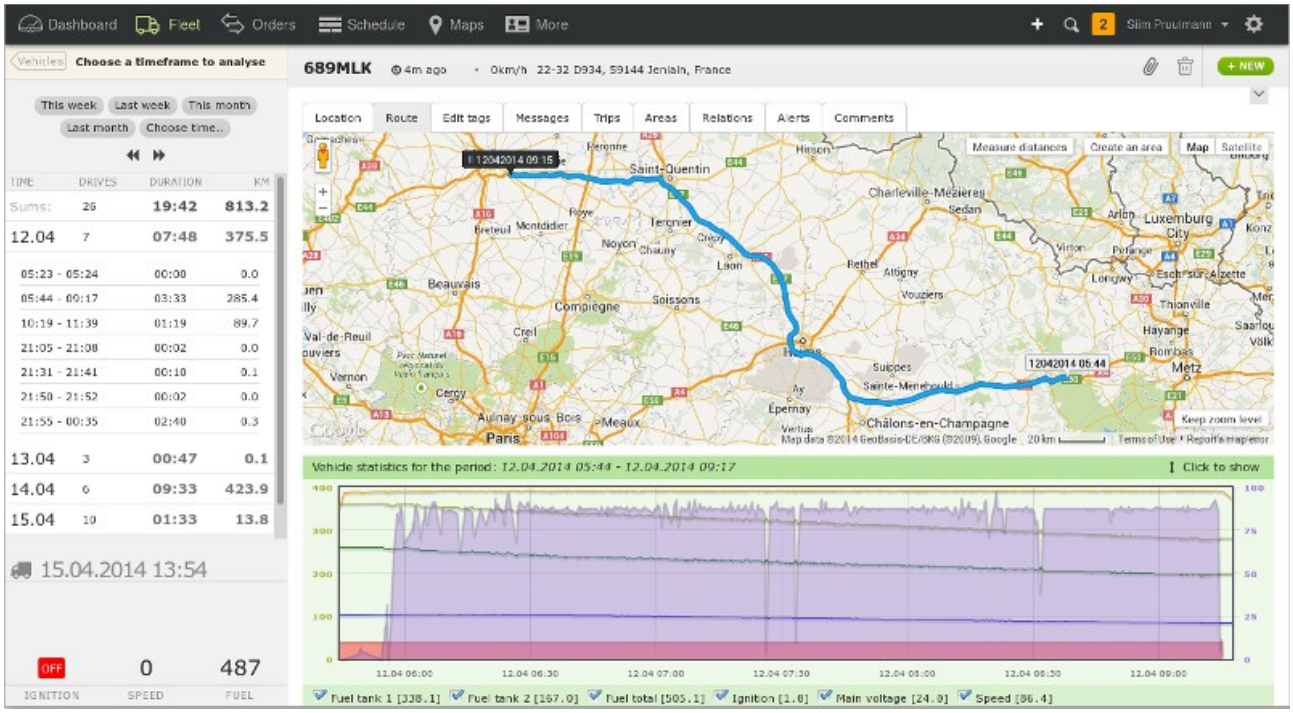
1.2 Praeguse kasutajaliidese probleemid

Peamine kasutajaliides on valdavas osas kasutatav väikestel puutetundlikel ekraanidel. Paraku ei ole see mugav. Leidub ka üksikuid vaateid, mida ei ole väiksematel telefoni ekraanidel (<4" diameetriga) võimalik efektiivselt kasutada. Enamik tahvelarvuteid on piisava suuruse ja resolutsiooniga (>= 1024px laiust) kogu kasutajaliidese kasutamiseks. Mida väiksem on ekraan, seda väiksem on kuvatav sisu. See muudab raskeks elementidele klikkimise. Originaalses kasutajaliideses on kohati väga palju klikitavaid elemente üksteisele küllaltki lähedal. Selline kooslus teeb rakenduse kasutamise väikestel ekraanidel väga keeruliseks. [4,5]

Teine sisuline probleem on kasutajaliidese sõltuvus internetiühendusest. [6] Iga liigutuse peale tehakse päringuid serverisse ja eeldatakse vastust. Kui serveriga ühendus puudub on rakendus põhimõtteliselt kasutamatu. Kasutaja poolne internetiühenduse katkemine on kontoris oleva tööarvuti puhul küllaltki vähetõenäoline. Mobiilse seadme puhul aga tavapärane. Internetiühenduse puudumist võib põhjustada maa-aluses parklas või suures rahvahulgas viibimine. Ka väga aeglase ühenduse (nt. EDGE [7]) võib kasutaja vaatenurgast lugeda ühenduse üldse puudumiseks.

2. Mobiilne kasutajaliides

Mobiilsed seadmed on oma füüsiliselt mõõtmetelt piiratud. Kõik, mis on võimalik suure ekraani peal ei ole teostatav väikesel ekraanil. Vähemalt mitte samal kujul. Keerulisemate vaadete esitamisel on ekraanil korraga väga palju informatsiooni [*vt. Pilt 2*]. Sellises olukorras on valik, kas üritada ikkagi kõike kuidagi esitada või võtta kõigest võimalikust ainult piiratud alamhulk esitamiseks.



Pilt 2 – Sõiduki liikumise ajaloo vaade

Esimene valik säilitab küll kogu funktsionaalsuse ja informatsiooni, aga võib tuua ka mitmeid negatiivseid külgi. Saab üritada kõike korraga ekraanile paigutada. Sellisel juhul tekib risk, et kuvatav informatsioon on liiga tihedalt koos. Puutetundlikud ekraanid, eriti sõrmedega juhtimiseks mõeldud, on väikese sisu korral küllaltki ebatäpsed [4,5]. Paigutades palju võimalikke tegevusi üksteisele väga lähedale, tekib ka palju eksimisvõimalusi. Mobiilsed seadmed pakuvad võimalusi kunstlikult laiendada pinda jättes osa vaadet servade taha ning kasutaja liigutuste peale õigeid asju kuvada [8]. Sellise lähenemise risk on, et kasutaja ei leia üles kavalalt paigutatud informatsiooni ja funktsionaalsusi.

