TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Informaatikainstituut

Infosüsteemide õppetool

Andmebaasitehnoloogia valik mobiilseadmele Party Starter näitel

Magistritöö

Autor: Mart-Indrek Süld

Üliõpilaskood: 122372 IABM

Juhendaja: Ingmar Pappel

Tallinn 2014

Autorideklaratsioon

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

…… ……

(kuupäev) (lõputöö kaitsja allkiri)

Annotatsioon

Käesoleva magistritöö eesmärgiks on selgitada välja mobiilseadmete rakenduste arendamiseks sobilikeim andmebaasitehnoloogia. Tihtipeale kasutusel olevad andmehoidlad on puudulikud ja efektiivset arendust pärssivad. Nende alternatiivideks on vähetuntud ja populaarsust alles koguvad, kuid sobilikumad andmebaasitehnoloogiad.

Töö aluseks on autori poolt arendatav rakendus nimega Party Starter, mis pakub kasutajatele erinevaid situatsioone arvestavaid seltskonnamänge. Töös käsitletakse esmalt detailselt Party Starteri süsteemi, antakse ülevaade välja töötatud rakenduse lahendusest ning kirjeldatakse aplikatsiooni põhiosa täitvat arhitektuuri, et anda ülevaade rakenduse olemusest ja prototüübi arendusest. Töö teises pooles keskendutakse olemasolevate andmebaasitehnoloogiate uurimisele ja liigutakse edasi mobiilseadmetes kasutatavatele andmehoidlatele. Lugejale antakse ülevaade mobiilseadmete arendamises tavapäraselt kasutatava tehnoloogia puudujääkidest ning tutvustatakse alternatiivsetehnoloogiat NoSQL ja selle kasutamisega kaasnevaid võimalusi, mida seejärel rakenduse Party Starteri peal testitakse. Mobiilseadme jaoks osutus sobilikumaks us paindliku andmestruktuuriga NoSQL tehnoloogia, mille rakendamine muudab arendamise efektiivsemaks ja muudatustele vastuvõtlikumaks. Töö on koostatud ühe projekti näitel, kuid töö tulemused on rakendatavad väga laialdaselt.

Lõputöö on kirjutatud eesti keeles ja sisaldab teksti 68 leheküljel, 5 peatükki, 10 joonist, 19 koodinäidet ja 5 tabelit.

Annotation

The aim of this thesis is to identify the most suitable database technology option for the development of mobile applications. Insufficient database technologies are often used, what inhibit the effective development. Their alternatives and more suitable technologies are not well known and are yet to gain popularity.

The work is based on the application developed by the author called Party Starter, which offers their users parlor games, taking a variety of situations into account. At first, the thesis disserts the detailed description of the Party Starters system, provides an overview of currently developed application and describes the system's main core functional architecture to give an overview of the applications existence and prototype's development. The second part of this thesis focuses on describing the current database technologies and moves on to data repositories used in mobile applications development. Reader gets an overview covering the shortages of the technologies used in the development of mobile applications and an introduction to more advanced technology NoSQL with its usage possibilites which are then tested on the Party Starter application. The most suitable technology for mobile applications turned out to be NoSQL technology due to its flexible data structure, making the development more efficient and more adaptive to changes. Master's thesis is based on one project, but the results can be implemented widely.

This master's thesis is written in Estonian language and contains text on 68 pages, 5 chapters, 10 diagrams, 19 examples and 5 tables.

Lühendid ja mõisted

Käesoleva töö raames on kasutatud järgmisi alltoodud mõisteid kirjeldatud tähendustes

API	Rakendusliides ehk programmiliides on reeglistik olemasoleva valmisprogrammiga suhtlemiseks.
Dokument	Andmete kogum, mille sees asuvad võti-väärtus kollektsioonid.
Dünaamiline tüübiskeem	Muutuja väärtustamisel ei pea kirjeldama muutuja tüüpi.
Foursquare	Asukohapõhine sotsiaalvõrgustiku veebileht, mis on mõeldud mobiilseadmetele.
Freemium ärimudel	Ärimudel, kus minimalistlik toode pakutakse kasutajatele tasuta, kuid lisafunktsionaalsus võimaldatakse kasutajatele eraldi tasu eest.
GRASP muster	(General Responsibility Assignment Software Patterns) disainimustrid, mis kirjeldavad objektide ülesannete või vastutuste jaotamise fundamentaalseid printsiipe.
Initsialiseerima	Algväärtustama, andmeobjektile eluea algul andmeväärtust andma.
Iteraator	Liides andmekogumi elementide süstemaatiliseks läbikäimiseks.
JSON	kergekaaluline andmevahetusformaat, mis põhineb JavaScripti programmeerimiskeele alamhulgal. JSON on tekstiformaat ning sõltumatu programmeerimiskeelest.
Kategooria	Sarnaseid seltskonnamänge koondav andmete kogum.
Kirjeldus	Andmekogu, mis sisaldab endas seltskonnamängu detailset kirjeldust.
Kirjelduse eelvaade	Andmekogu, mis sisaldab endas seltskonnamängu kokkuvõtliku informatsiooni.
Klaster	Arvutisüsteemides nimetatakse klastriks serveritest ja muudest ressurssidest koosnevat rühma, mis funktsioneerib ühe tervikliku

	süsteemina ning võimaldab hõlpsat juurdepääsu ning mõnel juhul ka koormuse tasakaalustamist ja paralleeltöötlust.
Massiiv	Andmestruktuur, mis koosneb sama tüüpi elementidest ja on indekseeritav. Massiiv võib olla ühe- või mitmemõõtmeline.
Objekt	andmestruktuur, mis koosneb sarnastest andmeväljadest ning meetoditest.
Platvorm teenusena	Pilvandmetöötluse pakkumine kasutaja teenuse arenduseks ja osutamiseks teenusepakkuja määratud andmetöötluse platvormina.
Raamistik	Objektorienteeritud süsteemide puhul objektiklasside hulk, mis annab kasutajale või programmile omavahel seotud funktsioonide kollektsiooni.
Skaleerima	Seadme, objekti või süsteemi suurust muutma. Termin leiab laialdast kasutamist nii riistvara kui tarkvara laiendamisvõimaluste kirjeldamisel.
SQL	andmebaasi päringukeel, mis on loodud relatsioonbaasihaldurite jaoks.
Teek	Infoobjektide kogu.
Transaktsioon	järjestatud hulk SQL lauseid SQL-andmebaasis, mis täidetakse ühes või mitmes andmebaasis ja moodustavad ühe loogilise terviku.
Tugev tüübiskeem	Muutujate väärtustamisel peab kirjeldama muutuja tüüpi.

Joonised

Koodinäited

Tabelid

Sisukord

1. Sissejuhatus

1.1. Mis on Party Starter

Tänapäeva noorte inimeste koosviibimised on valdavalt igavad, sisutühjad ja ei julgusta kvaliteetsete suhete loomist. Taolised eesmärgipäratud koosviibimised on laialt levinud ja isegi kui inimesed on pikaajalised tuttavad, on õhtu muster reeglina ette teada. Internetis leidub küll väga palju informatsiooni, kuidas antud probleemile leevendust leida ja koosviibimised huvitavamaks ning meeldejäävamaks muuta, kuid kvaliteetset ja hetkekonteksti sobivat infot on raske ja tülikas leida.

Party Starter on rakendus, mille abil on võimalik leida inimesi ühendavaid seltskonnamänge vastavalt mängutüübile, mängijate arvule ja olemasolevale lisavarustusele. Läbi mängulise keskkonna saavad inimesed omavahel palju rohkem tuttavaks, neil tekib rohkem ühiseid mälestusi ja tugev vundament, millele rajada tulevikus võimalikult tugevad suhted.

Rakendus koosneb erinevate seltskonnamängude kategooriatest ja lisaks boonuskategooriatest, mis käsitlevad inimeste omavaheliste suhete parandamist. Kategooriad on loodud tutvumispõhimõtetest lähtuvalt ja grupeerivad sarnased mängud, mille seast on otsingukriteeriumeid kasutades võimalik leida endale sobivaim. Rakendus sisaldab endas rohkelt funktsionaalsust, kuid kasutajale on nähtav võimalikult minimalistlik kasutajaliides, kindlustamaks aplikatsiooni lihtne ja üheselt mõistetav kasutamine.

Rakenduse loomisel on lähtutud järgnevatest põhimõtetest:

- Lihtsus – sobivate mängude leidmine ja kasutamine peab olema võimalikult mugav ja lihtne
- Ligipääsetavus – ilma suure otsimise ja uurimiseta pakutakse kasutajale kvaliteetset informatsiooni
- Kasutusmugavus – kasutajaliides on konkreetne, lihtne ja puhas

Meeskond

Party Starteri meeskonna moodustavad igapäevaselt omavahel suhtlevad ja hästi läbisaavad inimesed, kes erinevad üksteisest nii tausta, hariduse kui ka kogemuste

poolest. Meeskonnaliikmete pädevusalad on erinevad, millest tulenevalt on kaetud nii äri- kui ka infotehnoloogia valdkond.

Party Starter sai alguse meeskonnaliikmete initsiatiivil, kes puutusid eelnevalt mainitud probleemiga isiklikult kokku ja samal ajal täheldasid ka lahendust, kuidas valupunkti lahendada. Sellest tulenevalt tekkiski idee koguda ja sorteerida võimalikult palju internetist leitavat informatsiooni, moodustada sellest terviklik kogum ja teha kättesaadavaks kõigile soovijatele.

Projekti algusfaasis otsustas meeskond nii ennast kui ka ideed proovile panna ning esmast tagasisidet saada. Võeti osa MEKTORY ärimudelite konkursilt, millel osalemine kujunes väga edukaks ja pääseti finaali 10 parima töö hulka. Ärimudelite konkursi finaalis pälviti teine koht, mille auhinnaks oli võimalus osaleda Euroopa Innovatsiooniakadeemia 2014 aasta suvekoolitusprogrammis Prantsusmaal, Nizzas.

1.2. Magistritöö eesmärgid ja meetodid

Käesoleva magistritöö probleemvaldkonnaks on sobiliku andmebaasitehnoloogia välja selgitamine mobiilseadmete rakenduste arendamiseks. Tänapäeval kasutatavad populaarsed tehnoloogiad pole piisavalt paindlikud ja alternatiivsed ning arendamiseks sobilikumad andmehoidlad on avalikkusele tundmatud ja vähe kasutatud. Töö eesmärgiks on selgitada välja relatsiooniliste ja NoSQL andmebaasisüsteemide tugevused ja nõrkused mobiilirakendustes, valida välja mobiilseadmete rakenduste arendamiseks kõige sobilikum andmebaasitehnoloogia ning rakendada valitud tehnoloogiat Party Starteri rakenduses ja testida selle sobivust.

Toodud eesmärkide saavutamiseks kirjeldatakse detailselt olemasolevat Party Starteri rakendust. Analüüsitakse relatsiooniliste ja NoSQL andmebaasisüsteemide tugevusi ja nõrkusi ning võrreldakse andmehoidlate kasutuseeliseid mobiilirakendustes. Seejärel rakendatakse valitud tehnoloogiat autori poolt arendatud Party Starteri süsteemile ja testitakse tehnoloogia sobivust rakendusega.

Andmebaasitehnoloogiate võrdlemisel toetutakse erinevate autorite nagu Anni Martin, Matt Asay, Michael Stonebraker, Neal Leavitt, Mudassir Malik ja Nicholas Greene poolt

välja toodud seisukohtadele. Kasutuseeliste välja toomisel on aluseks võetud nii teadusartiklid kui ka arvamusartiklid.

1.3. Ülevaade töö struktuurist

Magistritöö teises peatükis tuuakse välja Party Starteri süsteemi ülevaade ja kirjeldatakse käesolevat rakendust detailselt. Eesmärk on muuta rakendus omavahel seostatud hallatavateks osadeks, mis muudab aplikatsiooni kergemini mõistetavamaks ja võimaldab mõista lahenduse lähtepõhimõtteid.

Kolmandas peatüki eesmärgiks on selgitada relatsioonilise ja NoSQL andmebaasitehnoloogia tööpõhimõtteid. Tuuakse välja andmehoidlate tööpõhimõtted ning kirjeldatakse tehnoloogiate tugevusi ja nõrkusi.

Neljandas peatükis võrreldakse kolmandas peatükis väljatoodud andmebaasitehnoloogiaid mobiilseadmete rakenduste arenduses. Tuuakse välja eelised ja kitsaskohad ning sellest lähtuvalt valitakse välja sobilik tehnoloogia. Seejärel rakendatakse välja valitud tehnoloogiat Party Starteri süsteemis ja testitakse tehnoloogia kasutuselevõtmist ja sobilikkust. Lõpetuseks tuuakse välja Party Starteri edasiarendamise võimalused, mis uue tehnoloogia kasutuselevõtmisega kaasnevad.

Kokkuvõttes rõhutatakse tarkvaramustrite kasutamise ja sobiliku tehnoloogia rakendamise vajadust pidevalt muutumisjärgus olevate projektide puhul.

2. Party Starteri detailanalüüs

Käesolevas peatükis tuuakse välja Party Starteri süsteemi ülevaade, mis kirjeldab käesolevat rakendust detailselt, muudab kergemini mõistetavamaks ja jaotab omavahel seostatud hallatavateks osadeks.

2.1. Party Starteri süsteemiarendus

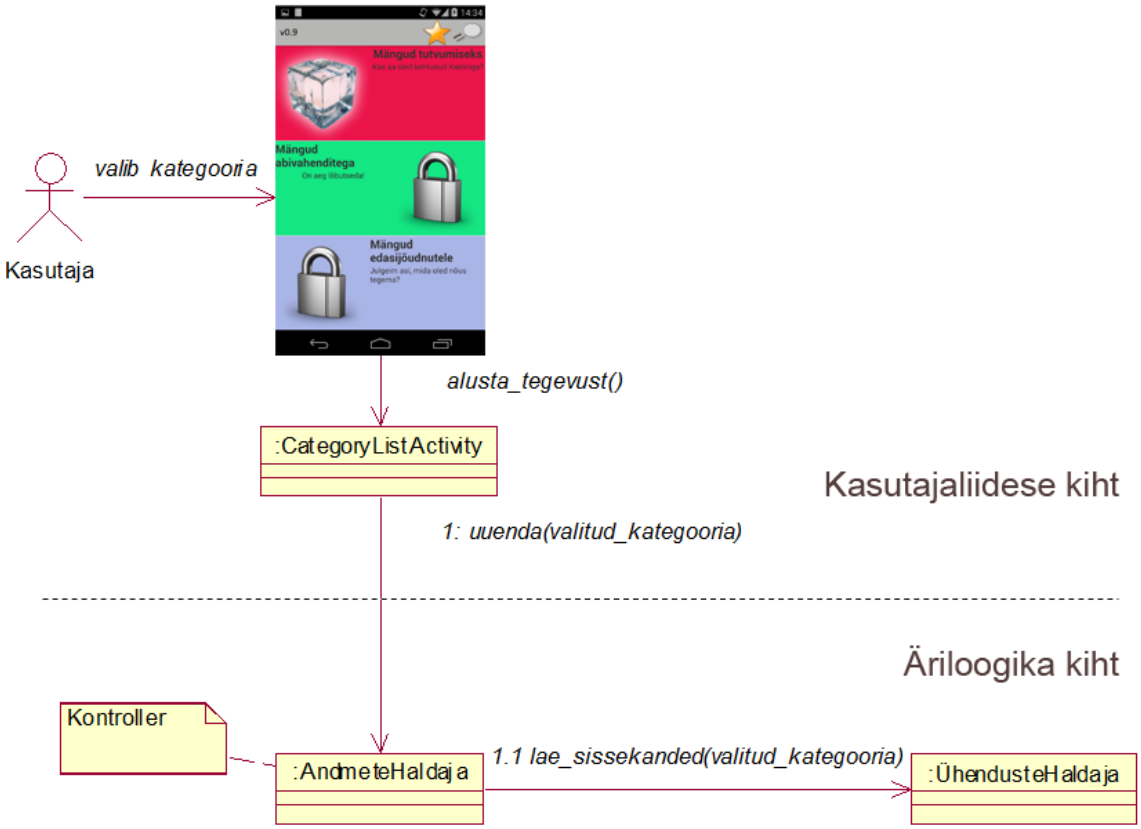
Party Starterist on valminud vähima funktsionaalsusega prototüüp (*minimum viable product*), mis on loodud Androidi operatsioonisüsteemi põhiselt ja rakenduse arendamisel on lähtutud objektorienteeritud disaini ja programmeerimise põhimõtetest. Andmete lugemiseks ja salvestamiseks on kasutatud populaarset ning kõige enim levinud relatsioonilise andmebaasi mootorit SQLite, mis on tuntud oma stabiilsuse poolest. (About SQLite, 2014).

Prototüübi arendamiseks kasutatud tehnoloogiad ei ole lõplikud ja kuuluvad vajadusel välja vahetamisesse. Hetkel on tehnoloogiate valimisel lähtutud arendajate subjektiivsetest valikutest ja ei pruugi olla mobiilsete rakenduste jaoks kõige sobivamad. Kasutatavad tehnoloogiad peavad vastama erinevatele nõuetele ja olema konkreetseid vajadusi arvesse võtvad, mille valimise ülevaade tuuakse välja peatükis „Tehnoloogia valimise üldised põhimõtted“.

2.1.1. GRASP muster – kontroller

Lähtudes tarkvaraarenduse headest tavadest, on Party Starteri loomisel kasutatud GRASP Kontrolleri mustrit, mille rakendamisel eraldatakse üksteisest kasutajaliidese- ja äriloogika kihid. Kontroller (GRASP *controller*) mustri kasutamisel luuakse kasutajaliidest eraldiseisev objekt, mis esindab terviksüsteemi. Kontroller ei ole osa kasutajaliidesest, vaid vastutab süsteemi tasemel toimuvate sündmuste vastuvõtmise, käsitlemise ja käskude edasi delegeerimise eest teistele süsteemiobjektidele (GRASP patterns, 2012).

GRASP kontrolleri toimimise põhimõtet illustreerib joonis 1.



Joonis 1. Kasutajaliidese ja äriloogika disain

Kõige lihtsam kood, kutsumaks Kontrollerit välja kasutajaliidesest, võiks välja näha järgnev:

```
DataManager dm = DataManager.getInstance(this);
```

Koodinäide 1. Kontrolleri kutsumine kasutajaliidesest

Antud koodireaga kutsutakse välja süsteemi esindav objekt, mis võimaldab ligipääsu peidetud süsteemioperatsioonidele.

2.1.2. SQLite Andmebaas

SQLite on avatud lähtekoodiga andmebaas, mis toetab relatsioonilise andmebaasi standardseid funktsioone nagu SQL süntaks, tehingud ja ettevalmistus käsud. Andmebaas suudab töötamise ajal hakkama saada umbes 250 kilobaidiga, mis teeb temast käitusfaasis kasutamiseks väga hea kandidaadi. SQLite toetab tekstil (`TEXT`), täisarvudel (*INTEGER*) ja reaalarvudel (`REAL`) põhinevaid andmetüüpe. Kasutamaks teisi tüüpe, tuleb nad enne

andmebaasi salvestamist konverteerida üheks eelnevalt mainitud tüübiks. SQLite andmetüüpide valideerimist läbi ei vii (SQLite and Android, 2013).

SQLite Android

SQLite on kättesaadav igal Androidi kasutaval seadmel, mille tõttu otsustati antud lahendust Party Starterit arendades kasutada. SQLite kasutamiseks ei ole nõutud kasutajapoolne eelseadistamine või andmebaasi administreerimine. Defineerida tuleb ainult SQL lausendid loomaks ja uuendamaks andmebaasi, peale mida uuendatakse andmeid Android platvormi poolt automaatselt. SQLite ei vaja opereerimiseks eraldi serveri protsessi või süsteemi, sest salvestamiseks kasutatavaid faile redigeeritakse otse teegi siseselt. See tähendab seda, et vahelüli on välja jäetud ja kolmanda osapoole protsesside seadistamine ei ole vajalik. Kuna ligipääsu andmebaasile tähendab ligipääsu failisüsteemile, peab andmete uuendamine toimuma asünkroonselt (SQLite and Android, 2013).

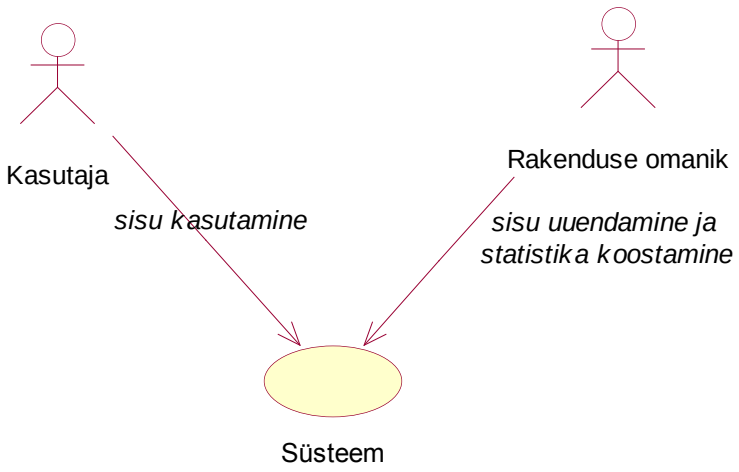
Tabelite detailsed kirjeldused on välja toodud punktis Tabelite detailsed kirjeldused.

2.2. Rollide ülevaade

Erinevates süsteemides on igal kasutajal roll ja sellega kaasnev vastutus, mille eesmärk on luua lihtsustatud ülevaade süsteemi toimimisest. Andmaks arusaadav ülevaade süsteemi kirjeldusest, on järgnevalt välja toodud süsteemis olevad osapooled ja tegutsejad:

- Kasutaja – vajab või kasutab süsteemi poolt teostatavaid või toetatavaid tegevusi
- Rakenduse omanik – pakub või täidab süsteemi poolt teostatavaid või toetatavaid tegevusi

Joonisel 2 on kirjeldatud Party Starteri süsteemi kasutajate peamised rollid, koos tegevustega:



Joonis 2. Süsteemi rollid

2.3. Funktsionaalsus

Kontseptuaalsed lausendid

Kirjeldamaks Party Starteri funktsionaalsust on välja toodud käesoleva töö skoopi kuuluv nimekiri lühilausetest, mis väljendavad loogilisi seoseid süsteemis läbiviidavate toimingutega. Lausendite eesmärk on väljendada nii seoseid kui ka iseloomustada süsteemis kasutatavaid põhimõisteid ehk põhiobjekte. Lausendid on välja toodud

loogilises järjestuses – põhilisemad mõisted ja seosed, iseloomustused enne, seejärel täiendavad:

- Süsteem genereerib dünaamiliselt menüüs kuvatavad elemendid
- Süsteem tagab kasutajale vaikimisi seltskonnamängud rakenduse paigaldamisel
- Süsteem genereerib dünaamiliselt kategooriate nimekirjad
- Süsteem genereerib dünaamiliselt seltskonnamängude nimekirjad
- Süsteemi genereerib dünaamiliselt seltskonnamängu kirjelduse
- Süsteem võimaldab otsida nime, formatsiooni, varustuse ja kategooriast lähtuvalt
- Süsteem koostab kasutaja tegevuse kohta statistikat
- Süsteem võimaldab kasutajal seltskonnamänge hinnata
- Süsteem võimaldab sirvida hinnanguga seltskonnamänge
- Süsteemi hinnatud seltskonnamängude nimekirja suurus on piiramatu
- Süsteemi käivitades uuendatakse vajadusel rakenduse andmebaas
- Süsteem kuvab kasutajale reklaamlahendusi
- Süsteem tagab seltskonnamängude unikaalsuse
- Süsteem võimaldab kasutajale mugava navigeerimise
- Süsteem võimaldab kasutajal osta lisapakette
- Süsteem tagab ostetud ja ostmata seltskonnamängude visuaalse eristatavuse
- Süsteem kuvab hinnangute nimekirja vaates teiste kasutajate TOP10 nimekirja
- Süsteem on kasutatav ilma välisühenduse olemasoluta
- Süsteemi käivitades kontrollitakse välisühenduse olemasolu
    - Välisühenduse olemasolul uuendatakse globaalset lemmikute nimekirja
    - Välisühenduse olemasolul laetakse üles kasutaja lemmikute nimekiri
    - Välisühenduse olemasolul laadima koostatud statistikat süsteemi omanikule
- Süsteem kuvab kasutajale rakenduse avalehel lisamenüü
- Süsteem võimaldab seltskonnamängu mängimise teavitamist levinud sotsiaalmeediates

Süsteemis toimuvad sündmused

Kontseptuaalsetest lausungitest lähtuvalt koostatud Party Starteri süsteemis toimuvad peamised protsessid, millele reageerimise peab süsteem tagama. Süsteemi peamised

protsessid on tuletatud lausungitest lähtuvalt ja annavad ülevaade aset leidvast sündmusest, teda kasutavast kasutajast ja süsteemi reageerimisest sündmusele:

Sündmus	Tegutsejad	Süsteemi tegevus
Rakenduse paigaldamine	Kasutaja	Süsteem loob andmebaasi ja initsialiseerib andmebaasi vaikimisi väärtustega
Seltskonnamängude uuendamine	Kasutaja, Rakenduse omanik	Süsteem kontrollib rakenduse käivitumisel välisühendust ja andmebaasi uuendamise vajadust. Kui on uuendusi, uuendatakse andmebaasis olevad vanad seltskonnamängud uutega.
Statistika üleslaadimine	Kasutaja, Rakenduse omanik	Süsteem kontrollib rakenduse käivitumisel välisühendust ja saadab Rakenduse omaniku poolt määratud lahendust kasutades kogutud statistika üleslaadimisele.
Teiste kasutajate hinnangute laadimine	Kasutaja, Rakenduse omanik	Süsteem kontrollib rakenduse käivitumisel välisühendust ja laeb Rakenduse omaniku poolt määratud lahendust kasutades teiste kasutajate hinnangute nimekirja.
Statistika kogumine	Kasutaja	Süsteem kogub jooksvalt kasutaja tegevuste kohta statistikat. Statistikat kogutakse selleks, et parandada sisu ja muuta kogu rakendus kasutajale atraktiivsemaks. Lisaks kogutakse infot kasutaja reageerimistest kuvatavatele reklaamidele.
Kategooriate nimekirja kuvamine	Kasutaja	Rakenduse käivitamisel kuvatakse kasutajale kategooriate nimekiri.
Kategooria seltskonnamängude nimekirja kuvamine	Kasutaja	Süsteem genereerib kasutajale valitud kategooria alla kuuluvate seltskonnamängude vaate.

Seltskonnamängu kuvamine	Kasutaja	Kasutaja valib mängude nimekirjast mängu, peale mida kuvab süsteem detailse kirjelduse seltskonnamängu mängimiseks.
Seltskonnamängude soovitamine	Kasutaja	Kasutaja on avanud seltskonnamängu kirjelduse vaate ja süsteem soovitab talle sarnaseid seltskonnamänge.
Seltskonnamängu hindamine	Kasutaja, Rakenduse omanik	Süsteem võimaldab kasutajal hinnata seltskonnamänge 10-punkti skaalal. Rakenduse poolt kogutud andmeid kasutab Rakenduse omanik kõige kõrgemini hinnatud seltskonnamängude nimekirja koostamiseks.
Hinnatud seltskonnamängude nimekirja kuvamine	Kasutaja	Süsteem kuvab kasutajale hinnatud seltskonnamängude nimekirja. Lisaks lokaalset hinnatud mängude nimekirjale kuvatakse kasutajale ka teiste kasutajate TOP 10 nimekiri.
Seltskonnamängu eemaldamine hinnatud mängude nimekirjast	Kasutaja	Süsteem kuvab kasutajale hinnatud mängude nimekirja vaatamise lehe. Mängu eemaldamiseks nimekirjast hoiab kasutaja soovitud mängul näppu peal, mis eemaldatakse lühikese aja möödudes süsteemi poolt lemmikute nimekirjast või kasutaja eemaldab hinnangu seltskonnamängu kirjeldavast vaatest.
Otsingu sooritamine	Kasutaja	Süsteem kuvab kasutajale otsingu täpsustamiseks erinevad parameetrid. Kasutaja täidab need soovile vastavalt ja süsteem kuvab kasutajale seltskonnamängud, mis vastavad otsingukriteeriumitele.
Lisamenüü kuvamine	Kasutaja	Süsteem kuvab kasutajale lisamenüü, milles on võimalik kerge vaevaga minna rakenduse kodulehele või lugeda infot rakenduse kohta.

Reklaami kuvamine	Kasutaja, Rakenduse omanik	Süsteem kuvab kasutajale seltskonnamängu kirjeldust vaadates reklaami. Reklaami ei näidata iga kord.
Sotsiaalmeedia teavitamine seltskonnamängu mängimisest	Kasutaja	Kasutaja on avanud seltskonnamängu kirjelduse vaate. Peale sotsiaalmeedia ikoonile vajutamist teavitab süsteem valitud sotsiaalmeediat seltskonnamängu mängimisest.

Tabel 1. Süsteemis toimuvad sündmused

2.4. Andmevaade

Käesolevas peatükis antakse ülevaade objektidest, mis on Party Starteris aset leidvate protsesside sisenditeks ja väljunditeks. Lisaks määratletakse mõisted, mida rakenduses kasutatakse ja antakse alus andmehoidlates säilitavate andmeobjektide määratlemiseks. Tulemuseks on süsteemi mõistete kirjeldused ja seletused ning süsteemi mõisteid kajastav kontseptuaalmudel.

2.4.1. Põhiobjektid

Peale süsteemis toimuvate sündmuste defineerimist on seejärel koostatud põhiobjektide nimekiri, mille kirjeldamisel on välja toodud sündmustega seonduvad teadmised, kasutamised, vastutused või omadused. Põhiobjekti kirjeldamisel antakse ülevaade objekti osaks olevatest atribuutidest või andme seostest teiste põhiobjektidega, millistel eesmärkidel antud objekti kasutatakse, milliseid süsteemi ülesandeid põhiobjektidega teostatakse või mis iseloomustab täiendavalt antud põhiobjekti.

Tabelis 2 on välja toodud süsteemis olevad põhiobjektid koos objektide kirjeldustega:

Objekti nimetus	Objekti kirjeldus
DataManager	Andmete vahendaja, mis esindab süsteemioperatsioone ja delegeerib käsklusi edasi peidetud objektidele. Süsteemi haldja.
LocalConnectionManager	Vastutab seadmes oleva andmebaasi andmete salvestamise ja lugemise eest.
ExternalConnectionManager	Vastutab väliste ühenduste loomise, andmete üleslaadimise ja allalaadimise eest.
Advertisement	Reklaamide haldamine.
UserInfo	Salvestab endas kasutaja tegevusest lähtuvalt kogutud statistikat.
Description	Seltskonnamängu kirjeldus.
DescriptionView	Seltskonnamängu vaade nimekirjas.
Category	Kategooria, mis koondab enda alla olemuselt sarnased seltskonnamängud.

Tabel 2. Süsteemi põhiobjektid

2.4.2. Tabelite detailsed kirjeldused

Põhimõistetest lähtuvalt ja süsteemi protsesside toetamiseks on andmehoidlates erinevate andmeobjektide väärtuste säilitamiseks koostatud tabelid. Tabeli atribuudid tähistavad objekti erinevate omaduste väärtuseid ja iga tabelis sisalduv komponent ehk veerg salvestab objekti ühe omaduse.

Andmete salvestamiseks ning süsteemi tööprotsesside toetamiseks on kasutusel kolm erinevat tabelit, mis jagunevad järgnevalt:

Tabel Category	
Atribuut	**Atribuudi kirjeldus**
_id	Primaarvõti.
name	Kategooria nimetus.
image_id	Kategooriat illustreeriva pildi nimetus.
description	Kategooriat kirjeldav tekst
is_bought	Tasulise kategooria sisu olemasolu iseloomustav tõeväärtus (True = sisu on ostetud, False = sisu ei ole ostetud).

Tabel 3. Kategooria tabeli kirjeldus

Tabel Description	
Atribuut	**Atribuudi kirjeldus**
_id	Primaarvõti.
category_id	Võti, mis seob seltskonnamängu kategooriaga.
name	Seltskonnamängu nimi.
number_of_players	Mängijate arv.
description	Seltskonnamängu kirjeldus.
image_id	Seltskonnamängu illustreeriv pilt.
is_bought	Tasulise seltskonnamängu olemasolu iseloomustav tõeväärtus (True = mäng on ostetud, False = mäng ei ole ostetud).
short_description	Mängude nimekirjas kuvatav lühikokkuvõte mängust.

formation	Suurus, mis määrab mängitava mängu formatsiooni. Formatsioon võib olla individuaalne, paari - või tiimi põhine.
equipment	Mängu mängimiseks vajaliku varustuse nimekiri.
rating	Arv, mis näitab kasutajapoolt antud hinnangut seltskonnamängule. Vahemik 1-10.
view_count	Arv, mitu korda mängu on mängitud. Suureneb automaatselt mängu kirjelduse lugemise korral. Kasutatakse populaarsete seltskonnamängude välja selgitamiseks.