Teine variant, näidata välja ainult alamosa kogu valikust, on antud olukorras oluliselt parem. Olulise funktsionaalsuse kaotamise riski vähendab fakt, et meil on teine keskkond, kus on kogu funktsionaalsus olemas. Antud rakendus ei ole mõeldud asendama kogu esialgset rakendust, vaid täiendama seda situatsioonides, kus esimene ei ole parim lahendus. Nüüd on küsimuseks, mis on antud rakenduse skoobis vajalik ja mis ei ole?

2.1 Peamised ideed

Esimeses peatükis on kirjeldatud LogistITi ülesehitust ja mis kujul andmeid hoitakse. Antud mobiilse rakenduse mõte on olla logistiku abivahend olukorras, kus ta ei ole arvuti lähedal või mõnel muul tingimusel on originaalse rakenduse kasutamine raskendatud või võimatu. Selgema pildi loomiseks on järgnevalt esitatud mõned selliste olukordade näited.

Näide 1

Logistik on parasjagu kontorist väljas ja tuleb telefonikõne, et ühe tema autoga on tekkinud probleem. Ta teab, et LogistITi süsteemis on neil kirjas ühe katki läinud autole lähedalasuva töökoja telefoninumber ja muu vajalik info.

Näide 2

Sama logistik on lõunal ja oma lauakaaslasega vesteldes ilmneb, et tollel oleks pakkuda üks töö. Ta saaks kohe süsteemi luua uue tellimuse.

Näide 3

Eelmist näidet jätkates kirjutaks meie logistik üles kohe ka oma lauakaaslase ja nüüd juba kliendi kontaktandmed.

Näide 4

Ka kontorist väljas viibides on võimalik kontrollida tellimuse hetkeseisu.

Esimesed kolm neist näidetest peaks olema toimivad ka puuduliku internetiühenduse olukorras. Näite *1* puhul on tegemist küllaltki staatilise iseloomuga andmetega. Pole eriti tõenäoline, et töökoja telefoninumber väga tihti muutub.
Näide *2* kirjeldab olukorda kus internetiühendus ei ole kohe vajalik. Andmed on võimalik salvestada lokaalselt ja serveriga sünkroniseerida internetiühenduse taastumisel. Sama kehtib näite *3* korral.

Neis situatsioonides on loomulikult oluline kasutajat teavitada, et tegemist ei pruugi olla kõige värskema informatsiooniga, või et loodud uus informatsioon on veel serverisse salvestamata. Sünkroniseerimine peaks toimuma esimesel võimalusel, ning ka aktiivsest vaatest lahkudes peaks rakendus juhtima kasutaja tähelepanu faktile, et osa tegevusi on veel lõpetamata.

Näide *4* sisaldab endas põhimõtteliselt värskeima informatsiooni pärimist, seega pole seal loogiline kasutada varasemalt talletatud informatsiooni.

2.1.1 Kontaktid

Üks peamisi kasutusjuhtumeid on kontaktide otsimine. Kontakti laadset infot kannavad endas kolm andmebaasis olevat tabelit: persoonid, alad ja ettevõtted. Mingil määral on juba praegu alustatud nende objektide kokkusulatamisega kasutajaliidese tasandil. Näiteks SMSi saatmisel sooritatakse kontaktiotsing üle kõigi nende kolme tabeli. Siin võiks seda viia veelgi edasi ja neid kolme kõiki lihtsuse nimel esitada Kontaktidena. Antud kontsepti saavutamine vajab arvestatavat tööd ka LogistITi põhisüsteemiga. Üks olulisemaid infokandjaid on telefoninumber ja neid hoitakse kõigi kolme puhul spetsiaalse täägi *Contact* sees. Kõigil kolmel mudelil on ka muid rolle süsteemis, aga need ei takista objekti samal ajal kontakt olemast. Ka kontakti loomine peab toimuma lihtsalt. Kiires olukorras ei peaks sundima kasutajat valima, kas ta tahab luua süsteemi uut ettevõtet või persooni,

selle valiku saab alati teha hiljem. Peamine on talletada kontaktinfo.

2.1.2 Märkmed

Märkmed pakuvad olulise võimaluse leevendada olukorda, kus mobiilne kasutajaliides ei ole võimeline tegema kõike, mida suudab tema suurem vend. Peamises rakenduses kuvatakse kasutajale märkmeid *dashboard*il, kus nad on kohe silme ees. Tehes mobiilsest liidesest märkmete lisamise võimalikult kergeks ja mugavaks saab kasutaja vähemalt kirja panna tegevuse, mida ta hetkel ei saa ükskõik, mis põhjusel teha. Oma tööarvuti taha tagasi jõudes on meeldetuletus kohe näha. Olemasoleva märkmete lahenduse tõttu on neid võimalik kirjutada ka teistele süsteemi kasutajatele, mis pakub võimaluse kasutajatel üksteisele teateid jätta.

2.1.3 Tellimuse loomine

Tellimuse sisestamine, kui üks põhitegevusi, peaks olema võimalikult lihtne ja kiire protsess. Süsteem nõuab praegu uue tellimuse tegemiseks kliendi ettevõtte olemasolu. Täieliku tellimuse jaoks on veel vaja lähte- ja sihtkohta, nende mõlema jaoks aegu ning tellimuse lisainfot, kui seda on. Potentsiaalselt on võimalik kiiruse nimel eemaldada isegi kliendi nõue, aga sellisel juhul taandub tellimuse loomine märkme loomiseks. Alternatiivina on võimalik määrata tellimuse kliendiks mõni muu ettevõte ja teha tellimuse juurde vastav märge. Tellimusel saab igal hetkel vahetada klienti.

Terviklike andmete nimel tasub kasutajat pigem suunata kõiki etteantud välju täitma. Aga säilima peaks ka võimalus sisestada informatsioon kiirelt. Ideaalis ei haaraks kasutaja kiires olukorras mitte paberi järele, vaid saaks kohe luua tellimuse süsteemi, kus sellega tööd jätkata.

2.1.3 Sõnumite saatmine

Kuivõrd keskkonnaks on kommunikatsioonivahend (nutitelefon), peaks kommunikatsioon ka uues kasutajaliideses kesksel kohal olema. Eelnevalt kirjeldatud kontaktide ühildamisega on põhi loodud. Sõnumivahetus (SMS, e-mail, potentsiaalselt veel midagi) peab toimuma sujuvalt. Vanas lahenduses on sõnumivahetus natuke konarlik. Sõnumeid esitatakse mitmes kohas ja erinevalt. Sõnumivaade võiks võimaldada kasutajal kiirelt saada sõnumi kohta rohkem konteksti. Näiteks kuvada sõnumi saatja kohta infot, kui seda on, pakkuda võimalust vaadata eelnevaid sõnumeid sama inimesega. SMSide puhul võiks siin ka internetiühenduse puudumisel pakkuda kasutajale telefoni SMS rakenduse kasutamist. Levi täielikul puudumisel ei pruugi muidugi ka see töötada.

2.1.4 Kiirelt ajas muutuv informatsioon

Päris palju informatsiooni on LogistITi süsteemis kiirelt ajas muutuv. Sellised on näiteks käimasolevad veod ja tellimused, hoiatused, sõidukite hetkeasukohad. Oma olemuselt pole sellist informatsiooni mõtet lokaalselt salvestada. Oluline on anda kasutajale asjakohane ja kiire tagasiside, kui pole võimalik esitada nõutud andmeid. Kiiruse nimel tuleks vähendada küsitavate andmete hulka. Näiteks vedude korral ei tohiks sama päringuga kaasa tulla planeeritud ja sõidetud tee informatsioon, mis võib olla mahult mitu megabaiti. Esialgu piisab kuvamisest, kui palju on kogu distantsist läbitud. Vaates võiks olla eraldi nupp, mille vajutamise peale küsitakse ka teekondade info.

3. Internetiühenduseta veebirakenduste loomise tehnoloogiad

Internetiühenduseta toimiva veebirakenduse jaoks on vaja mitmeid tehnoloogilisi lahendusi. Moodsamate veebirakenduste puhul on enamasti tegu ülesehitusega, kus loogika ja esitatavad andmed on rangelt eraldi. Loogika all saab siin mõista näiteks HTMLi, CSSi, JavaScripti ja ka disainielemente nagu pildid ja fondid. Andmeid hoitakse üldiselt andmebaasis ja rakendus saab neile ligi näiteks läbi API päringuid tehes.
Ka tehnoloogiad internetiühenduseta toimimise tagamiseks jagunevad samamoodi. Loogika jaoks on *ApplicationCache* [9], millest on pikemalt juttu peatükis 3.1. Andmete hoiustamine on kahjuks praegu veel küllaltki killustunud erinevate lahenduste vahel. *Web Storage* [10], *IndexedDB* [11] ja *WebSQL* [12] lahenduste ülevaated on vastavalt esitatud peatükkides 3.2 – 3.4. Peatükis 3.5 uuritakse *localForage*'i [13] nimelist lahendust kolme viimase andmete hoiustamise vahendi lõhestatuse vastu. Lõpetuseks uuritakse veel varajases arendusjärgus olevat *Service Workers* [14] spetsifikatsiooni.

3.1 ApplicationCache

ApplicationCache (lühendatud kui *AppCache*) on osa HTML5 standarditest. *AppCache* võimaldab veebirakendustel töötada ilma internetiühenduseta. Lihtsalt kokkuvõttes on arendajatel *AppCache*'i vahendite abil võimalik kirjeldada, millised ressursid peavad olema kättesaadavad ka ühenduse puudumisel. Brauser salvestab esmasel veebirakenduse laadimisel loetletud ressursid ja edaspidi on neid võimalik ilma ühenduseta kätte saada. [9]

AppCache'i kasutamisel on lisaks võrguühenduseta töötamise võimaldamisele ka teisi eeliseid. Ressursid salvestatakse kohalikku arvutisse, seega on nende kättesaamine kiirem,

kui üle võrgu pärimine. Väiksem koormus serverile – serverist laetakse alla ainult failid, mis ei ole *AppCache*'is.

Millised failid tuleks vahemällu salvestada ja millised mitte on kirjeldatud spetsiaalses vahemälu manifesti failis, millele viidatakse HTMLis.

3.1.1 Vahemälu manifesti fail

Tegemist on tekstifailiga, mis loetleb ressursid, mille brauser peaks antud rakenduse jaoks salvestama. Manifesti failile viidatakse HTML failist <html> elemendi seest atribuudiga *manifest.*

```
<!DOCTYPE HTML>
<html manifest="manifest.appcache">
...
</html>
```

Näide 3.1.1

Manifesti viite olemasolu HTML failis lisab automaatselt ka selle faili enda vahemällu paigutatavate failide hulka.

AppCache'i kasutava lehe (viidatakse manifesti failile) laadimisel toimub järgnev:

1. Kui veel ei ole rakenduse jaoks faile salvestatud (näiteks tegemist on esmase külastusega), laetakse veebirakendus ja manifestis kirjeldatud ressurssidest luuakse vahemälu.
2. Järgnevatel laadimistel võetakse manifestis kirjeldatud ressursid kohalikust vahemälust. Samuti küsitakse serverilt kas manifestifail on uuenenud.
3. Kui manifestifail on muutunud laetakse taustal alla kõik uues manifestis kirjeldatud ressursid ja luuakse uus vahemälu. Kuna selleks hetkeks on leht juba vanast

vahemälust laetud tuleb uue sisu nägemiseks dokument uuesti laadida (*refresh*).