Tabel 4. Seltskonnamängu tabeli kirjeldus

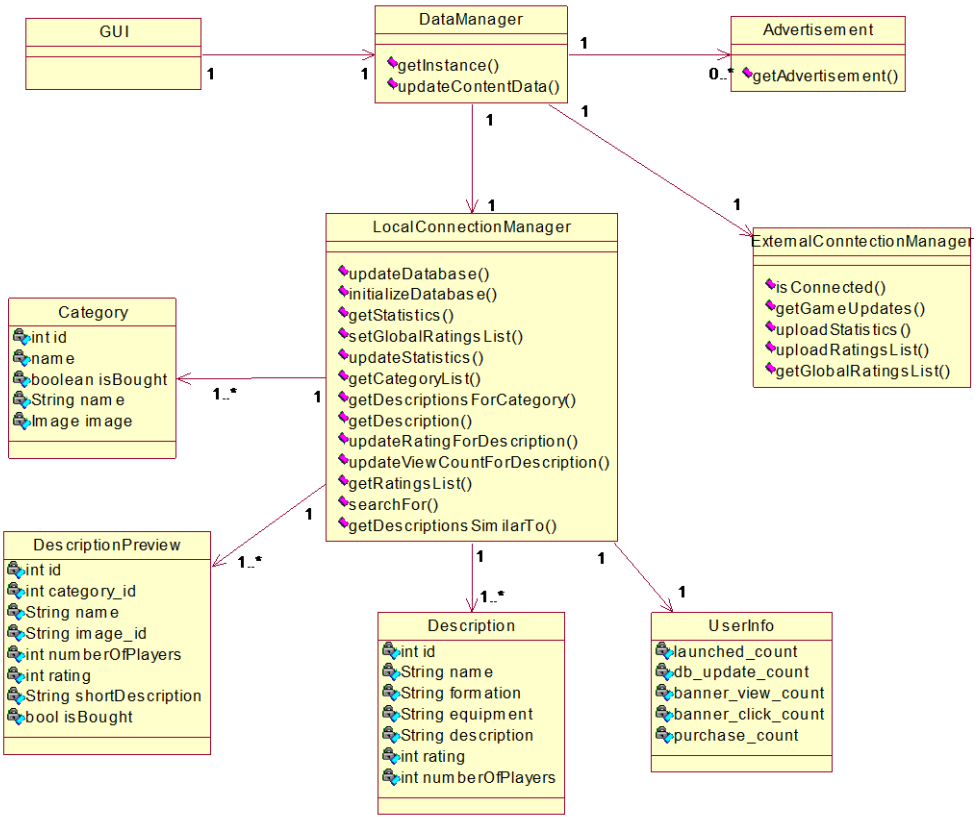
Tabel Userinfo	
Atribuut	**Atribuudi kirjeldus**
launched_count	Arv, mitu korda rakendust on käivitatud.
db_update_count	Arv, mitu korda andmebaasi on uuendatud.
banner_view_count	Arv, mitu korda kasutajale on reklaami näidatud.
banner_click_count	Arv, mitu korda kasutaja on reklaamile vajutanud.
purchase_count	Arv, mitu korda kasutaja on tasulist sisu ostnud.

Tabel 5. Kasutajainfo tabeli kirjeldus

2.4.3. Ülevaatlik disaini klassidiagramm

Eelkirjeldatud lausungid, süsteemi sündmused ja põhiobjektid on järgnevalt välja toodud realisatsiooni klassidiagrammina. Disaini klassidiagramm annab ülevaate eelkirjeldatud elementidest, klassides kasutatavatest põhiatribuutidest, peamistest operatsioonidest ning klasside omavahelistest seostest.

Diagramm on välja toodud joonisel 3 ning selle loomiseks on kasutatud Rational Rose tarkvara.



Joonis 3. Rakenduse disaini klassidiagramm

Käesolev klassidiagramm põhineb GRASP Kontroller mustril. Olulisimaks objektis on `DataManager`, mis esindab peidetud süsteemi operatsioone ja vahendab andmestikku kasutajaliidesega. `DataManager` on seotud põhiliste süsteemiobjektidega nagu `LocalConnectionManager`, `ExternalConnectionManager` ja `Advertisement`. Advertisement hoolitseb reklaamidega seotud andmete haldamise eest. `LocalConnectionManager` vastutab seadmes salvestatava informatsiooni haldamise ja kättesaadavuse eest. Objekt on vahelüli ja tagab ligipääsu andmebaasi salvestatud kategooriatele, seltskonnamängudele ja kasutajapoolt genereeritud informatsioonile. `LocalConnectionManager` on loojaks `Category`-, `DescriptionView`-, `Description`- ja `UserInfo`-objektidele. `Category` omab teadmisi kategooriaga seotud andmetest. `DescriptionPreview` on `Description` minimalistlik variant ja mõlemad omavad teadmisi seltskonnamänguga seotud

informatsioonist. `UserInfo` salvestab endas andmeid tegevustest, mida rakenduse kasutaja oma tegevustega jälgida võimaldab nagu reklaamidele vajutamiste arvud. `ExternalConnectionManager` vastutab välisühenduste loomise ja infovahetamise ning sünkroniseerimise eest välisserveritega.

3. Andmebaasitehnoloogiad

3.1. Tehnoloogia valimise üldised põhimõtted

Tarkvaraarenduses mõeldakse kasutatavate tehnoloogiate all koodi või riistvara moodulite kogumit, mida kasutatakse süsteemi järkjärguliseks ehitamiseks üldisest spetsiifiliseks. Tarkvara arendamisel puudutab tehnoloogia valimine eelkõige programmeerimiskeele, standardsete teekide, raamistike ja alamsüsteemide valimist.

Peamised faktorid, millega tuleb tehnoloogia valimisel üldiselt arvestada (Which technology stack should I choose to build my product, 2012):

- Produktiivsuse mõõtmine ja turule tulekuks kuluv aeg - teatud tehnoloogiatel on süsteemi algusfaasides produktiivsuse eelised. Viimasel ajal on esile tõusnud dünaamilise tüübiskeemiga keeled nagu Javascript, PHP, Python ja Ruby. Antud keelte omandamise barjäär on madalam, sest leidub rohkelt näidiseid, mida saab levinud probleemide lahendamiseks kiirelt kasutada.
- Inseneride talendikus - erinevatel programmeerijate generatsioonidel on oma eelis programmeerimiskeeled, mis antud grupi poolt kõige rohkem toetatud on. Sellest tulenevalt on oluline läbi mõelda, millise tööjõu puudus on. Kas vajatakse näiteks suurettevõtte arendajat või veebiskriptide kirjutajat. Valiku langetamisel tuleb mõelda eelkõige toote nõuetest lähtuvalt.
- Hooldatavus - iga tehnoloogia toetab kindlat stiili programmeerimist nagu näiteks objektorienteeritud- ja funktsionaalne programmeerimine või dünaamiline tüübiskeem ja tugev tüübiskeem. Tugeva tüübiskeemiga programmeerimiskeeled on toetatud paremate tööriistadega ja toetatud rohkem ettevõtete poolt, kus olulisel kohal on koodi suurus, arendajate arv ja teenuse alalhoidmine.
- Skaleeritavus - tehnoloogia valimise kontekstis tähendab skaleeritavus rohkemate klientide ja päringute teenindamist stabiilse reaktsiooniaja ja maksumusega. Skaleeritavus võib olla nii *scale-out* (rohkem masinaid) kui ka *scale-up* (rohkem tehteid ühe arvuti kohta).
- Keskkond - arendused erinevatele platvormidele toovad kaasa ranged piirangud. Spetsiifilisele platvormile toodet arendav ettevõtte võib olla sunnitud kasutama arendamiseks spetsiifilist tehnoloogiat. Soovi korral võib kasutada alternatiivset platvorm teenusega võimalust.

- Kommuun - aktiivne kommuun tagab programmivigade kiire parandamise ja spetsiifiliste vajaduste toetamiseks informatsiooni.
- Litsentsid - suure osa tänapäeva tarkvarast saab üles ehitada avatud lähtekoodiga, kuid oluline on olla teadlik erinevatest piirangutest, mis kaasnevad erinevate tehnoloogiatega.

Aitamaks langetada otsuseid tehnoloogiate valimisel, on asjatundjate poolt antud ka rida soovitusi, mida järgima peaks (Choosing your Realtime Web App Tech Stack, 2013):

1. Kasuta eksisteerivat lahendust, sest sellisel juhul kuluvad ressursid eelkõige rakenduse arendamisele.
2. Kasuta tuttavaid programmeerimiskeeli, sest võib tekkida vajadus viia sisse süsteemseid koodimuutusi.
3. Kasuta mobiilseadmete poolt toetatud tehnoloogiaid.
4. Täpsusta rakenduse funktsionaalsus, sest teades rakendusele seatud nõudmisi, on kergem valida kasutatavaid vahendeid.
5. Täpsusta rakenduse küpsusaste. Valmimisjärgus oleva rakenduse puhul on vali tehnoloogia, mida on võimalikult lihtne integreerida olemasolevaga.

Tehnoloogiat valides tasub meeles pidada, et ainuõiget valikut ei eksisteeri. Matt Aimonetti on öelnud, et head ideed on võimalik rakendada peaaegu igal tehnoloogial. Neil kõigil on omad eelised ja miinused. Pole vahet, mille kasuks valik langetatakse, eeliste saamiseks tuleb teatud ohverdused tahest tahtmata tuua. Projekti edu või ebaõnnestumine sõltub eelkõige visioonist, juhtimisest, täidesaatmisest ja turustamisest kui tehnoloogilistest valikutest. Tehnilised valikud tuleb põhjendada eelkõige endale, partneritele ja potentsiaalsetele investoritele (What Technology Should My Startup Use, Aimonetti).

3.2. Andmebaasid

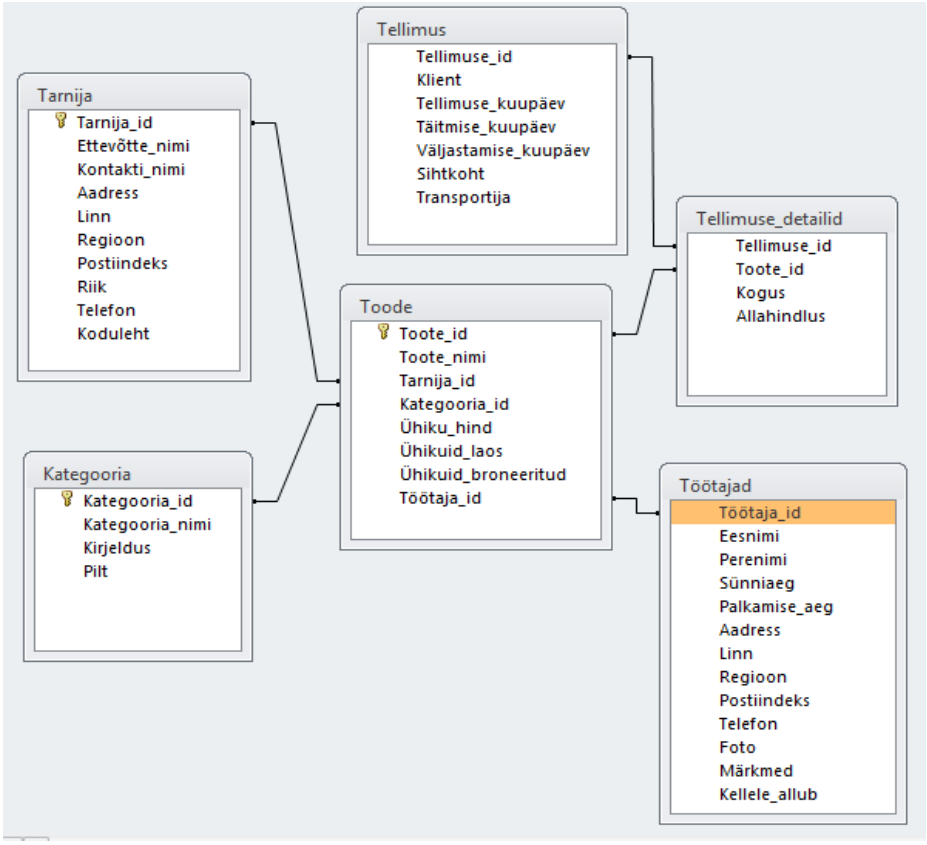
Andmebaas on organiseeritud andmete kollektsioon, mille andmestikule on tagatud ligipääs ja selle uuendamise võimalused. Enamlevinud päringukeel andmebaasiga suhtlemiseks on SQL (*Structured Query Language*), mida loetakse andmebaasistandardiks.

3.2.1. Relatsiooniline andmebaas

Relatsiooniline andmebaas on tabelite hulk, mis sisaldab eeldefineeritud kategooriatesse sisestatud andmeid. Iga tabel sisaldab ühte või mitut veerus olevat andmekategooriat. Iga rida sisaldab veergude kategooriates unikaalset andmeeksemplari. Kasutajad saavad andmestikku kasutada või erinevaid moodusi kasutades uuesti kokku panna, ilma vajaduseta reorganiseerida andmebaasis olevaid tabeleid (Will NoSQL Databases Live Up to Their Promise, 2010).

Relatsioonilises mudelis peab iga tabel identifitseerima primaarvõtme plaani, milles veerg või grupeeritud veerud võimaldavad iga rida üheselt identifitseerida. Sellisel juhul saab iga tabelis oleva rea vahel luua seose või viidata võõrvõtme abil teise tabeli rea primaarvõtmele (võõrvõti on teise tabeli primaarvõti). See võimaldab ka tabelite kombineerimise tulemusena luua kohaldatud vaateid. Relatsiooniline mudel pakub tabelite organiseerimiseks ja reorganiseerimiseks erinevaid viimistlemise tasemeid, mida nimetatakse andmebaasi normaliseerimiseks (Relational database, 2006).

Joonisel 4 on näide relatsioonilisest andmebaasist:



Joonis 4. Relatsioonilise andmebaasi näide

3.2.1.1. Relatsioonilise andmebaasi tugevused

Kirjeldav andmestruktuur

Relatsioonilist andmebaasi tundva kasutaja jaoks on andmestiku formaat kergesti mõistetav ja lihtsasti kasutatav. Kasutades ära andmebaasi võimet siduda tabelites olevat andmestikku, võib andmebaasis olevast infost luua erinevaid vaateid ja leida uusi uurimist vajavaid valdkondi (The Advantages of a Relational Database Management System, 2013).

Andmete samaaegne kasutamine

Relatsiooniline andmebaas võimaldab erinevatel kasutajatel samaaegset ligipääsu andmebaasile. Sisseehitatud lukustamise ja transaktsioonide funktsionaalsus võimaldavad kasutajatel ligipääsu andmestikule samal ajal kui seda muudetakse, hoida ära andmestiku samaaegsel uuendamisel tekkivad põrked ja hoida kasutajad eemal osaliselt uuendatud andmestikust (The Advantages of a Relational Database Management System, 2013).

Kõrge turvalisus

Autoriseerimise ja privileegide määramise olemasolu võimaldab andmebaasi administraatoritel piirata ligipääsu autoriseeritud kasutajatele. Privileegide andmine kasutajatele sõltub läbiviidavatest toimingu tüüpidest ja kasutaja õigustest pääseda ligi erineva kategooria andmestikule. Ehk töötaja kasutajakontot võib piirata nägema ainult tema valdkonda puudutavat informatsiooni ja peita ülejäänu. Autoriseerimine võib seejuures olla IP-aadressist lähtuvalt, kitsendades ligipääsu ainult kindla võrguaadressiga seadmetele (The Advantages of a Relational Database Management System, 2013).

Kergesti hooldatav

Relatsioonilised andmebaasisüsteemid sisaldavad utiliite, mis võimaldavad administraatoritel erinevaid tööriistu kasutades andmebaase kergesti hooldada, testida, parandada ja varundada. Enamuse funktsionaalsusest saab automatiseerida, kasutades ära andmebaasi sisseehitatud võimalusi või erinevate operatsioonisüsteemide peale loodud automatiseerimise tööriistu (The Advantages of a Relational Database Management System, 2013).

SQL päringukeele lihtsus

Relatsioonilised andmebaasisüsteemid toetavad päringukeelt nimega SQL, mille süntaks on lihtne ja mis kasutab standardseid inglise keele võtmesõnu ja kõnepruuki. See teeb keele küllaltki intuitiivselt ja lihtsalt õpitavaks. Paljud süsteemid lisavad omalt poolt ka SQL väliseid, kuid andmebaasiga seonduvaid võtmesõnu, funktsioone ja võimalusi, rikastamaks kasutatavat süsteemi veelgi rohkem (The Advantages of a Relational Database Management System, 2013).