Manifestis saab olla kolme tüüpi sektsioone: CACHE, NETWORK ja FALLBACK.

1. CACHE – see on vaikimisi sektsioon (kui sektsiooni pole kirjeldatud, eeldatakse seda). Siin kirjeldatud ressursid paigutab brauser vahemälusse ja loeb neid edaspidi sealt.
2. NETWORK – siin all olevad ressursid vajavad alati võrguühendust. Neid ei paigutata kunagi vahemällu.
3. FALLBACK – siin kirjeldatakse kaks viidet ressurssidele. Teist kasutatakse kui esimene ei ole kättesaadav.

```
CACHE MANIFEST
# v4 2014-05-02
index.html
styles/main.css
NETWORK:
*
```

Näide 3.1.2 – osa loodud prototüübi esialgsest AppCache'i manifestist

Märgiga # algavad read on kommentaarid. Salvestatud failide uuendamiseks on vajalik, et manifesti fail ka ise muutuks. Kui teha manifestis viidatud failis main.css muudatusi, tuleb muuta ka manifestifaili. Kõige lihtsam variant on pidada kommentaaris järge näiteks versioonist ja/või millal viimati uuendati faili. [15]

3.1.2 AppCache'i puudused

AppCache'l on omadusi ja kõrvalefekte, mis ei ole alati soovitud. [16] Potentsiaalsed probleemid on näiteks:

1. Manifestis loetletud ressursid tulevad alati vahemälust, isegi kui tegelikult on internetiühendus olemas. Nagu eelnevalt kirjeldatud, kontrollitakse pärast lehe laadimist kas manifesti fail on uuenenud. Selleks ajaks on aga rakendus juba laetud vahemälust ja kasutaja võib olla juba rakendust kasutamas, seetõttu ei saa automaatselt kohe lehte värskendada.

2. *AppCache* ise on lisavõimalus ressursside salvestamiseks. HTTP protokollil on samuti olemas meetodid kirjeldamaks kas faile võiks säilitada ja kui kaua. Mõningates olukordades võivad need instruktsioonid omavahel kokku põrgata ja tekitada ootamatu situatsiooni, kus *AppCache* ei käitu eelduste kohaselt.

3. Manifesti faili ennast ei tohiks mitte ühegi meetodiga lubada salvestada. Vastasel juhul võib tekkida olukord, kus vahemälust leitakse koguaeg sama vana manifesti-fail. Kuna see pole ilmselgelt muutunud ei üritata ka uut hankida.

4. Vahemälust loetud veebileht ei lae ressursse, mis ei ole vahemälus. Ehk kui vahemälust võetakse *index.html*, mis viitab *kass.png* pildile, aga see pilt ise ei ole vahemälus, siis antud pilti sellel lehel ei kuvata. Selle vastu aitab NETWORK sektsioonis kõige lubamine (märgitakse sümboliga tärn: „*”)

5. Tingimuslikud ressursid. Veebilehel võib olla mitu komplekti pilte (näiteks erinevate ekraaniresolutsioonide tarbeks), millest näiteks meediapäringute abil valitakse üks komplekt, mis alla laetakse. Manifestis peaks aga kõik pildid kõigist komplektidest kirjas olema. Kui nad seal kirjas on, siis nad ka manifesti uuenemisel kõik alla laetakse.

3.1.3 ApplicationCache'i tugi brauserites.

ApplicationCache on kirjeldatud probleemidest hoolimata peamine vahend internetiühenduseta töötavate veebirakenduste loomisel. *ApplicationCache* on laialt toetatud. [17] Hetkeseisuga on see implementeeritud kõigis suuremates stabiilsetes töölaua (Firefox 29, Google Chrome 34, Internet Explorer 11, Safari 7, Opera 20) ja mobiilsetes (iOS Safari 7, Android Browser 4.4, Blackberry Browser 10, IE Mobile 10) brauserites. Enamikes neist juba pikka aega, mis on võimaldanud *ApplicationCache*'l muutuda väga laialt levinuks.

ApplicationCache ei ole siiski suutnud kõiki vajadusi rahuldada ja seetõttu on asutud looma uusi alternatiivseid lähenemisi. Üks selline on näiteks *Service Workers*, millest on lähemalt juttu peatükis 3.6.

3.2 Web Storage

Web Storage on spetsifikatsioon, mis on loodud paremaks alternatiiviks küpsistesse informatsiooni salvestamisele. Informatsioon salvestatakse võti/väärtus paaridena. *Web Storage* kirjeldab kaks meetodit andmete salvestamiseks. [10]

Esimene on *sessionStorage*, mis lahendab olukorra, kus kasutaja võib mitmes aknas sama rakendust kasutada. Näiteks kahes eri aknas korraga lennupileteid osta. Küpsiseid kasutades võib informatsioon lekkida ühest aknast teise ja põhjustada näiteks samale lennule mitme pileti ostmist. *SessionStorage* kestab ühe seansi ja tühjendatakse brauseri sulgemisel.

Teine vahend on *localStorage*, mis on püsivam ja jääb alles ka pärast brauseriakna sulgemist ja taasavamist ning on kättesaadav kõigist akendest, kust sama veebileht lahti on. *LocalStorage* lahendab olukorra, kus on vaja salvestada mõõdukalt palju informat-

siooni. Küpsised ei ole sellises olukorras hea lahendus, sest neid liigutatakse koos iga päringuga. Nende mahutavus on samuti palju vähem kui *localStorage*'l.

3.2.1 Kasutamine

SessionStorage ja *localStorage* eksisteerivad brauseris *window* objekti alamobjektidena ja nende kasutamine käib täpselt ühtemoodi. Mõlemad on *Storage* objektid, mis on spetsifikatsioonis kirjeldatud näites 3.2.1 esitatud kujul.

```
interface Storage {
  readonly attribute unsigned long length;
  [IndexGetter] DOMString key(in unsigned long index);
  [NameGetter] DOMString getItem(in DOMString key);
  [NameSetter] void setItem(in DOMString key, in DOMString data);
  [NameDeleter] void removeItem(in DOMString key);
  void clear();
};
```

Näide 3.2.1

Web Storage'i suureks eeliseks teiste informatsiooni talletamise meetodite ees on tema lihtsus. Peamised meetodid *Storage* objektil on *getItem(key), setItem(key, value)* ja *removeItem(key).* Nende kolmega on kõik vajalik tehtav. Salvestamaks sisseloginud kasutaja nime piisab koodireast *window.localStorage.setItem('kasutajanimi', 'Jaan')*, kus 'kasutajanimi' on salvestatud väärtuse võtmeks ja 'Jaan' salvestatud väärtus ise. Kasutajanime saab nüüd kätte *window.localStorage.getItem('kasutajanimi')* kaudu. [18]

3.2.2 Kasutuspiirangud

Nagu igasugusel informatsiooni talletamisel on ka siin piirangud, kui palju on lubatud. Spetsifikatsioon jätab konkreetse numbri vabaks, öeldes et brauserid peaksid ise piirama, kui palju lubatud. Soovituslikult on piir 5MB ühe lähtekoha kohta. Brauserid võivad kasutajalt küsida luba rohkema ruumi kasutamiseks. [10]

Reaalsed mõõtmised on näidanud, et brauserite implementatsioonid erinevad. Üldises plaanis on töölaua brauseritel eraldatud 10MB ja mobiilsetel 5MB. Eksisteerib aga ka erandeid. Näiteks Android Browser 4.3 lubab *localStorage*'le ainult 2MB ja töölaua Safari 5MB. [19]

Nagu *ApplicationCache*'gi, on *Web Storage* implementeeritud sisuliselt kõigis moodsates veebibrauserites. Erandiks on ainult Opera Mini. [20]

3.3 IndexedDB

Suuremate andmehulkade puhul ei ole *Web Storage* enam piisav ei mahutavuselt ega efektiivsuselt. *IndexedDB* on transaktsiooniline andmebaasi süsteem just suurte stuktureeritud andmemahtude hoidmiseks ja nende peal indekseerimise abil kiirete otsingute tegemiseks. [11,21] *IndexedDB* spetsifikatsioon näeb ette nii sünkroonse kui asünkroonse API. Praegu eksisteerib brauserites ainult asünkroonne implementatsioon.

3.3.1 Kasutuspiirangud

IndexedDB on valitud standardiks millega edasi minna, aga standard ei ole veel kõigis brauserites implementeeritud. Spetsifikatsiooni ei ole implementeeritud Apple'i Safari brauserites ei töölaual ega iOSi peal ning Internet Exploreri implementatsioon on veel puudulik [22]. Mahulimiidid on oluliselt varieeruvamad ulatudes fikseeritud 5MBst mobiilse Firefoxi (50MB töölaual) peal kuni 10%-ni vabast kettaruumist Chrome'i puhul. Firefoxi

puhul limiite ületades küsitakse kasutajalt, kas ta on sellega nõus. Nõusoleku saamisel enam mahul piiri ei ole. Internet Explorer lubab vaikimisi 10MB, siis küsib kasutajalt ja kui luba antakse saab kasutada kuni 250MB. Internet Explorer suudab salvestada ka kuni 999MB, aga selleks peab kasutaja muutma brauseri seadeid. [19]

(Antud töö kirjutamise ajal veel ametlikult välja laskmata Safari 8 ja iOS Safari 8 versioonid juba toetavad IndexedDB spetsifikatsiooni.)

3.4 WebSQL

WebSQL spetsifitseerib API andmete paigutamiseks andmebaasi, millele saab päringuid teha variandiga SQL keelest. Novembris 2010 loobus W3C veebirakenduste töögrupp *WebSQL*i spetsifikatsiooniga edasi töötamast. Põhjuseks oli, et kõik, kes implementeerisid *WebSQL*i kasutasid taustal *Sqlite*'i, aga standardi väljatöötamiseks oleks vaja erinevaid lahendusi. Selle asemel keskendutakse *Web Storage*'i ja *IndexedDB* spetsifikatsioonidele. [12]

WebSQL võiks olla juba unustusse vajunud, aga Apple ei ole oma veebibrauseritel endiselt realiseerinud *IndexedDB* lahendust. *WebSQL* on Safari peal seega endiselt ainus valik, kui on vaja *Web Storage*'st võimekamat lahendust.

3.4.1 Kasutuspiirangud

WebSQL ei ole toetatud Firefoxi ja Internet Exploreri peal. [23] Mahupiirangute poolest kehtib Chrome'i peal sama loogika, mis *IndexedDB* puhulgi. Apple'i brauserite puhul on olukord keerulisem. Alates 5MB'st küsib Safari 10MB, 50MB, 100MB, 500MB, 600MB, 700MB... järel kasutajalt kas lubada ruumi kasutada. Safaril iOSi peal on maksimaalne maht 50MB. Salvestades rohkem kui 5MB küsitakse kasutajalt andmete talletamiseks luba. [23]

3.5 LocalForage

LocalForage on JavaScripti teek, mis ühtlustab andmete hoiustamist kliendi pool. [13] *LocalForage* kasutab taustal vastavalt kliendi võimekusele kas *IndexedDB*'d, *WebSQL*'i või *LocalStorage*'t kasutades selleks lihtsat *Web Storage*'i laadset APIt. Erinevalt viimasest on *localForage* asünkroonne ja võimaldab hoida igasuguseid andmeid. APIt saab kasutada kasutada kas *callback* stiilis või lubadustega (*Promise*).