3.2.1.2. Relatsioonilise andmebaasi nõrkused

Piirangud struktuuris

Relatsioonilistes andmebaasides on andmetele seatud piirangud. Andmebaasi disainides peab määratlema tabelites salvestuva andmete tüübi ja pikkuse. Kui sisestatav andmestik ei sobi eeldefineeritud struktuuriga, siis andmete hoiustamiseks peab andmebaasi ümber disainima, mis on keerukas ja tülikalt teostatav tegevus (Disadvantages of a Relational Database, 2010).

Keerukalt skaleeritav

Relatsioonilise andmebaasi skaleerimiseks on vajalik võimsama ja kallima seadme olemasolu, kuid teatud punktist edasi on vajalik andmete hajutamine erinevate serverite vahel. Relatsioonilised andmebaasid ei tööta kergesti eraldiseisvate süsteemide korral, sest tabelite ühendamine erinevate süsteemide vahel on Craigslisti inseneri Jeremy Zadowny sõnul keeruline. Lisaks ei ole RedMonk analüütiku Stephen O'Grady sõnul relatsioonilised andmebaasid disainitud töötama andmete jaotamise korral (Will NoSQL Databases Live Up to Their Promise, 2010).

Suure jõudluse vajadus

Relatsiooniliste andmebaasisüsteemide suurim puudus on jõudluse olemasolu vajalikkus. Mida rohkem on andmebaasis omavaheliste relatsioonidega seotud tabeleid, seda suuremat jõudlust läheb vaja SQL päringule vastuse koostamiseks. Keerukate SQL päringute töötlemine nõuab ka suuremat jõudlust. Kuigi väikeettevõtetes vajamineva andmebaasi suuruse ja keerukuse haldamisega tulevad toime ka enamus lauaarvuteid, siis

väliste andmehoidlate või keeruka andmestruktuuri töötlemiseks läheb mõistliku aja jooksul vastuse andmiseks väga võimsaid servereid (The "NoSQL" Discussion has Nothing to Do With SQL, 2009).

Andmestiku keerukus

Andmestik jaotub relatsioonilises andmebaasisüsteemis mitmete tabelite vahel, mis viitavad üksteisele võõrvõtmete abil. Relatsiooniline andmebaas ei sunni andmebaasi disainides seoseid kasutavad struktuuri kasutama, mille tõttu võivad kogenematud disainerid luua tarbetu keerukuse või limiteerida tuleviku arendust kehvasti valitud andmetüüpide tõttu. Kogenud disainerid loovad väärtust pakkuvaid süsteeme, kuid kogenematus võib põhjustada ettevõttesisese kaose (Advantage & disadvantage of relational database, 2013).

Ebatäpsed sissekanded

Relatsioonilised andmebaasisüsteemi nõuavad andmete omavaheliseks seostamiseks võõrvõtmete kasutamist. Kui tabelite vahel on ebakõla ja võõrvõtmed viitavad erinevatele andmetele, siis andmebaasi korrapärane töö taastub alles peale andmestiku ümbertegemist. Kui tabelis puudub võõrvõtmele vastav andmestik, siis päringutulemused võivad olla ebatäpsed. Kui andmebaasi kasutavasse rakendusse ei ole sisseehitatud andmebaasi lukustamine andmete uuendamisel, võivad kasutajad tahtmatult tekitada olukorra, kus andmestik on rikutud ja sissekanne muutub kasutuskõlbmatuks (Advantage & disadvantage of relational database, 2013).

Kõrged nõudmised arendajate professionaalsusele

Koos andmebaasi keerukuse kasvuga, tõusevad ka nõuded andmebaasi administraatori oskustele. Kriitilise tähtsusega süsteemid võivad nõuda kogemusi, mille rahastamine käib väikesele ettevõttele üle jõu. Kui andmebaasi disainides ei rakendata koheselt korrektseid tehnikaid, siis edasiarendamisel võivad märkamatuks jääda erinevad nüansid, mis lõppevad katkiste päringute või ebatäpsete tulemustega. Risk suureneb veelgi, kui andmebaasi disain ja rakenduse arendamine tehakse erinevate inimeste poolt (Advantage & disadvantage of relational database, 2013).

SQL päringukeele puudulikkus

SQL kasutamine on mugav, kuid seda juhul kui tegu on struktureeritud andmestikuga. Kuid teist tüüpi informatsiooni puhul on keele kasutamine keeruline, sest SQL on disainitud töötama struktuurse, relatsiooniliselt organiseeritud andmebaasiga, mis sisaldab fikseeritud tabelites informatsiooni. SQL päringukeelega võib kaasneda suurel hulgal keerulist koodi ja koostöö modernse ning agiilse arendusega on puudulik (Will NoSQL Databases Live Up to Their Promise, 2010).

3.2.2. NoSQL andmebaas

NoSQL (tuntud ka kui *Not Only SQL*) hõlmab endas mitmeid erinevaid andmebaasitehnoloogiaid ja arendati välja seoses kasutajate poolt salvestatava mahu suurenemise, objektide ja toodete koguse ja andmete poole pöördumise sageduse tõusuga. Kõige rohkem on NoSQL andmebaase mõjutanud suured, kasvava andmestiku ja nõudliku infrastruktuuriga Veeb 2.0 gigandid nagu Amazon ja Google. Nende poolt on arendatud ka Dynamo ja Big Table NoSQL andmebaasid, mis on inspireerinud mitmeid tänapäeva NoSQL rakendusi (Will NoSQL Databases Live Up to Their Promise, 2010). Relatsioonilised andmebaasid ei ole disainitud tulema toime ulatuslike ja agiilsete väljakutsetega, millega tänapäeva rakendused silmitsi seisavad, ega kasuta ära tänapäeval kättesaadavat vähest jõudlust ja töötlemist nõudvat salvestamist ja andmetöötlus (NoSQL database explained, 2014).

NoSQL andmebaasid on fokusseeritud kindlate probleemide lahendamiseks, millega relatsioonilised andmebaasid hakkama ei saa - paindlikust salvestavast andmestruktuurist kuni andmebaasi lihtsustamiseni, mis väljendub võtmete ja sellele vastavate väärtuste salvestamisel. NoSQL esindab uue ajastu tehnikaid andmete salvestamisel ja lugemisel, kuid uut lähenemist on tihtipeale ignoreeritud, sest erinetakse traditsioonilisest ja omaks võetud SQList liiga palju (Everyone knows about SQL, but what is NoSQL, 2011).

NoSQL andmebaasid jagunevad neljaks suureks tüübiks (The four categories of NoSQL databases, 2011)):

1. Võti-väärtus salvestamine - koostatakse räsitabel, kus on unikaalne võti ja viitaja konkreetse objekti andmestikule. Kõige lihtsamini ja kergemini rakendatav, aga küllaltki ebaefektiivne.
2. Laiades veergudes salvestamine - eesmärk on salvestada ja töödelda paljude masinate vahel jaotunud suuri andmemahtusid. Kasutusel on võtmed, aga nad viitavad mitmetele veergudele.
3. Dokumendi andmebaas - Sarnane võti-väärtus salvestamisele. Mudel on kogum dokumentidest, mille sees asuvad omakorda võti-väärtus kollektsioonid.
4. Graafide andmebaas - kasutatakse paindliku graafi mudelit, mis võib katta mitmeid masinaid. Ei taga kõrgetasemelist deklaratiivset päringu keelt nagu SQL vältimaks töötlemisel liigset aja kulumist.

Võrrelduna relatsioonilise andmebaasiga on NoSQL andmebaas rohkem skaleeruv ja pakub suuremat jõudlust. Tema andmemudel võtab arvesse erinevaid punkte, millele relatsioonilise andmebaasi disain ei ole sobilik (NoSQL database explained, 2014):

- suuremahulised struktureeritud, pool-struktureeritud ja struktureerimata andmestik
- Agiilsed sprindid, kiired iteratsioonid ja sagedased versiooniuuendused
- objektorienteeritud programmeerimine, mis on lihtne kasutada ja paindlik
- tõhus ja suure mastaabiline arhitektuur kalli ja monoliitse arhitektuuri asemel

3.2.2.1. NoSQL andmebaasi tugevused

Paindlik andmemudel

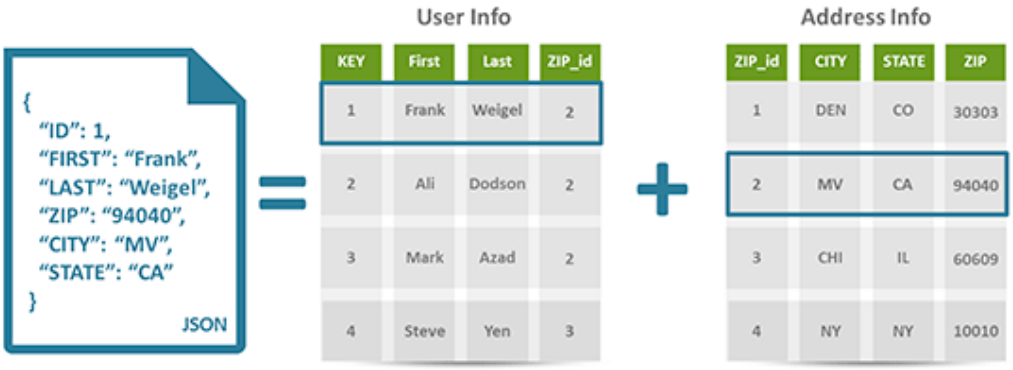
Relatsiooniline andmebaasi mudel jagab andmestiku omavahel seotud tabelitesse. Tabelid viitavad üksteisele läbi võõrvõtmete, ning andmeid pärides peab informatsiooni enne rakendusele kasutamiseks andmist kõigepealt kokku panema ja erinevate tabelitega kombineerima.

NoSQL andmebaasidel on kasutusel teistsugune mudel. Näiteks dokument-orienteeritud NoSQL andmebaas koondab salvestatava andmestiku erinevateks dokumentideks kasutades JSON formaati. Igat JSON dokumenti võib käsitleda kui rakenduse poolt kasutatavat objekti. Informatsiooni koondamine võib lõppeda info dubleerimisega, aga

kuna hoiustamine pole enam takistuseks, andmemudel on paindlik, dokumentide levitamine kerge ja lugemine-kirjutamine sooritatakse kiiremini, siis on tegu hea vahetusega.

Relatsioonilised tehnoloogiad järgivad rangeid skeeme, kuid NoSQL mudelid on skeemitud. Relatsioonilised tehnoloogiad nõuavad enne andmete salvestamist jäika skeemi, mille alusel andmete salvestamine toimub. Andmebaasi juba lisatud andmete puhul on skeemi tagantjärgi muutmine väga segadusttekitav ja sagedasti välditav tegevus, kuid tänapäeva arendajad peavad rakenduste rikastamiseks pidevalt uute andmetüüpidega kokku puutuma (Why NoSQL?, 2014).

Järgneval joonisel on välja toodud erinevus andmete salvestamisel NoSQL ja relatsioonilises andmebaasis. NoSQL andmebaasis esitletakse andmed ühe suure kogumina kui relatsioonilises andmebaasis on andmed grupeeritud.



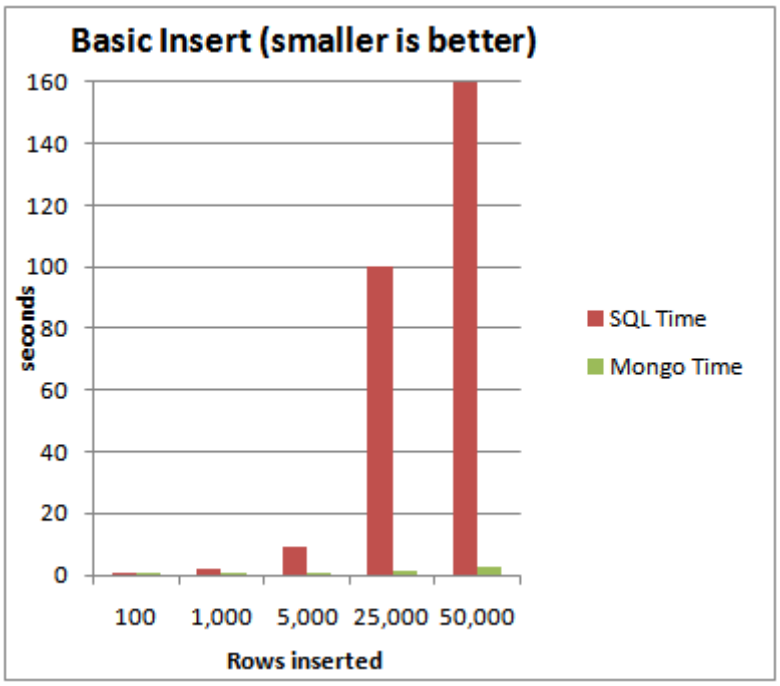
Joonis 5. NoSQL ja relatsioonilise andmebaasi salvestamise erinevus

Kergesti skaleeritav

Tulemaks toime kasutajate arvu ja töödeldavate andmete hüppelise kasvuga, on NoSQL arendatud *scale-out* tüüpi andmebaasiks. Andmete salvestamiseks ja andmebaasi tegevuste toetamiseks on kasutusel standardsed, füüsilised või virtuaalsed klastrid. Skaleerumiseks lisatakse olemasolevatesse klastritesse servereid juurde ja edaspidi levib andmebaas koos andmetega üle suurema klastri. Lisaks ei ole süsteemi laiendamisel vajalik modifitseerida rakendust, sest skaleerudes näeb rakendus andmebaasi alati ühtse tervikuna (Why NoSQL?, 2014).

Vähese jõudluse vajadus

Joonis 6 annab ülevaate kiiruse erinevusest, mis kulub NoSQL ja SQL Server 2008 andmebaasil sissekannete tegemiseks andmebaasi. Joonisel on näha, et väiksemate koguste juures täidetakse sissekanded võrdsetel kiirustel, kuid andmete hüppelisel suurenemisel edastab MongoDB konkurentsitult SQL-andmebaasi. Andmed pärinevad võrdlusest, mis viidi läbi 29. Aprillil 2010 aastal kui võrreldi MongoDB ja SQL Server 2008 vahelist jõudlust:



Joonis 6. MongoDB ja SQL Server 2008 sissekannete võrdlus

Tugev andmete terviklikkus

Kõik andmebaasisüsteemi kasutavad kliendid näevad sama versiooni andmestikust. Seda isegi juhul kui andmebaasis on läbi viidud erinevaid uuendusi. Erinevalt relatsioonilistest andmebaasisüsteemidest ei teki andmetes tulenevalt uuendamistest või kustutamistest erinevusi (NoSQL Database: New Era of Databases for Big Data Analytics - Classification, Characteristics and Comparison, 2013).

Kõrge andmete kättesaadavus

Kõikidel klientidel on alati võimalus saada ligipääs vähemalt ühele andmestiku instantsile, seda isegi juhul kui mõned klastris olevad masinad on lõpetanud töötamise.

Sellest tulenevalt on andmebaasi töökindlus kõrge, sest andmebaasisüsteem jätkab talle ette nähtud tegevuste sooritamist isegi kui sõlmedes esinevad tõrked (NoSQL Database: New Era of Databases for Big Data Analytics - Classification, Characteristics and Comparison, 2013).

3.2.2.2. NoSQL andmebaasi nõrkused

NoSQL andmebaasid on tekitanud rohkelt entusiasmi ja köitnud erinevate probleemide lahendamisega arendajate tähelepanu. Uute tehnoloogiate kasutuselevõtmine on arendajate jaoks alati põnevust tekitav, kuid ettevõtted peaksid hetkel veel neisse ettevaatlikult suhtuma. Märkimisväärsetest eelistest hoolimata esineb ka puuduseid ja enne massilise leviku saavutamist peavad NoSQL andmebaasid ületama mitmeid takistusi.

Välja arenemata andmebaas

Relatsioonilised andmebaasid on nüüdseks väga kaua aega kättesaadavad olnud. NoSQL toetajate jaoks on tegu argumendiga, et kõrge vanus on märk vananemisest, kuid erinevate juhtide jaoks lisab relatsiooniliste andmebaaside küpsus kindlustunnet. Lisaks on relatsiooniliste andmebaaside puhul tegu stabiilsete ja rohkelt funktsionaalsust pakkuvate süsteemidega. See-eest NoSQL süsteemides ei ole võtmetähtsusega funktsionaalsus tihtipeale koheselt kättesaadav ja paigaldamiseks on vajalik lisategevus. (Everyone knows about SQL, but what is NoSQL, 2011). NoSQL andmebaasides puudub ka lukustamise ja transaktsioonide funktsionaalsus, mis võib andmebaasis olevat andmestikku rikkuda kui seda manuaalselt ei kindlustata. Vastava võimekuse puudumine tagab parema jõudluse ja skaleerimise, kuid muutub teatud valdkondades probleemseks nagu panganduses.