API kasutamine on sisuliselt samamoodi kui *Web Storage*, mida on peatükis 3.2.1 kirjeldatud. Oluliseks vaheks on, et siin on kõik meetodid asünkroonsed, mis tähendab, et tagastusväärtuseks on neil *Promise* objekt, mitte leitud või kirjutatud väärtus. *Callback* stiilis kirjutades on igal meetodil üks lisaparameeter – funktsioon, mis käivitatakse, kui kõik õnnestus.

```
localforage.getItem('võti', function(väärtus) {
      alert(väärtus);
});
```

Näide 3.5.1 – callback stiilis localForage'i kasutamine

```
localforage.getItem('võti').then(function(väärtus) {
      alert(väärtus);
});
```

Näide 3.5.2 – lubaduste stiilis localForage'i kasutamine

LocalForage'i *setItem()* meetod lubab salvestada järgnevaid JavaScripti objekte: Array, ArrayBuffer, Blob, Float32Array, Float64Array, Int8Array, Int16Array, Int32Array, Number, Object, Uint8Array, Uint8ClampedArray, Uint16Array, Uint32Array, String. *LocalStorage* ja *WebSQL* mootoreid kasutades serialiseeritakse binaarkujul andmed enne salvestamist.

LocalForage võimaldab ka käsitsi valida, millist mootorit kasutada. Vaikimisi valitakse esimene olemasolev selles järjestuses: *IndexedDB, WebSQL, localStorage.* Mootori muutmiseks saab kasutada meetodit *setDriver('mootori_nimi').* Arenduse käigus võib olla mõistlik kasutada *localStorage*'t, kuivõrd selle sisu inspekteerimine on brauseri vahenditega küllaltki triviaalne.

3.6 Service Workers

Service Workers on uus lähenemine probleemile, mille lahendamisega alustas *Application-Cache.* Sisuliselt võimaldavad *Service Workerid* arendajal täpselt kontrollida kuidas kasutatakse vahemälu ja kuidas päringutele vastatakse - isegi kui võrguühendus puudub.

Seda sama üritab teha ka *ApplicationCache,* aga *AppCache*'i lähenemine on deklaratiivne – brauser saab manifestifaili ja juhtub maagia. Nagu peatükis 3.1.2 on kirjeldatud on *AppCache*'l mitmeid puudusi. *Service Worker* annab arendaja kätte kontrolli maagia üle.

```
navigator.serviceWorker.register('worker.js', { scope: '/*' }).then(
    successHandler,
    failureHandler
);
```

Näide 3.6.1 ServiceWorkeri lehele lisamine

Näites 3.6.1 oleva koodi peale üritab brauser alla laadida ja paigaldada *worker.js* skripti. Oluline on tähele panna, et *Service Worker* töötab praegu ainult üle HTTPS'i laetavatel lehtedel. Paigaldatud *Service Worker* hakkab tööle pärast järgmist lehe laadimist. Sellest hetkest alates käivad kõik edasised päringud, mis sobivad määratud skoobiga, läbi loodud *Service Workeri,* kus otsustatakse, mida teha ja milline vastus anda – isegi, kui puudub internetiühendus.

Service Worker sisaldab ka vahemälu mehhanismi, mis on ehk selgem näidete abil. Vt. näidet 3.6.2

```
this.addEventListener('install', function(e) {
      // loob ressurssidest vahemälu. Alustab nende alla laadimist
      var cachedResources = new Cache(
            'index.html',
            'style.css'
      );
      // oota, kuni kõik ressurssid on alla laetud
      e.waitUntil(cachedResources.ready());
      // lisa loodud vahemälu globaalselt kättesaadavasse objekti
      caches.set('cache1', cachedResources);
});
// iga päring jõuab siia.
this.addEventListener('fetch', function(e) {
      // e.request on url, mida päriti.
      // otsitakse vahemälust urlile vastav ressurss ja vastatakse
```

```
        // sellega.
        e.respondWith(caches.match(e.request));
});
```

Näide 3.6.2 – worker.js faili potentsiaalne sisu.

Sellega on juba võimalik ehitada internetiühenduseta töötavaid rakendusi. Enamasti on soovitav olukord, kus ühenduse olemasolul võetakse ressursid internetist ja alles siis, kui ühendus puudub pöördutakse kohaliku koopia poole. Seda on samuti lihtsam demonstreerida näitega. Näide 3.6.3 eeldab näites 3.6.2 loodud vahemälu.

```
this.addEventListener('fetch', function(e) {
        // kui päritakse index.html faili
        if (e.request.url === 'index.html') {
                e.respondWith(
                        // teeb tavalise päringu index.html saamiseks
                        fetch('index.html').then(
                                null,
                                function() {
                                        return caches.match('index.html');
                                }
                        )
                );
        }
```

```
});
```

Näide 3.6.3

Näide 3.6.3 kirjeldab situatsiooni, kus üritatakse saada üle võrgu *index.html* faili, kui see ebaõnnestub (näiteks ühenduse puudumise tõttu), siis otsitakse *index.html* üles vahemälust. Funktsioon *fetch()* tagastab *Promise* objekti. Viimase meetod *then()* võtab kaks funktsiooni parameetriteks, esimene käivitatakse kui *index.html* saadakse üle võrgu kätte, teine siis kui *index.html*i ei õnnestu üle võrgu saada. Esimene parameeter on siin näites *null*, seega tagastakse õnnestumise korral lihtsalt saadud *index.html* ise. Ebaõnnestumisel otsitakse kohalik koopia *index.hmtl*-ist. Selle koha peal annab teha veel spetsiifilisemalt, näiteks seada ajalimiidi, kui kaua üritatakse võrgust maksimaalselt ressurssi kätte saada enne alla andmist. [24, 25]

Service Workersil on potentsiaali anda arendajale tohutu kontroll rakenduste käitumise üle ühenduse puudumisel ja paljudel muudel juhtudel. Praegu on *ServiceWorkers* veel aktiivses arendusjärgus. *Service Worker* jõudis 08.05.2014 oma esimese World Wide Web Consortiumi spetsifikatsiooni mustandini. [14] Üheski brauseris ta siiski praegu veel ilma abivahenditeta täielikult ei tööta.

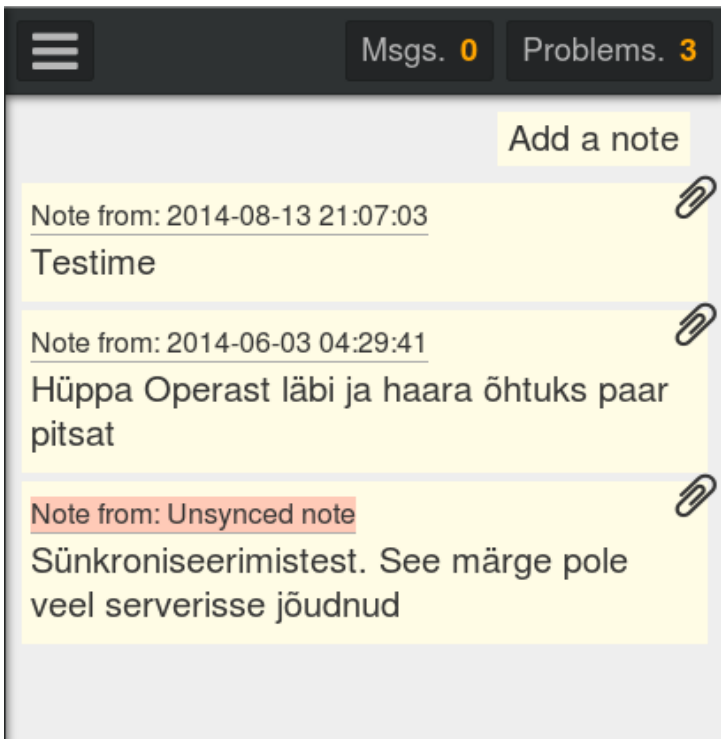
4. Valminud lahendus

Antud töö osana valminud prototüüp LogistITi mobiilse kasutajaliidese jaoks on projekt, mis on alles oma algusjärgus. Mitmed siin kirjeldatud ideed on leidnud oma esialgse kuju, aga mõned on pidanud ka edasi lükkuma, kuna on osutunud oodatust ulatuslikumaks ülesandeks.

Visuaalse lahenduse poolest on prototüübis üritatud säilitada peamise kasutajaliidesega sarnasust. Eelkõige aitab sellele kaasa ülemises ääres asuv kontrastne päis. Harjumuspäraselt on päise vasakus ääres ligipääs menüüle ja paremas ääres kiire ülevaade aktiivsete hoiatuste hulgast ja ligipääs neile. Samuti on sisse logimise järel esmaseks vaateks märkmed. Menüü on lahendatud vasaku ääre taha peituvana, kust ta libiseb esile, kas päises asuvale vastavale nupule vajutades või üle ekraani sõrmega vasakult paremale libistades. Menüü peaks tulevikus sisaldama endas ka üldist otsingut.

Funktsionaalsusest on prototüübis esindatud hoiatused, märkmed ja kontaktid. Hoiatuste kaudu on kättesaadavad ka sõnumid, kuivõrd LogistITi süsteem käsitleb ka süsteemi saabuvaid sõnumeid eriliiki hoiatustena. Märkmete ja kontaktide puhul on toetatud ka nende vaatamine ja loomine ka internetiühenduse puudumisel. Sellisel juhul uue märkme/kontakti puhul luuakse vastav objekt kõigepealt lokaalselt (on juba teiste hulgas kasutatav) ning internetiühenduse taastumisel salvestatakse kohe serverisse. Tähele tuleb panna, et sisse logimiseks peab seadmel olema internetiühendus.

Tehnoloogilisest küljest kasutab prototüüp internetiühenduseta toimimiseks eelmises peatükis kirjeldatud tehnoloogiatest *Application Cache*'i ja *Web Storage*'t. Peamine kasutuselolev keel on JavaScript.



Pilt 3 – Esilehe vaade, kui internetiühenduseta olekus on loodud uus märge, mis on veel serveriga sünkroniseerimata

Kokkuvõte

Käesoleva töö eesmärgiks on uurida mobiilsetele seadmetele suunatud kasutajaliidese lahendusi LogistIT süsteemi jaoks. Tingimustes, kus mobiilsed nutiseadmed on järjest levinumad ja võimekamad, ei ole võimalik neid ignoreerida. Olemasolev LogistITi rakendus teeb keskkondadele, kus ta töötab eeldusi, mis ei ole alati mobiilsetel seadmetel täidetud. Antud töö uuris, millised on need eeldused ja millises ulatuses on võimalik neid muuta tagamaks sujuv kasutajakogemus ka mobiilsetel seadmetel. Töö sisaldab endas ülevaadet olemasolevast taustsüsteemist. Kirjeldusi lahendustest, kuidas saada kasutajaliides toimima piiratud keskkondadel ja ülevaadet tehnoloogilistest lahendustest, mille abil eelnevalt kirjeldatut ellu viia. Samuti on valminud esialgne prototüüp, mis on aktiivselt edasi arendamisel.

Tähtajaks valminud prototüüp võimaldab kasutajal võtta vastu hoiatusi, sõnumeid (osana hoiatustest), märkmeid, sirvida kontakte. Samuti on võimaldatud uute märkmete ja kontaktide loomine, mis töötavad ka internetiühenduse puudumisel. Esimesel võimalusel sünkroniseeritakse uued loodud märkmed ja kontaktid serveriga. Peale esimest hankimist on andmed ka internetiühenduseta olekus saadaval.