Võrrelduna traditsioonilise relatsioonilise andmebaasiga, on NoSQL aktsepteeritavus väga puudulik. Relatsioonilised andmebaasid on ettevõtlus maastikul paremini väljakujunenud ja pakuvad paremat funktsionaalsust. (The five key advantages and disadvantages of NoSQL, 2013).

Ebapiisav tugi

Tähtsad teenused, mille pidev olemasolu on vajalik, vajavad tõrgete esinemisel kohest ja kompetentset tuge. Enamus relatsiooniliste andmebaaside tarnijatest seda teevad ja pakuvad väga kõrgel tasemel nõustamist, mis pakub erinevatele ettevõtetele kindlustunnet. Hetkel on NoSQL süsteemide arendajad enamasti idufirmadest väikeettevõtted, mis pakuvad ilma globaalse kättesaadavuseta tuge. Lisaks puuduvad neil ressursid vajamineva nõudluse rahuldamiseks ja piisav usaldusväärsus võrreldes Oracle, Microsofti või IBMiga (Everyone knows about SQL, but what is NoSQL, 2011).

Vähe võimalusi andmebaasi analüüsimiseks

NoSQL andmebaasid on arendatud modernsete Veeb 2.0 rakenduste nõuetele sobivaks, mis tähendab seda, et enamus omadusi on tuletatud Veeb 2.0 rakendustest. Probleem on selles, et tüüpilistes rakendustes tehtavatest toimingutest nagu „sisesta, loe, uuenda ja kustuta“ jääb väheks. Andmebaasides olev informatsioon pakub palju rohkem väärtust kui algselt tunduda võib ja ettevõtted kaevandavad seda muutumaks efektiivsemaks ja konkurentsivõimelisemaks. Seal on ettevõtetele väga väärtuslikku informatsiooni. NoSQL andmebaasid pakuvad taoliste analüüside läbiviimiseks väga vähe võimalusi ja erinevalt relatsioonilistest andmebaasidest, enamus analüüsi tööriistu ei ühildu NoSQL andmebaasidega. (Everyone knows about SQL, but what is NoSQL, 2011).

Keerukas administreerimine

NoSQL eesmärk võib olla administreerimisest vaba lahendus, aga hetke reaalsus seda ei kajasta. NoSQL vajab paigaldamiseks väga häid teadmisi ja teenuse ülalpidamiseks järjepidevat pingutamist (Everyone knows about SQL, but what is NoSQL, 2011).

Kogenematud arendajad

Relatsioonilise andmebaasi kontsepti ja programmeerimisega tuttavaid olevaid arendajaid on miljoneid. See-eest peaaegu kõik NoSQL arendajad on hetkel õpirollis, alles õpivad tehnoloogiat tundma ja omavad NoSQL kontseptsioonist limiteeritud teadmisi. See tähendab seda, et palju lihtsam on leida kogenud arendajat ja administraatorit relatsioonilisele andmebaasile kui NoSQL eksperti (Everyone knows about SQL, but what is NoSQL, 2011).

Kitsas fookus

Üks peamisi põhjuseid, miks NoSQL ei vaheta kunagi SQLi välja on see, et ta pole mõeldudki seda tegema. NoSQL andmebaasidel on väga kitsas fookus: nad on disainitud peamiselt andmete salvestamiseks ja pakuvad lisaks sellele väga vähest funktsionaalsust. Kui andmebaasis viiakse läbi transaktsioone, siis on relatsiooniline andmebaas parem valik. Lisaks sellele, NoSQL andmebaasides ei ole andmete varundamine samal tasemel nagu SQL-andmebaasides (The five key advantages and disadvantages of NoSQL, 2013).

Avatud lähtekood ja standardi puudumine

Avatud lähtekoodiga NoSQL on samaaegselt nii tugevus kui ka nõrkus. Probleem seisneb selles, et NoSQL andmebaasidel puudub ühtne standard. See tähendab seda, et kaks väliselt sarnast andmebaasi ei ole loodud samadel alustel, mis võib väljenduda samasisuliste andmebaaside teistsuguses kohtlemises (The five key advantages and disadvantages of NoSQL, 2013).

Päringute koostamise keerukus

Kuna NoSQL andmebaasid ei toeta SQL päringukeelt, siis iga päring vajab käsitsi programmeerimist, mida on lihtsate ülesannete puhul kiire teostada, kuid aeganõudvamad keerukamate jaoks. Ühtlasi on keerulise päringu programmeerimine ka väga raske (Will NoSQL Databases Live Up to Their Promise, 2010).

4. Relatsiooniline andmebaas ja NoSQL andmebaas mobiilseadmetes

Mobiilirakenduste arendamine on viimastel aastatel tõusnud hüppeliselt. Nutitelefonid ja tahvelarvutid on tõstnud võimsalt mobiilseadmete numbreid üle terve maailma. Koos mobiilseadmete kommuuni kasvuga on tõusnud ka kasutajate nõudlused ja ootused. Täitmaks arendajatele püstitatud nõudmisi, on järjest olulisemaks saanud vajadus muuta arendusprotsessi tõhusamaks ja võimalikult stressivabaks. (Why NoSQL Trumps Relational Databases for Mobile Applications, 2013).

Arendades mobiilseadmetele rakendusi, on Android operatsioonisüsteemis relatsiooniline andmebaas vaikimisi kättesaadav, millest tulenevalt pole paigaldamine ega eelnev seadistamine vajalik, erinevalt NoSQL andmebaasist. NoSQL vajab seevastu tehnoloogiaga tutvumist, välja valimist ja vanade harjumuste väljavahetamist. Kuid kas relatsiooniline andmebaas siiski on parim valik mobiilseadmetes kasutamiseks? Võttes arvesse NoSQL andmebaasi eeliseid relatsioonilise andmebaasi ees nagu skaleeritavus, jõudlus ja paindlik andmemudel, siis NoSQL rakendamine arendamises seda kõike täpselt teebki.

Relatsioonilised andmebaasid pole piisavalt paindlikud

Traditsioonilised SQL andmebaasid kasutavad fikseeritud skeeme. See tekitab küsimusi, sest mobiilirakendustes on rohkelt situatsioonipõhiseid nõudeid. Nii kui arendajad tulevad uute ideedega lagedale, muutuvad muudatuste tegemised rakenduses aeganõudvateks tegevusteks, sest muudatused peavad olema läbiviidud andmebaasi skeemi tasemel. Arendades näiteks mängu, mille erinevat tüüpi tegelaskujudel on erinevad tegevused, siis relatsioonilises andmebaasis võib andmebaas uue muudatuse salvestamiseks muutuda täies mahus. Lisaks ei ole relatsioonilised andmebaasid ehitatud nii, et nad haldaksid kõiki erinevaid kasutusjuhte, mida mobiilsed rakendused nõuavad. Kasutusjuhud võib jaotada nii mobiilseadme tüübi, operatsioonisüsteemi, operatsioonisüsteemi versiooni ja asukoha järgi. See muutub veelgi keerulisemaks kui võtta arvesse erinevaid kombinatsioone, mis tulenevad kasutajate aegunud operatsioonisüsteemide kasutamisest, reisimistest ja paljudest teistest situatsioonidest. Isegi SQL asjatundjad tunnistavad, et relatsioonilise andmebaasi mudel ei ole kõige

sobivam mobiilsetele rakendustele (Why NoSQL Trumps Relational Databases for Mobile Applications, 2013).

NoSQL on mobiilirakenduse arendamiseks sobivam

NoSQL andmebaasid on disainitud mobiilirakenduste dünaamilistest vajadustest lähtuvalt, nad ei kasuta fikseeritud skemaatikat. Arendades näiteks mängu, ei pea arendajad uue tegelaskuju lisamisel andmebaasis drastilisi muudatusi tegema.

NoSQL andmebaasid annavad lahenduse ka eelnevalt mainitud suurele kasutusjuhtude kogusele. Üks parimaid näiteid NoSQL andmebaasi rakendamisel rohkete kasutusjuhtude katmiseks on Foursquare. Kuna Foursquare on asukoha põhine rakendus, siis kasutajate päringute tulemused või rakenduses seadistatavad valikud erinevad asukohast lähtuvalt. Georuumi võimekust ärakasutav vabavaraline NoSQL andmebaas nagu MongoDB teeb arendajatele võimalikuks kerge vaevaga lisada asukoha põhiseid tegevusi (Why NoSQL Trumps Relational Databases for Mobile Applications, 2013).

NoSQL pöörab tähelepanu ka rakenduste pideva uuendamise vajadusele. Peale rakenduse valmimist saab hooldamine peamiseks tegevuseks. Kuna NoSQL on dokumendi põhine, siis teatud tüüpi vigade ja probleemide parandamine ei nõua andmebaasi täielikku uuendamist, sest arendajate tehtud parandused ei mõjuta tihtipeale rakenduse kõiki aspekte (Why NoSQL Trumps Relational Databases for Mobile Applications, 2013).

NoSQL on tuntud oma skaleeritavuse poolest. On väga tähtis, et enne rakenduse arendamist oleks kasutajabaasi kasvamise ja andmestiku suurenemise kasvustrateegia paigas, sest tänapäeval võib kasutajate hulk kasvada ööga miljoniteni. Pöörates tähelepanu mahu piirangutele peale rakenduse väljalaskmist, lõppeb muudatuste tegemine teenuse töös seisakutega ja ärritunud kasutajatega (Why NoSQL Trumps Relational Databases for Mobile Applications, 2013).

4.1.1. NoSQL tehnoloogiad mobiilseadmetes

Tänapäeval leidub rohkelt üksteisele konkurentsi pakkuvaid NoSQL tehnoloogiaid, mis on kasutajate seas laialdaselt levinud, kuid operatsioonisüsteemi põhiselt loodud andmebaasid (*native databases*) veel levinud ei ole. Miks see oluline on? Sest spetsiifilise keskkonna jaoks loodud teenused võtavad arvesse kindla operatsioonisüsteemi eripärasid,

töötavad kiiremini ja on töökindlamad (Native App, 2013). Samas on populaarsetele NoSQL andmebaasidele loodud erinevaid lahendusi, mis võimaldavad rakendustel ühendada ennast pilves töötavate andmebaasidega.

Käesoleval hetkel puuduvad ülevaated olemasolevatest ja teiste seas esiletükkivatest tehnoloogiatest ja sarnane veebileht nagu http://db-engines.com/en/ranking/ puudub, ehk subjektiivse otsuse, millist platvormi kasutada, peab rakenduse arendaja langetama üksikute kasutajate arvamustele tuginedes. Sellest hoolimata leidub mitme-platvormilisi lahendusi, mis imiteerivad näiteks dokumendipõhise andmebaasi käitumist, kuid tehnoloogiate valik ja koodinäidete olemasolu on suhteliselt kehv.

Otsingute tulemustena on välja toodud osa olemasolevatest NoSQL tehnoloogiatest, mida on võimalik kasutada Android operatsioonisüsteemil:

- JasDB
- LevelDB
- WaspDB
- Couchbase Lite
- SnappyDB

4.2. NoSQL tehnoloogia Party Starteris

Võttes arvesse eelnevalt välja toodud NoSQL andmebaasi tugevusi relatsioonilise andmebaasi ees, on Party Starteris otstarbekaks kasutada andmehoidlana NoSQL andmebaasi. Seetõttu tuleb rakenduse prototüübis kasutatud relatsiooniline andmebaas SQLite välja vahetada NoSQL andmebaasi Couchbase Lite vastu, sest tegu on sobivama tehnoloogiaga ja sobib mobiilsetes rakendustes kasutamiseks paremini.

NoSQL andmebaasi looja Couchbase on võtnud eesmärgiks muuta Couchbase Lite mobiilirakenduste arendajate esmaseks eelistuseks. Couchbase Lite on disainitud eelkõige töötama spetsiaalselt iOS ja Android operatsioonisüsteemi kasutavates seadmetes, mitte pilves, kus mobiilirakenduste andmebaasid enamasti hetkel töötavad (Couchbase rolls out native NoSQL databases for iOS, Android, 2013).

Couchbase Lite on esimene iOS ja Android operatsioonisüsteemi põhiselt loodud *native* andmebaas. Tegu on JSONi formaati kasutava dokumendi andmebaasiga, mis on integreeritud juhtivate mobiilirakenduste arendamise platvormidesse. Couchbase on partner selliste ettevõtetega nagu Xamarin, Appcelerator ja Adobe PhoneGap, et antud kommuunide arendajad saaksid eelistatud platvormidel Couchbase Lite tehnoloogiat kasutada. Lisaks võimaldab Couchbase Lite lokaalset andmebaasi sünkroniseerida ka pilves töötava Couchbase serveriga kasutades selleks Couchbase Sync Gateway tehnoloogiat (Couchbase for Mobile, 2014).

Couchbase Lite kasuks räägivad ka otsingumootori Google tulemused. Tehnoloogia kasutajatel on positiivsed kogemused ja julgetakse soovitada ka teistel antud tehnoloogia kasutusele võtta. Lisaks kasutajate arvamustele, on erinevate NoSQL tehnoloogiate koodinäidete otsingutulemused kõige paremad just Couchbase Lite tehnoloogial. Leidub rohkelt õpetusi ja erinevatel konverentsidel lindistatud videotutvustusi, mis annavad ülevaate tehnoloogiast ja tema kasutamisest.

Erinevate tehnoloogiate tutvumisel tekkis käesoleva töö autoril vajadus leida täiendavat informatsiooni andmebaasi initsialiseerimise kohta. Uue tehnoloogia tõttu on leiduvad näited pealiskaudsed ja spetsiifilistele küsimustele ammendavaid vastuseid reeglina ei leia. Probleemi lahendamiseks pöördus autor Google grupis oleva Couchbase Mobile nimelise kommuuni poole. Probleemi kirjeldavast postitusest kulus ammendava vastuseni 48h, mis tähendab, et tegu on väga tugeva ja abivalmi kommuuniga. Kommuun asub aadressil: https://groups.google.com/forum/#!forum/mobile-couchbase/

4.3. Couchbase Lite rakendamine

Käesoleva magistritöö raames asendati töö autori poolt Party Starteri rakenduses relatsiooniline andmebaas NoSQL andmebaasiga. Eesmärgiks oli tagada süsteemi olulisemate protsesside korrapärane toimimine ehk põhifunktsionaalsuse olemasolu. Antud peatükis välja toodud koodinäited on pärit Party Starteri realisatsioonist, mis annavad ülevaate tehnoloogia toimimisest ja sellega kaasnevatest tähelepanekutest.

Relatsioonilise andmebaasi väljavahetamine ja NoSQL andmebaasi kasutuselevõtmine jaguneb viieks suuremaks iteratsiooniks, milleks on:

1. Andmebaasi initsialiseerimine
2. Andmebaasi dokumentide loomine
3. Kategooriate pärimine
4. Kirjelduste eelvaadete pärimine
5. Kirjelduse pärimine

4.3.1. Andmebaasi initsialiseerimine

Couchbase Lite tehnoloogia kasutusele võtmiseks on rakenduse esmakordsel käivitamisel vajalik andmebaasi initsialiseerimine vaikimisi väärtustega. Selle toimingu sooritamiseks on mitmeid võimalusi:

- Tühja andmebaasi ülekirjutamine eelnevalt initsialiseeritud `.cblite` andmebaasi failiga
- Andmebaasi sünkroniseerimine pilves asuva serveriga -
- Andmefaili kasutamine andmebaasi initsialiseerimisel

Andmebaasi sünkroniseerimiseks on vajalik eraldiseisev Couchbase serveri olemasolu ja käesoleval hetkel ei ole mõistliku panusega võimalik luua ka eelnevalt initsialiseeritud andmebaasi, sest puuduvad vajaminevad tööriistad nagu SQLite Database Browser. Sellest tulenevalt on eelnevalt välja toodud valikust kõige mõistlikum kasutada andmebaasi initsialiseerimiseks JSON formaadil põhinevat andmefaili, sest võti-väärtus andmebaasi on JSON-objekte kõige kergem üle kanda.

JSON andmefaili loomine

Rakendus on oma ülesehituselt hierarhiline, mis tähendab seda, et aplikatsioonis kasutatav andmestik on grupeeritud erinevate kategooriate vahel. Tagamaks võimalikult lihtsa andmestiku initsialiseerimise ja vaikimisi saadaval oleva informatsiooni redigeerimise vajaduse tulevikus, jälgib sarnast struktuuri ka JSON andmefail, mis võiks välja näha järgnev:

```
{
  "categories":[
    {
      "name" : "Pealkiri",
      "description" : "Kirjeldus",
```

```
      "imageID" : "Meedia ",
      "descriptions" : [
        {
          "name" : "Sissekanne",
          "shortDescription" : "Eelvaate kirjeldus",
          "formation" : "Formatsiooni tunnus",
          "equipment" : "Varustus",
          "longDescription" : "Põhikirjeldus",
          "imageID" : "Meedia",
          "rating" : "Hinnang"
        },
        {
          "name" : "Sissekanne",
          "shortDescription" : "Eelvaate kirjeldus",
          "formation" : "Formatsiooni tunnus",
          "equipment" : "Varustus",
          "longDescription" : "Põhikirjeldus",
          "imageID" : "Meedia",
          "rating" : "Hinnang"
        }
      ]
    }
  ]
}
```

Koodinäide 2. JSON andmefail

Tegu on vähima funktsionaalsuse toimimiseks vajamineva andmestikuga ega peegelda rakenduse poolt kasutatavat lõplikku struktuuri.

Andmebaasi initsialiseerimine

Andmebaasi initsialiseerimiseks on rakenduse esmakordsel käivitamisel vaja kontrollida andmebaasi olemasolu. Selleks on vaja pöörduda kõiki NoSQL andmebaase haldava objekti poole, mis tagab ligipääsu vajaminevale informatsioonile. Kui andmebaasi süsteemis ei leidu, siis järelikult on tegu esmakordse käivitamisega ja vajalik on andmebaasi initsialiseerimine. Selleks tuleb avada eelnevalt defineeritud andmefail ja sooritada vajaminevad toimingud.

Järgnevalt on välja toodud andmebaasi olemasolu tuvastamiseks vajaminevad toimingud:

```
//Open Database Manager
```

```
Manager = new Manager(context.getApplicationContext().getFilesDir(),
                      Manager.DEFAULT_OPTIONS);
//Define database
Database db = null;
//Get existing database
db = manager.getExistingDatabase(DATABASE_NAME);
//if equals null, need to initialize
if (db == null) {
  //initialize database
```

Koodinäide 3. Andmebaasi olemasolu tuvastamine

Kui süsteemi poolt on tuvastatud andmebaasi puudumine, siis tekitatakse andmebaas `getDatabase()` meetodiga, avatakse eelnevalt loodud JSON andmefail ja luuakse sisust `JSONObject` objekt:

```
  //create database
  db = manager.getDatabase(DATABASE_NAME);
  String jsonString = null;
  //open JSON-file
  InputStream inputStream = context.getAssets().open(DATA_FILE);
  int size = inputStream.available();
  byte[] buffer = new byte[size];
  inputStream.read(buffer);
  inputStream.close();
  jsonString = new String(buffer);
  //create JSON-object
  JSONObject data = new JSONObject(jsonString);
}
```

Koodinäide 4. Andmebaasi loomine ja andmefaili avamine

4.3.2. Andmebaasi dokumentide loomine

Peale andmebaasi initsialiseerimist on vajalik sisestada vaikimisi olemasolev informatsioon. Kuid enne andmebaasi sissekannete lisamist, on vajalik otsustada andmebaasis olevate dokumentide struktuur. Kuna dokumentidesse salvestatakse andmed võti-väärtus põhimõttel ja puudub relatsioonilisele andmebaasile omane piirangutega struktuur, siis sellest tulenevalt on dokumentide loomist võimalik realiseerida mitmel viisil.

Objektis sisalduvate objektide keerukuse illustreerimiseks on joonisel 7 välja toodud näide dokumendi võimalikust struktuurist:

```
{
 key : List<Map<String, Object>>[
  {
   key : value,
   key : value,
   key : value,
   key : List<Map<String, Object>>[
    {
     key : List<Map<String, Object>>[
          {
               key : value
               key : value
          },
          {
               key : value,
               key : value
          }
      ],
     key : value,
     key : value
    },
    {
     key : value,
     key : value,
     key : value
    }
   ]
  }
 ]
}
```

Joonis 7. JSON andmestruktuur

Kuna dokumentide andmestruktuur ei ole fikseeritud, siis andmete dokumenti lisamiseks on kasutusel `putProperties()` meetod, mille argument on kujul `Map<String, Object> properties`. Rakenduse arendaja jaoks tähendab see seda, et ainuke piirang asub võtmel, mis peab olema `String` tüüpi, kuid võtme väärtuseks võib näiteks omakorda olla `Map<String, Object> properties`. See võimaldab luua dokumendi siseselt sügava hierarhia, kus võtme väärtuseks on omakorda andmeid sisaldav objekt või objektide massiiv.

Alternatiivne võimalus on loobuda dokumendisisese hierarhia koostamisest ja käsitleda igat objekti eraldi dokumendina. Sellisel juhul peab rakenduse arendaja tagama objektide

omavaheliste seoste loomise, sest vastasel juhul ei ole võimalik eristada erineva taseme objekte või luua otsene ligipääs objektile. Hierarhilist struktuuri kasutavas dokumendis dubleerimise vajadus samas puuduks.

Käesoleva magistritöö raames puutus töö autor kõigepealt kokku hierarhilise struktuuriga ja alles seejärel alternatiivse variandiga, milles igat objekti käsitleti eraldi dokumendina. Mõlema variandi puhul alustati hierarhia tipus olevate objektide töötlemisega ja seejärel objektide sees olevate objektidega. Sellega saadi ülevaade erinevate struktuuride kasutamise tehnikatest ja erinevustest, mis väljenduvad objektide töötlemise keerukuses. Ühtlasi selgus ka rakenduse arendamiseks kõige mõistlikum variant.

Hierarhilise struktuuriga dokumendid

Hierarhilist struktuuri kasutavate dokumentide loomiseks on efektiivne moodus kasutada utiliit meetodid, mis andmestiku dokumenti lisamiseks õigesse formaati teisendavad. Selgus, et tipus asuvate objektide töötlemine probleeme ei tekita ja on kiirelt rakendatav tehnoloogiasse sisseehitatud mugavusmeetodite tõttu, kuid probleemid hakkavad tekkima hierarhias üha sügavamale minnes. Koodi haldamine, sellest arusaamine ja muudatuste sisseviimine muutub aina aega nõudvamaks, keerulisemaks ja tehnoloogia kasutamine üha raskemaks.

Järgnevalt on välja toodud koodinäide tipus asuvate dokumentide lugemisest, mis kasutavad hierarhilist struktuuri. Näites tagab andmebaase haldav objekt ligipääsu konkreetsele andmebaasile, mis omakorda koostab kõikidest dokumentidest koosneva päringu. Päringus sisalduvad kõik andmebaasi sisestatud dokumendid, mis on ühtlasi hierarhia tipus asuvad objektid. Seejärel on objektide hulka võimalik iteraatorit kasutades ükshaaval läbi käia ja rakenduse siseselt kasutada:

```
Database db = manager.getExistingDatabase(DATABASE_NAME);
Query query = db.createAllDocumentsQuery();
QueryEnumerator rowEnum = query.run();
for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
  QueryRow row = it.next();
  Document d = db.getExistingDocument(row.getDocumentId());
  list.add(documentToCategory(d));
}
```

Koodinäide 5. Kategooriate lugemine hierarhilises struktuuris

Kui eelnev näide illustreeris kõrgeima taseme objektide leidmist, siis järgnevalt on välja toodud hierarhilise struktuuri koodinäide, mis kirjeldab sügavamal tasemel olevate objektide leidmist, päringu tulemuse koostamist ja seal leiduvate objektide töötlemist.

Sügavamate tasemete või kindlatele kriteeriumitele vastavate objektide lugemiseks peab defineerima andmestiku struktuuriskeemi, mille tööpõhimõte on sarnane relatsioonilises andmebaasis kasutatavale päringule. Struktuuriskeem sisaldab endas eeltingimuste kogumit, millele vastates lisatakse dokument `emitter.emit()` meetodit kasutades tulemuste hulka:

```
db = manager.getExistingDatabase(GAMES_DATABASE_NAME);
View gamesForCategory = db.getView("descriptionsForCategory");
gamesForCategory.setMap(new Mapper() {
@Override
public void map(Map<String, Object> document, Emitter emitter) {
  if(document.get("name").equals(categoryID)){
    Object gamesData = document.get(GAMES_KEY_IN_JSON);
    List<Object> gamesList = (List<Object>) gamesData;
    for(Object g : gamesList){
       Map<String, Object> m = (Map<String, Object>) g;
       emitter.emit(m.get("name"), m);
    }
  }
}
}, "1.0");
```

Koodinäide 6. Kirjelduste eelvaadete andmestruktuur hierarhilises struktuuris

Peale struktuuriskeemi koostamist käivitatakse päring ja kõik struktuuriskeemile vastavad dokumendid lisatakse tulemuste hulka. Kuna tegu on hierarhilise struktuuriga, siis peab ka tulemusi läbides järge pidama, millisel tasemel andmetega tegu on:

```
View v = db.getView("descriptionsForCategory");
Query q = v.createQuery();
QueryEnumerator rowEnum = q.run();
for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
  QueryRow row = it.next();
  Map<String, Object> m = (Map<String, Object>) row.getValue();
  showText(m.get("shortDescription").toString());
}
```

Koodinäide 7. Kirjelduste eelvaadete päringu töötlemine hierarhilises struktuuris

Rakenduse arendamise keerukus tuleneb sellest, et andmebaasis olev dokument on võti-väärtus andmehoidla ja ei võimalda universaalset ligipääsu sügavamal tasemel olevatele objektidele. Ligipääs andmetele toimub meetodi `getProperty(String key)` väljakutsumisel, mis tagastab võtmele vastava `Object` tüüpi sisu. Sellest tulenevalt peab rakenduse arendaja pidevalt kursis olema andmestruktuuriga ja käsitsi defineerima objektide sügavused. See tähendab seda, et muudatuste sisseviimine ning uue funktsionaalsuse lisamine ei ole kiirelt teostatav, vaid aeganõudev tegevus.

Sellest tulenevalt ei ole dokumentides sügava hierarhilise struktuuri kasutamine põhjendatud ja tehnoloogia kasutuselevõtmisel on käsitletud igat objekti eraldi dokumendina.

Objektid eraldi dokumentidena

Sõltuvalt hierarhilise struktuuri kasutamise keerukusest, on otstarbekas käsitleda andmebaasi initsialiseerimisel igat `Map<String, Object>` tüüpi objekti eraldi dokumendina, lisada eristatavust tagavad tunnused ja tekitada dokumentide vahelised seosed.

Hierarhia tipus asuvad objektid lisainformatsiooni ei vaja, kuid eristatavuse tagamiseks ja lisafunktsionaalsuse võimalikuks lisamiseks on lisatud tüübi atribuut. Dokumentide koostamine ja andmebaasi lisamine võiks välja näha järgnev:

```
JSONArray categories = data.getJSONArray(CATEGORY_KEY_IN_JSON);
for (int i = 0; i < categories.length(); i++) {
  JSONObject category = categories.getJSONObject(i);
  Map<String, Object> categoryMap = new HashMap<String, Object>();
  categoryMap.put("name", category.getString("name"));
  categoryMap.put("description", category.getString("description"));
  categoryMap.put("imageID", category.getString("imageID"));
  categoryMap.put("isBought", category.getString("isBought"));
  //additional information. Set type attribute
  categoryMap.put(TYPE_ATTRIBUTE, CATEGORY_TYPE_ATTRIBUTE);

  Document c = db.getDocument(category.getString(CATEGORY_ID_IN_JSON));
  c.putProperties(categoryMap);
```

Koodinäide 8. Objektidest andmebaasi dokumentide loomine

Objektides sisalduvatele objektidele tuleb lisaks tüübi tunnusele lisada ka relatsioonilisest andmebaasist pärit võõrvõti, mis võimaldab luua objektide vahelisi seoseid. Võõrvõtmeks peab olema unikaalne tunnus, mis käesoleva projekti raames on kategooria nimetus.

Dokumentide loomine ja seoste loomine võiks välja näha järgnev:

```
  JSONArray games = category.getJSONArray(GAMES_KEY_IN_JSON);
  for(int j = 0; j < games.length(); j++){
    JSONObject game = games.getJSONObject(j);
    Map<String, Object> gameMap = new HashMap<String, Object>();
    gameMap.put("name", game.getString("name"));
    gameMap.put("shortDescription", game.getString("shortDescription"));
    gameMap.put("formation", game.getString("formation"));
    gameMap.put("equipment", game.getString("equipment"));
    gameMap.put("longDescription", game.getString("longDescription"));
    gameMap.put("imageID", game.getString("imageID"));
    gameMap.put("rating", game.getString("rating"));
    gameMap.put("isCategoryBought", category.getString("isBought"));
    //reference to type attribute
    gameMap.put(TYPE_ATTRIBUTE, GAME_TYPE_ATTRIBUTE);
    //reference to category object
    gameMap.put(CATEGORY_FID, category.getString(CATEGORY_ID_IN_JSON));

    Document g = db.getDocument(game.getString(GAME_ID_IN_JSON));
    g.putProperties(gameMap);
  }
}
```

Koodinäide 9. Dokumentide vaheliste seoste loomine

4.3.3. Kategooriate pärimine

Peale andmebaasi initsialiseerimist ja dokumentide loomist tuleb realiseerida järgmine iteratsioon, milleks on kategooriate pärimine. Kategooriate kuvamine kasutajale on süsteemis toimuvatest põhisündmustest, mis kirjeldab kasutajale andmebaasis olevaid sarnaste sissekannete gruppe.

Andmestruktuur

Kategooriate pärimise puhul on tegu hierarhilise struktuuri tipus asuvate objektide lugemisega. See tähendab seda, et andmestiku struktuuriskeemi koostades peavad

objektid läbima ainult tüübi tunnuse eeltingimuse, mis lisati andmebaasi initsialiseerimisel igale dokumendile. Iga kategooria on andmestiku siseselt omakorda seotud ka kirjelduste eelvaadete ja kirjeldustega, kuid kuna kategoorias sisalduvad kirjeldused on andmebaasis eraldiseisvad dokumendid, siis andmestruktuuri koostades neid arvesse võtma ei pea.

Kategooriate struktuuriskeem võiks välja näha järgnev:

```
List<Category> list = new ArrayList<Category>();
Database db = manager.getExistingDatabase(DATABASE_NAME);
View categoryView = db.getView("categoryView");
categoryView.setMap(new Mapper() {
  @Override
  public void map(Map<String, Object> document, Emitter emitter) {
    if(document.get(TYPE_ATTRIBUTE).equals(CATEGORY_TYPE_ATTRIBUTE))
      emitter.emit(document.get(CATEGORY_ID_IN_JSON), document);
  }
}, "1.0");
```

Koodinäide 10. Kategooriate struktuuriskeem

Objektide loomine

Peale struktuuriskeemi koostamist saab päringus olevatest andmetest luua vajaminevad `Category` objektid. Kuna dokumendis ei sisaldu teistest objektidest koosnevaid massiive, siis saab igat võtmele vastavat väärtust kohelda `String` tüübina ja lugeda andmetest vajaminev informatsioon `getProperty()` meetodiga, mida kasutajatele kasutajaliideses kategooriatena kuvatakse:

```
Query q = categoryView.createQuery();
QueryEnumerator rowEnum = q.run();
for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
  QueryRow row = it.next();
  Document d = row.getDocument();
  String name = (String) d.getProperty("name");
  String imageId = (String) d.getProperty("imageID");
  String description = (String) d.getProperty("description");
  boolean isBought = Boolean.parseBoolean((String) d.getProperty("isBought"));
  list.add(new Category(name, imageId, description, isBought));
}
```

Koodinäide 11. Kategooriate päringu töötlemine

4.3.4. Kirjelduste eelvaadete pärimine

Peale kategooriate pärimise realiseerimist on vajalik realiseerida järgmine iteratsioon, milleks on kirjelduste eelvaadete pärimine. Kirjelduste eelvaadete kuvamine kasutajale on süsteemis toimuvatest põhisündmustest, mis kirjeldab kasutajale kindlasse kategooriatesse kuuluvaid seltskonnamänge.