Esimene ja teine peatükk seavad aluse jätkuvaks tööks, leidmaks veelgi paremaid ja kasutajasõbralikumaid lahendusi. Välja pakutud lahendused vajavad veel reaalsetes olukordades testimist ja täpsustamist. Kolmas peatükk on kogunud kokku ülevaate peamistest tehnoloogiatest, mille abil luua internetiühenduseta toimivaid veebirakendusi. Autori lootus on, et tehtud tööst leiavad potentsiaalseid lahendusi teisedki, kes seisavad silmitsi sarnaste probleemidega.

Mitmed ideed, mida töös uuritakse on laiendatavad tagasi peamisele kasutajaliidesele. Lahendused, mis aitavad rakendusel efektiivselt toimida mobiilsetel seadmetel võivad oluliselt parendada kasutajakogemust ka teistsugustel seadmetel.

Kirjandus

[1] The PostgreSQL Global Development Group, PostgreSQL, (Online) http://www.postgresql.org/ (11.05.2014)

[2] Python Software Foundation, Python, (Online) https://www.python.org/ (11.05.2014)

[3] Ecma-262 Edition 5.1, The ECMAScript Language Specification (2011)

[4] Lyza D. Gardner, Stephanie Rieger, Luke Wroblewski et al., „Vexing Viewports", *A List Apart* (ISSN: 1534-0295), No 367 (2012)

[5] Scott Kellum, „A Pixel Identity Crisis", *A List Apart* (ISSN: 1534-0295), No 342 (2012)

[6] Alex Feyerke, „Designing Offline-First Web Apps", *A List Apart* (ISSN: 1534-0295), No 386 (2013)

[7] Rysavy Research, „EDGE, HSPA and LTE The Mobile Broadband Advantage" (2007) lk 34

[8] Kyle Peatt, „Off The Beaten Canvas: Exploring The Potential Of The Off-Canvas Pattern", (Online) http://www.smashingmagazine.com/2014/02/24/off-the-beaten-canvas-exploring-the-potential-of-the-off-canvas-pattern/ (11.05.2014)

[9] World Wide Web Consortium, HTML5 specification, Offline web applications, (Online) http://www.w3.org/TR/2011/WD-html5-20110525/offline.html (11.05.2014)

[10] World Wide Web Consortium, Web Storage, (Online) http://www.w3.org/TR/webstorage/ (11.05.2014)

[11] World Wide Web Consortium, Indexed Database API, (Online) http://www.w3.org/TR/IndexedDB/ (11.05.2014)

[12] World Wide Web Consortium, Web SQL Database, (Online) http://www.w3.org/TR/webdatabase/ (11.05.2014)

[13] Mozilla Project, localForage, (Online) http://mozilla.github.io/localForage/ (11.05.2014)

[14] World Wide Web Consortium, Service Workers, (Online) http://www.w3.org/TR/2014/WD-service-workers-20140508/ (11.05.2014)

[15] Mozilla Developer Network, Using the application cache, (Online) https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache (11.05.2014)

[16] Jake Archibald, „Application Cache is a Douchebag", *A List Apart* (ISSN 1534-0295), No 350 (2012)

[17] Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers, Offline web applications, (Online) http://caniuse.com/#feat=offline-apps (11.05.2014)

[18] Mozilla Developer Network, Storage, (Online) https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage (11.05.2014)

[19] Eiji Kitamura, „Working with quota on mobile browsers", *HTML 5 Rocks*, (2014), (Online) http://www.html5rocks.com/en/tutorials/offline/quota-research/ (11.05.2014)

[20] Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers, Web Storage, (Online) http://caniuse.com/#feat=namevalue-storage (11.05.2014)

[21] Mozilla Developer Network, IndexedDB, (Online) https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (11.05.2014)

[22] Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers, IndexedDB, (Online) http://caniuse.com/#feat=indexeddb (11.05.2014)

[23] Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers, Web SQL Database, (Online) http://caniuse.com/#feat=sql-storage (11.05.2014)

[24] Alex Russell, „ServiceWorkers explained", *GitHub* (2014), (Online) https://github.com/slightlyoff/ServiceWorker/blob/master/explainer.md (11.05.2014)

[25] Jake Archibald, „Service Worker – first draft published", (Online) http://jakearchibald.com/2014/service-worker-first-draft/ (11.05.2014)

LogistIT User Interface for Mobile Devices

Bachelor's thesis (6 ECTS)

Siim Pruulmann

Summary

LogistIT is a service for transportation companies that merges fleet tracking and planning. It is built on a client-server model. The server provides an API, which can be used by different user interface solutions. Current user interface has several problems which make it difficult to use on mobile devices, especially smartphones. These problems fall into two main categories: issues with screen size and issues with network connectivity. The current user interface is built as a web application.

The aim of this thesis is to research creating a user interface tailored for use in smartphones. Thesis looks at the problems and tries to find solutions. It contains a description of the existing data model. Offers ideas on organising data in different ways to work better in limited environments and gives an overview of technologies that are used in creating offline web applications. A prototype showcasing the described approaches has been made and is in continuing development.

The first and second chapter of this thesis set the grounds for future work in making the LogistIT experience better on all devices and systems. The third chapter is in the form of introductory material to technologies for building offline web applications. It is the authors hope that others facing similar issues can find solutions or ideas in this thesis.

Lisa 1

Loodud prototüübi lähtekood on kättesaadaval avalikus GitHubi repositooriumis:

https://github.com/libahipi/logistit-mobile-ui

Elus näide on saadaval aadressil:

http://beta.logist.it/mobile-test/

NB! Repositooriumis olev kood võib olla uuem, kui näites kasutusel olev.

Lisa 2

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks.

Mina Siim Pruulmann (30.07.1989)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   LogistITi kasutajaliides mobiilsetele seadmetele,
   mille juhendaja on Vambola Leping,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace'i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

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

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

Tartus, 14.08.2014