Andmestruktuur

Kirjelduste eelvaadete puhul on tegu kirjelduse minimalistliku variandiga, mida kuvatakse kasutajale kirjeldusest lühikese ülevaate andmiseks. See tähendab seda, et kirjelduse eelvaate loomiseks kasutatav andmebaasi dokument sisaldab endas kirjelduse kohta käivat terviklikku informatsiooni, kuid `DescriptionPreview` objekti initsialiseerimiseks on vaja ainult osa sellest. Kuna see puudutab otseselt objektide loomist, siis erinevalt kategooriate struktuuriskeemist, peab kirjelduste eelvaadete päringu koostamisel defineerima lisaks tüübi tunnusele ka kategooria välisvõtme eeltingimuse, milleks on dokumendi initsialiseerimisel määratud unikaalne kategooria nimi.

Kirjelduste eelvaadete andmestiku struktuuriskeem võiks välja näha järgnev:

```
List<DescriptionPreview> list = new ArrayList<DescriptionPreview>();
db = manager.getExistingDatabase(DATABASE_NAME);
View gamesForCategory = db.getView("gamesView");
gamesForCategory.setMap(new Mapper() {
  @Override
  public void map(Map<String, Object> document, Emitter emitter) {
    if(document.get(TYPE_ATTRIBUTE).equals(GAME_TYPE_ATTRIBUTE) &&
document.get(CATEGORY_FID).equals(categoryID))
      emitter.emit(document.get(GAME_ID_IN_JSON), document);
  }
}, "1.0");
```

Koodinäide 12. Kirjelduste eelvaadete andmestruktuur

Objektide loomine

Peale kirjelduste eelvaadete struktuuriskeemi koostamist saab päringus olevatest andmetest luua vajaminevad `DescriptionPreview` objektid. Kuna dokumendis

sisalduvad ainult kirjeldustega seotud andmed, siis objektide initsialiseerimise tööpõhimõte on sarnane `Category` objektide initsialiseerimisele:

```
Query q = gamesForCategory.createQuery();
QueryEnumerator rowEnum = q.run();
for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
  QueryRow row = it.next();
  Document d = row.getDocument();
  String name = (String) d.getProperty("name");
  String shortDescription = (String) d.getProperty("shortDescription");
  String formation = (String) d.getProperty("formation");
  String equipment = (String) d.getProperty("equipment");
  String imageID = (String) d.getProperty("imageID");
  boolean isCategoryBought = Boolean.parseBoolean((String)
d.getProperty("isCategoryBought"));
  list.add(new DescriptionPreview(name, shortDescription, formation,
equipment, imageID, isCategoryBought);
}
db.deleteViewNamed("gamesView");
```

Koodinäide 13. Kirjelduste eelvaadete päringu töötlemine

4.3.5. Kirjelduse pärimine

Peale kirjelduste eelvaadete pärimise realiseerimist on vajalik realiseerida järgmine iteratsioon, milleks on kirjelduse pärimine. Kirjelduse kuvamisega kasutajale antakse detailne ülevaade seltskonnamängust, mis on ühtlasi ka süsteemi põhisündmustest.

Andmestruktuur

Kirjelduse pärimine on sarnane kirjelduse eelvaadete päringule, sest nii `Description` kui ka `DescriptionPreview` objekti initsialiseerimise aluseks on andmebaasis olev ühine dokument, mis salvestab endas vastavat informatsioon. Erinevalt kirjelduste eelvaadete andmestruktuurist ei ole vaja kontrollida enam kategooria välisvõtme vastavust, vaid piisab unikaalse nime kontrollimisest:

```
db = manager.getExistingDatabase(DATABASE_NAME);
View description = db.getView("description");
description.setMap(new Mapper() {
  @Override
  public void map(Map<String, Object> document, Emitter emitter) {
```

```
    if(document.get(TYPE_ATTRIBUTE).equals(GAME_TYPE_ATTRIBUTE) &&
document.get(GAME_ID_IN_JSON).equals(descriptionName))
      emitter.emit(document.get(GAME_ID_IN_JSON), document);
  }
}, "1.0");
```

Koodinäide 14. Kirjelduse andmestruktuur

Objektide loomine

Peale kirjelduste andmestruktuuri koostamist saab päringus olevatest andmetest luua vajamineva `Description` objekti. Erinevus kirjelduste eelvaadete pärimisega seisneb sellest, et päringu tulemuseks on üks objekt, mille tõttu on tulemuste limiteerimiseks kasutatud `setLimit(1)` meetodit. Süsteemi töö tulemiks on üksainus `Map<String, Object>` tüüpi objekt, mida on võimalik objekti initsialiseerimiseks järgnevalt kasutada:

```
Query q = description.createQuery();
q.setLimit(1);
QueryEnumerator rowEnum = q.run();
Document document = null;
for (Iterator<QueryRow> it = rowEnum; it.hasNext();) {
  QueryRow row = it.next();
  Document d = row.getDocument();
  String name = (String) d.getProperty("name");
  String formation = (String) d.getProperty("formation");
  String equipment = (String) d.getProperty("equipment");
  String description = (String) d.getProperty("longDescription");
  int rating = Integer.parseInt((String)d.getProperty("rating"));
  new Description(name, formation, equipment, description, rating);
  document = row.getDocument();
}
db.deleteViewNamed("description");
```

Koodinäide 15. Kirjelduse päringu töötlemine

4.4. Registridokumendi kasutamine alternatiivina

Rohkete sissekannete korral muutub peatükis 3.3 välja toodud lähenemise korral päringutöötlemise kiirus väga aeglaseks. Töötlemiskiiruse tulemused on välja toodud

peatükis „Kokkuvõte Couchbase Lite kasutuselevõtmisest“, kuid järgnevalt antakse ülevaade alternatiivsest lähenemisest, mis kõnealuse probleemi lahendab.

Päringutöötlemise kiiruse probleem seisneb NoSQL tehnoloogia omapäras, mis väljendub selles, et kõik andmebaasis olevad dokumendid käiakse ükshaaval läbi ja iga dokumendi andmestruktuur läbib eeltingimustele vastamise kontrolli. Rohkete sissekannete korral on iga dokumendi läbikäimine ja objektis olevate andmete kontrollimine ajakulukas. Alternatiivse lahendusena võib rakendusesiseselt luua registridokumendi, kuhu on salvestatud teadmised kategooriatest ja kategooriate alla kuuluvatest sissekannetest. Antud teadmisi kasutades saab muuta päringutöötluse mitmeid kordi kiiremaks, sest ligipääs luuakse ainult vajaminevatele dokumentidele.

```
registryDocument{
    “categories” : “categoryName_1, categoryName_2”,
    “categoryName_1” : “description_2, description_3”,
    “categoryName_2” : “description_1, description_4”
}
categoryName_1{
    key : value
}
categoryName_2{
    key : value
}
description_1{
    key : value
}
description_2{
    key : value
}
description_3{
    key : value
}
description_4{
    key : value
}
```

Joonis 8. Registrit kasutava andmebaasi tööpõhimõte

Joonisel 8 illustreeritakse registrit kasutava andmebaasi tööpõhimõtet. Ülesehituselt on rakenduse tööpõhimõte sarnane peatükis 3.3 väljatoodule, kuid asendunud on andmete töötlemine. Andmebaasis oleva andmestiku aluseks on `registryDocument` dokument, mis luuakse andmebaasi initsialiseerimisel. Dokumendis on „categories“-võtme väärtuseks salvestatud kategooriate nimekiri. Kategooriate nimetused on dokumendis kasutusel eraldi võtmetena ja igale kategooria võtmele vastab kategooriasse kuuluvate sissekannete nimistu. Sellega kaardistatakse ära andmebaasis olevad sissekanded ja tekitatakse kunstlik hierarhiline struktuur erineva taseme objektidest. Dokumentide loomisel peab meeles pidama, et kategooriate ja kirjelduste nimetused kanduksid edasi andmebaasi dokumendi nimele. Selleks tuleb kasutada

`getDocument(String documentName)` meetodit ja määrata `documentName` väärtuseks kategooria või kirjelduse nimetus.

Andmebaasi dokumentide loomine

Dokumentide loomise põhimõte sarnaneb peatükis 3.3.2 väljatoodule, kuid teadmised objektide vahelistest seostest salvestatakse registridokumenti:

```
JSONArray categories = data.getJSONArray(CATEGORIES_REFERENCE);
Document sourceDocument = db.getDocument(REGISTRY_DOCUMENT);
Map<String, Object> sourceMap = new HashMap<String, Object>();
String categoriesString ="";
for (int i = 0; i < categories.length(); i++) {
  //store category values to document...
  categoriesString = categoriesString + category.getString("name") + ",";
  String gamesString = "";
  JSONArray games = category.getJSONArray(GAMES_KEY_IN_JSON);
  for(int j = 0; j < games.length(); j++){
    //store description values to document
    descriptionsString = descriptionsString + game.getString("name") +",";
  }
  sourceMap.put(category.getString("name"), descriptionsString);
}
sourceMap.put(CATEGORIES_REFERENCE, categoriesString);
sourceDocument.putProperties(sourceMap);
```

Koodinäide 16. Registridokumendi loomine

Kategooriate pärimine

Peale registridokumendi loomist tuleb muuta ka kategooriate pärimise tööpõhimõtet, sest kasutusel pole enam andmestruktuurid ja päringutöötlemise asemel pöördutakse koheselt kõnealuste dokumentide poole:

```
List<Category> list = new ArrayList<Category>();
Database db = manager.getExistingDatabase(DATABASE_NAME);
Document sourceDocument = db.getDocument(REGISTRY_DOCUMENT);
String categoryList = (String) sourceDocument.getProperty("categories");
for(String category : categoryList.split(","))
  list.add(documentToCategory(db.getDocument(category)));
```

Koodinäide 17. Kategooriate pärimine registridokumendiga

Kirjelduste eelvaadete pärimine

Kirjelduste eelvaadete pärimise tööpõhimõte sarnaneb kategooria dokumentidele ligipääsu loomisega. Registridokument omab teadmisi kindla kategooria alla kuuluvatest kirjeldustest ja dokumentide lugemisel pöördutakse ainult vajaminevate objektide poole, mida rakendusesiseselt kasutatakse:

```
List<DescriptionPreview> list = new ArrayList<DescriptionPreview>();
db = manager.getExistingDatabase(DATABASE_NAME);
Document sourceDocument = db.getDocument(REGISTRY_DOCUMENT);
String descriptionsList = (String) sourceDocument.getProperty(categoryID);
for(String descriptionName : descriptionsList.split(","))
  list.add(documentToDescriptionPreview(db.getDocument(descriptionName)));
```

Koodinäide 18. Kirjelduste eelvaadete pärimine registridokumendiga

Kirjelduse pärimine

Peale kirjelduste eelvaadete pärimise kohandamist tuleb uuendada ka kirjelduse pärimist. Kuna andmebaasis olevad dokumendid on nimetatud kirjelduste või kategooriate nimest lähtuvalt, siis kirjelduse pärimisel tuleb pöörduda õige nimega andmebaasi dokumendi poole:

```
db = manager.getExistingDatabase(DATABASE_NAME);
Document document = db.getDocument(descriptionName);
return documentToDescription(document);
```

Koodinäide 19. Kirjelduse pärimine registridokumendiga

4.5. Kokkuvõte Couchbase Lite rakendamisest

Käesolev peatükis on välja toodud kokkuvõtlik ülevaade Couchbase Lite andmebaasi rakendamisest ja antud hinnang uue andmebaasisüsteemi kasutuselevõtmisest. Hinnangu andmisel on käesoleva magistritöö autori poolt arvesse võetud nii otseselt kui ka kaudselt mõõdetavaid mõõdikuid.

Efektiivne rakenduse arendamine

Käesoleva magistritöö raames sai töö autor ülevaate NoSQL tehnoloogia tõelisest eelisest, milleks oli andmebaasi poolt kasutatava struktuuri puudumine. NoSQL

tehnoloogia rakendamine mobiilseadmetes muutis eelkõige rakenduse arendamise kergemaks, sest muudatused oli võimalik sisse viia jooksvalt ja ressurssi ei pidanud kulutama kõrvalistele tegevustele.

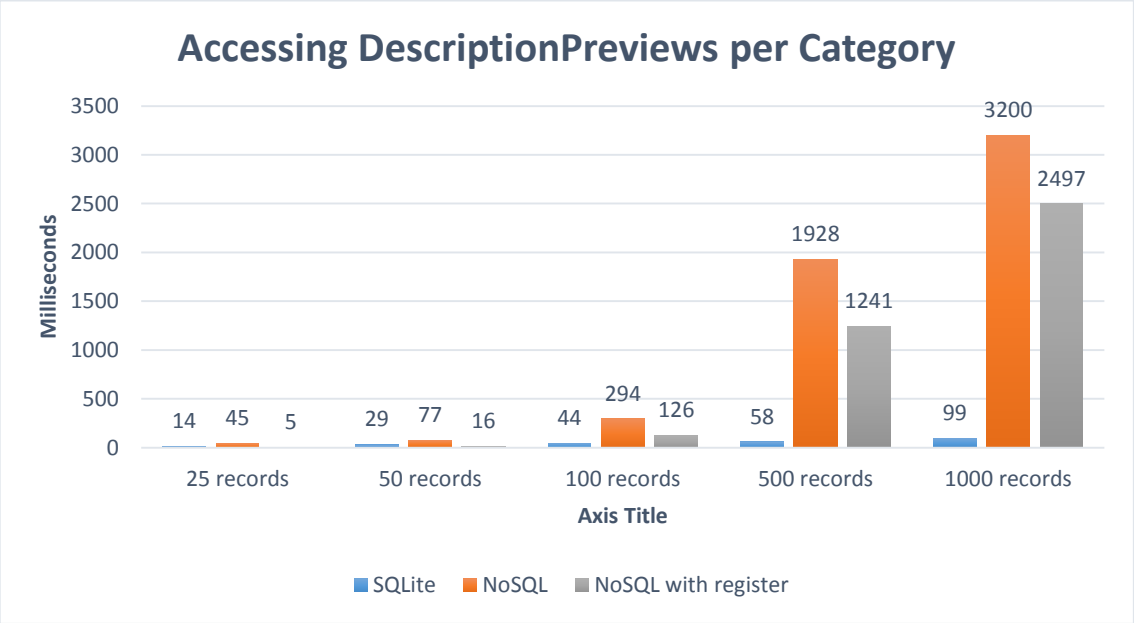
Päringutöötlemise kiiruse muutumine

Päringutöötlemisele kuluva aja mõõtmine on käesoleva magistritöö kontekstis üks objektiivsemaid mõõdikuid. Ülevaate saamiseks viidi seadmesisesed mõõtmised läbi kolmes erinevas rakenduse realisatsioonis, kus erinevus seisnes andmete lugemisel:

- Andmete lugemine relatsioonilisest andmebaasist
- Andmete lugemine NoSQL andmebaasist kasutades andmestruktuuri
- Andmete lugemine NoSQL andmebaasist kasutades registridokumenti

Kuna andmebaasides olevate sissekannete kogus on otseses seoses päringutöötlemiseks kuluva ajaga, siis sissekannete arvu määramisel lähtuti käesoleva rakenduse skoopi jäävatest sissekannete arvust. Andmebaasis olevate sissekannete kogus oli mõõtmistel vahemikus 25 – 1000.

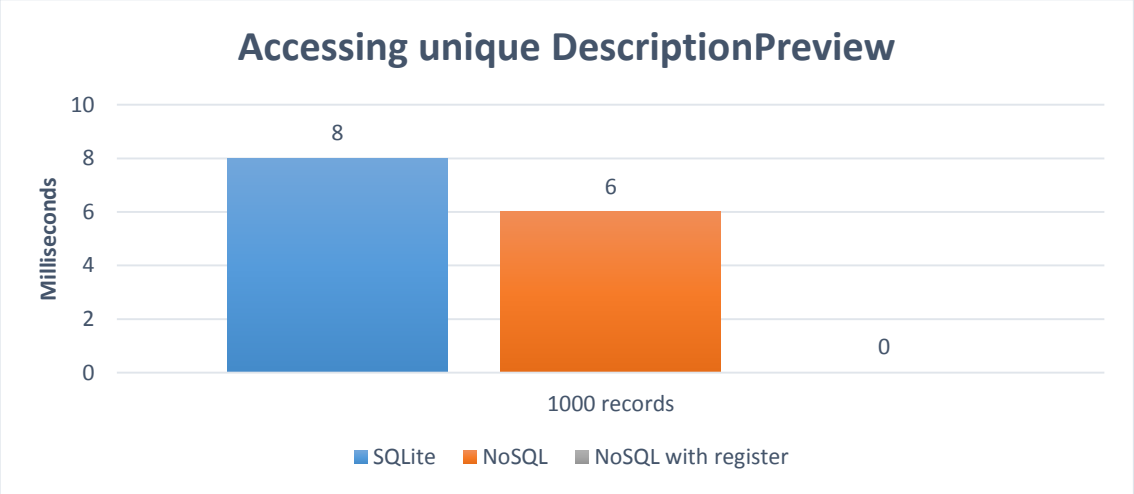
Joonisel number 9 on välja toodud kategooria kirjelduste eelvaadete töötlemiseks kuluv aeg:



Joonis 9. Kirjelduste eelvaadete andmetöötluse tulemused

Joonisel on näha, et käesoleva töö skoopi jäävate sissekannete arvu korral, milleks on kuni 50 kirjeldust kategooria kohta, on andmetöötlus kiireim registridokumenti kasutades. Sissekannete koguste kasvades tõuseb ka NoSQL andmebaasis andmetöötluseks kuluv aeg, mis on selgitatav töödeldavate dokumentide ja dokumendisiseste andmete lugemise kasvuga. Tulemustest selgus, et andmetöötluseks on registridokumendi kasutamine sobivaim variant. Kuid registridokumendi haldamine on rakendusesiseselt keerukas ja tähelepanu nõudev tegevus, sest kirjelduste lisamised ja eemaldamised peavad kajastuma kõnealuses dokumendis. Andmestruktuuri kasutavas andmetöötluses antud tegevused vajalikud ei ole. Sellest tulenevalt peab rakenduse arendaja ise otsustama, millist lähenemist kasutada, sest tegurid sõltuvad mitmest muutujast.

Joonisel number 10 on välja toodud unikaalse kirjelduse töötlemiseks kuluv aeg 1000 sissekande hulgast:



Joonis 10. Kirjelduse andmetöötluse tulemused

Andmetöötlus on kiireim registridokumenti kasutavas andmebaasis, sest ligipääs luuakse otse soovitavale dokumendile. Märkimisväärne on ka andmestruktuuri kasutava NoSQL andmebaasi tulemus, mille kiire reageerimisaeg on selgitatav andmetöötluse tulemuste mälus salvestamisega. Sellest tulenevalt luuakse ligipääs andmestikule momentaalselt.

NoSQL andmebaasitehnoloogia kasutuselevõtmisega ei ole käesoleva töö skoopi jäävate sissekannete puhul rakendus märgatavalt kiiremaks muutunud, vaid teatud juhtudel isegi aeglasemaks. Tehnoloogiate vahelised erinevused andmetöötluses on millisekundites ja rakenduse kasutaja Party Starteris toimunud muudatusi arvatavasti tähele ei pane. Küll

aga on rakendus muutunud vastuvõtlikuks muudatustele. Rakenduse arendajatel on aplikatsiooni funktsionaalsust võimalik juurde lisada jooksvalt. Uue tehnoloogiaga puudub vajadus valmistada ette muudatuste sisse viimine. See tähendab seda, et rakendus on uue tehnoloogia kasutusele võtmisega muutunud efektiivsemaks ja konkurentsivõimelisemaks.

4.6. Party Starteri edasiarendamise võimalused

Seoses Couchbase Lite andmebaasi kasutuselevõtmisega on arendajatele lisandunud hüppeliselt juurde võimalusi uue funktsionaalsuse lisamiseks. Ainuüksi uue tehnoloogia omadusi ära kasutades on võimalik rakendust täiendada, näiteks pakkudes kasutajatele uusi vaateid, mis lisab mitmekesisust, ilma et selleks oleks tarvis programmi erilisi muudatusi teha.

Reklaamide näitamise integreerimine

Käesoleva projekti raames on loobutud *freemium* ärimudelist ja kasutaja ei pea valima piiratud tasuta ja piiramatu tasulise funktsionaalsusega rakenduse vahel. Rakenduses on võimalik tekitada lisatulu reklaamide integreerimisega ja teenida kasutajabaasi suuruselt, kelleni reklaamid jõuavad. Kuid reklaamide näitamisega ei tohi langeda kasutusmugavus ja kogu protsess peab olema läbimõeldud. Pealetükkivad ja kasutajat sundivad reklaamid on tüütud ja tekitavad rakenduse kasutamisel positiivsete asemel negatiivseid emotsioone.

Couchbase Serveri kasutuselevõtmine

Lisaks tehnoloogia omadustest lisanduvatele uutele võimalustele, pakuvad Couchbase arendajad ka Couchbase Serverit, mida on võimalik Couchbase Lite tehnoloogiaga ühildada (Couchbase for Mobile, 2014). See tähendab seda, et pilves asuva serveri olemasolul, on võimalik luua andmevahetus mobiilse seadmega. Rakenduse arendajate jaoks muutub arendusprotsess lihtsamaks, sest vaikimisi olemasoleva informatsiooni uuendamisel puudub vajadus uuendada kogu süsteemi. See võimaldab arendajatel redigeerida olemasolevat ja lisada uut sisu kasutajat segamata. Sissekannete sünkroniseerimine pilves asuva serveriga võiks toimuda näiteks rakenduse käivitudes kui toimub süsteemi laadimine.

Koos seltskonnamängude sünkroniseerimise võimaluse lisandumisega, on rakenduse arendajatel võimalik erinevatest seadmetest koguda kerge vaevaga ka kasutaja poolt genereeritud statistikat. Kuna arendajad ei pea muret tundma andmevahetuse programmeerimisega kaasnevate probleemide üle, siis jääb rohkem ressurssi statistika töötlemiseks, mis väljendub kasutajale huvipakkuvas sisus. Suhtlus pilves asuva serveri ja mobiilseseadme vahel on võimalik ka relatsioonilises andmebaasis, kuid sünkroniseerimine, erinevalt Couchbase tehnoloogiast, vaikimisi puudub.

Kasutajate poolt loodud sisu kasutamine

Rakenduse levik ja elujõulisus sõltub kasutajale atraktiivsuse pakkumisest. Populaarsete rakenduste ja veebilahenduste eduvõtmeks on osutunud kasutajate kaasamine sisu tootmisesse. See tähendab seda, et lisaks rakenduste haldajatele on ka kasutajatel tähtis roll sisu üle, mis rakenduses kättesaadav on. Kuna Party Starteris pakutav sisu otsustatakse hetkel meeskonna siseselt, siis on suur tõenäosus, et kasutajatele huvipakkuv potentsiaalne informatsioon võib märkamatuks jääda. Siinkohal tulevadki appi rakenduse kasutajad, kes lisavad ka omalt poolt seltskonnamänge, mis nende seltskonnas positiivselt silma on jäänud.

Kindlustamaks rakenduses pakutava sisu kvaliteedi, peavad kasutajate poolt pakutud seltskonnamängud läbima eelkontrolli. Vastasel juhul võib rakenduse maht lühikese ajaga liiga suureks kasvada ja suureneb tõenäosus ebakvaliteetsele sisule. Eelkontrolliks võiks olla väljatöötatud hindamismudel, kus kasutajad hindavad üksteise poolt lisatud seltskonnamänge. Ületades eeldefineeritud lävendi, lisatakse sissekanne pilves asuva serveri andmebaasis sobivasse kategooriasse ja sünkroniseeritakse hiljem automaatselt kasutaja seadmega.

Sissekannete lisamist ja teiste kasutajate seltskonnamängude hindamist on eeldatavasti kõige mugavam lauaarvuti tagant teha. Sellest tulenevalt on arukas luua vastava funktsionaalsusega veebilahendus, mis seda kõige võimaldaks. Lisaks annab see ka uue väljundi kliendibaasi suurendamiseks, sest kõik ei oma nutitelefoni. Seetõttu oleks hea mõte luua sissekannetele ka veebipõhine ligipääs, mis annaks võimaluse ka lisatulu teenimiseks.

Olemasoleva funktsionaalsuse kasutamine teistes projektides

Käesoleva projekti arenduses on arendajad lähtunud objektorienteeritud programmeerimise põhimõtetest. See on väljendunud objektide vahelistes madalates sõltuvustes, mis tähendab seda, et projektis kasutatavad komponendid on korduvkasutatavad ka teistes rakendustes, sest puudub üksteisest liigne sõltuvus. See võimaldab meeskonnal luua uusi projekte ja kasutada olemasolevat ning toimivat arhitektuuri. Äriprotsessidega seotud ülesannete loogika ja näiteks andmebaasist andmestiku lugemise tööpõhimõte on läbivalt sama, kohandamist vajab ainult andmete struktuur.

5. Kokkuvõte

Töös käsitletud rakenduse arhitektuur ning tarkvaraarenduse headest tavadest lähtumine on tarkvara arendamise puhul kriitilise tähtsusega. See määrab olemasoleva tarkvaraprojekti hallatavuse ja edasise arenduse kulgemise. Teema muutub eriti aktuaalseks mobiilsete rakenduste puhul, mis peavad pidevalt muutusteks valmis olema, mida "Party Starter" ka on. Rakenduse arendamisel on kasutatud objektorienteeritud mõtteviisi ja rakenduse edasiarendamisel on süsteemi komponente võimalik välja vahetada osade kaupa, ilma terviksüsteemi muutmata. See tõstab oluliselt arendusprotsessi efektiivsust.

Süsteemi toimimise seisukohalt toetab välja toodud arhitektuur rakendusesisest koostööd erinevate komponentide vahel, kus üksteisega ollakse madalas sõltuvuses - komponentides sisse viidavad muudatused ei tähenda muudatusi teistes komponentides. Kuid ainult tarkvaramustrite rakendamisest ei piisa, vaid õiged valikud peab tegema ka tehnoloogiates, mida rakendusesiseselt kasutatakse. Vaikimisi olemasolevate ja esmapilgul ainulahendustena tunduvate tehnoloogiatega paralleelselt on võimalik kasutada uusi, kuid veel vähetuntud tehnoloogiaid. Uued tehnoloogiad võivad arendamiseks olla palju sobilikumad, sest lisaks vana tehnoloogia kasutamisel esinevate probleemide lahendamisega võimaldatakse arendajatele ka uusi funktsionaalseid võimalusi. Kuid süsteemseid muudatusi saab ainult siis mõistliku vaevaga sisse viia kui tarkvaraarendus on laotud korrektsele vundamendile. Valesti kasutatava tööriista kasutegur on väike ning tehnoloogiat valides tuleb samuti meeles pidada, et iga projekt on käsitletav erijuhtumina – universaalne ja kõigile sobiv tehnoloogia puudub ning lähtuma peab eelkõige konkreetsetest vajadustest.

Töö on koostatud ühe projekti näitel, kuid töö tulemused on rakendatavad väga laialdaselt.

"Party Starteri" arenduse seisukohalt on tulevik konkreetne ja selge - käesoleval poolaastal valmib esimene avalikkusele kättesaadav versioon, mis vastab töös kirjeldatud arhitektuurile ja pakub kasutajale uue tehnoloogia rakendamisest tulenevaid hüvesid. Funktsionaalsuse seisukohalt jätkub rakenduse arendamine, mis ideaalis väljendub töös kirjeldatud edasiarendamise võimaluste teostamises.

5.1. Summary

The best practices of architecture and software development discussed in this master's thesis are critical when it comes to development of applications. It determines the existing software manageability and future development of the project. This issue becomes very important for mobile applications what must be constantly ready to change when necessary, just like „Party Starter". This application is developed using object-oriented thinking and its system components can be partly replaced in the future development if necessary, without altering the whole system. It significantly increases the efficiency of the development process.

The system functionality is supported by its architecture, what ensures the cooperation between the different components, which have low dependent on each other - changes in components do not imply changes in other components. But implementing only software patterns is not enough, right choices have to be made within the technologies that are used within the application. In addition to the default, it is possible to implement new, but not yet widely known technologies. New technologies can be more suitable in the application development, because in addition to problem solving, what occurs when we use older technologies, they add new functionalities as well. But systemic changes can only be introduced with reasonable effort if the software development has been stacked in the correct foundation. Improperly used tools have low efficiency and it should be kept in mind, when choosing a technology, that every project should be treated as a special case - there is no universal technology and decisions should be based on its specific needs.

Work is based on one project, but the results can be widely implemented.

Development regarding "Party Starter" is specific and clear - the first version will be made publicly available this semester and corresponds to the architecture described in this thesis and will provide its user the benefits derived from the implementation of the new technology. The development of the applications functionality continues and will reflect the opportunities introduced in this master's thesis.

Kasutatud kirjandus

1. A. B. M. Moniruzzaman. NoSQL Database: New Era of Databases for Big Data Analytics - Classification, Characteristics and Comparison. [WWW] https://www.academia.edu/5352898/NoSQL_Database_New_Era_of_Databases_for_Big_Data_Analytics_-_Classification_Characteristics_and_Comparison (01.08.2013) (Veebiartikkel)
2. About SQLite. [WWW] https://sqlite.org/about.html (2014) (Veebilehekülg)
3. Aimonetti, M. What Technology Should My Startup Use? [WWW] http://matt.aimonetti.net/posts/2013/08/27/what-technology-should-my-startup-use/ (27.08.2013) (Veebiartikkel)
4. Amiri, S. M. A. Advantage & disadvantage of relational database [WWW] http://smhamiri.blogspot.com/2013/04/advantage-disadvantage-of-relational.html (16.04.2013) (Veebiartikkel)
5. Asay, M. Why NoSQL Trumps Relational Databases for Mobile Applications. [WWW] http://www.techopedia.com/2/29256/development/mobile-development/why-nosql-trumps-relational-databases-for-mobile-applications (5.03.2013) (Veebiartikkel)
6. Azharuddin, K. Difference between SQL and NoSQL : Comparision [WWW] http://www.thewindowsclub.com/difference-sql-nosql-comparision (08.11.2011) (Veebiartikkel)
7. Couchbase for Mobile. [WWW] http://www.couchbase.com/mobile/ (2014) Veebilehekülg
8. Duncan, D. GRASP Patterns. [WWW] http://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/duncan.pdf (16.11.2012) (Veebiartikkel)
9. Greene, N. The five key advantages and disadvantages of NoSQL. [WWW] http://www.greendatacenterconference.com/blog/the-five-key-advantages-and-disadvantages-of-nosql/ (14.05.2013) (Veebiartikkel)
10. Harris, D. Couchbase rolls out native NoSQL databases for iOS, Android. [WWW] http://gigaom.com/2013/09/13/couchbase-rolls-out-native-nosql-databases-for-ios-android/ (13.09.2013) (Veebiartikkel)

11. Leavitt, N. Will NoSQL Databases Live Up to Their Promise? [WWW] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5410700 (08.02.2010) (Veebiartikkel)
12. Leggetter, P. Choosing your Realtime Web App Tech Stack. [WWW] http://www.leggetter.co.uk/2013/12/09/choosing-realtime-web-app-tech-stack.html (09.12.2013) (Veebiartikkel)
13. Malik, M. Everyone knows about SQL, but what is NoSQL? [WWW] http://technirvanaa.wordpress.com/2011/03/26/everyone-knows-about-sql-but-what-is-nosql/ (26.03.2011) (Veebiartikkel)
14. Martin, A. Disadvantages of a Relational Database [WWW] http://www.ehow.com/list_5977286_disadvantages-relational-database.html (10.02.2010) (Veebiartikkel)
15. NoSQL database explained. [WWW] http://www.mongodb.com/learn/nosql (2014) (Veebilehekülg)
16. Patel, R. Which technology stack should I choose to build my product? [WWW] http://www.southerntechnologyleaders.com/news/software-and-services/which-technology-stack-should-i-choose-to-build-my-product/ (21.12.2012) (Veebiartikkel)
17. Rouse, M. Native App. [WWW] http://searchsoftwarequality.techtarget.com/definition/native-application-native-app (14.02.2013) (Veebiartikkel)
18. Rouse, M. Relational Database. [WWW] http://searchsqlserver.techtarget.com/definition/relational-database (6.04.2006) (Veebiartikkel)
19. Soltesz, D. L. The Advantages of a Relational Database Management System. [WWW] http://www.ehow.com/list_6121487_advantages-relational-database-management-system.html (23.03.2010) (Veebiartikkel)
20. Stonebraker, M. The "NoSQL" Discussion has Nothing to Do With SQL. [WWW] http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-with-sql/fulltext (09.11.2009) (Veebiartikkel)
21. The four categories of NoSQL databases. [WWW] http://rebelic.nl/2011/05/28/the-four-categories-of-nosql-databases/ (28.05.2011) (Veebiartikkel)
22. Why NoSQL? [WWW] http://www.couchbase.com/why-nosql/nosql-database/ (2014) (Veebilehekülg)

23. Vogel, L. SQLite and Android. [WWW]
http://www.vogella.com/tutorials/AndroidSQLite/article.html (19.08.2013)
(Veebiartikkel)