TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Sigrid Närep 179595IADB

VABA AJA ORGANISEERIMISE RAKENDUS

Bakalaureusetöö

Juhendaja: Meelis Antoi

Magistrikraad

Tallinn 2020

Autorideklaratsioon

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

Autor: Sigrid Närep

30.04.2020

Annotatsioon

Bakalaureusetöö eesmärgiks on luua rakendus vaba aja tegevuste organiseerimiseks. Rakendus peaks lahendama konkreetsele tegevusele osalejate leidmise probleemi. Töö rõhk on olemasolevate lahenduste analüüsil ja uue täiustatud lahenduse välja pakkumisel.

Siiani puudus keskne koht, mis võimaldaks leida inimesi vaba aja tegevuse jaoks, jaotada tööülesandeid loodud sündmuse liikmete vahel ja näha võimalikult täpseid ilmastikutingimusi, otsustamaks, millal sündmus organiseerida.

Analüüsi käigus selgitatakse välja funktsionaalsed ja mittefunktsionaalsed nõuded, peamised kasutaja kasutusjuhud ning lõputöö kirjutamise ajal parimad tehnoloogiad lahenduse elluviimiseks. Arendusprotsessi käigus luuakse serveri- ning kliendipoolne rakendus ja kirjeldatakse kasutatud tehnoloogiaid, metoodikaid ja protsesse vastavalt diplomitöö eraldiseisvates peatükkides.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 80 leheküljel, 8 peatükki, 14 joonist, 1 tabelit.

Abstract

Application for organising leisure time activities

The aim of this bachelor's thesis is to create an application for organising leisure time activities. The application is supposed to solve the problem of finding participants for a specific activity. The emphasis of the work is on the analysis of existing solutions and proposing a new improved solution.

Until now, there was no central place to find people for leisure time activities, to allocate work tasks among the members of the event, as well as being able to see detailed weather conditions in order to decide when and where to organise the event.

In the course of the analysis, the functional and non-functional requirements are identified, together with main use cases. In addition, the best technologies for implementing the solution at the time of writing the dissertation have been analysed in detail. Finally, the implementation and development process of both, the client-side and server-side application, have been described. The server-side and client-side applications are created separately, and the technologies, methodologies and processes used are described in separate chapters of the dissertation.

The result of this thesis is an extensive analysis and a new mobile application. The project covered all parts of software development life cycle. A solid foundation for further development of the mobile application was built. Some functionality has been achieved for both iOS and Android platforms.

The thesis is in Estonian and contains 80 pages of text, 8 chapters, 14 figures, 1 table.

Lühendite ja mõistete sõnastik

.NET	Microsofti tarkvaraplatvorm
ACID	*Atomicity, consistency, isolation, durability* – andmebaasitehingute omaduste komplekt, mille eesmärk on tagada terviklikkus/ kehtivus
ADO	*ActiveX Data Objects* – Microsofti rakendusprogrammi liides
AI	*Artificial Intelligence* – arvutiprogrammi või masina võime mõelda ja õppida
API	*Application Programming Interface* – rakendustarkvara liides
Backend	Teenusepoolne keskkond (server, rakendus ja andmebaas)
Bluetooth	Traadita side tehnoloogia
BPD	*Business Process Diagram* – äriprotsessi visuaalne esitus
BPMN	*Business Process Model and Notation* – äriprotsessi visuaalne esitus
BSON	*Binary JSON* – binaarne serialiseerimisvorming
CGI	*Common Gateway Interface* – defineerib, kuidas veebiserver saab veebilehtede genereerimise delegeerida teistele tekstipõhist sisend-väljundit toetavatele programmidele
CLI	*Command-line interface* – käsurea liides
CMS	*Content Management System* – tarkvararakendus, mida kasutatakse digitaalse sisu loomiseks ja haldamiseks
CSS	*Cascading Style Sheets* – keel, mida kasutatakse märgistuskeeles, näiteks HTML-is, kirjutatud veebilehe kujundamiseks ja esitamiseks
DAL	*Data Access Layer* – arvutiprogrammi kiht, mis pakub lihtsustatud juurdepääsu salvestatud andmetele

DAO	*Data Access Object* – muster, mis pakub teatud tüüpi andmebaasi abstraktset liidest
Dependency Injection	Programmeerimistehnika, mis muudab klassi iseseisvaks selle sõltuvustest
DLL	*Dynamic Link Library* – see fail sisaldab funktsioonide kogu ja muud teavet, millele Windowsi programm juurde pääseb
EF	*Entity Framework* – avatud lähtekoodiga ORM-i raamistik .NET-rakendustele
ERD	*Entity Relationship Diagram* – näitab andmebaasis salvestatud olemikomplektide suhteid
ES6	*ECMA Script 6* – ECMAScripti programmeerimiskeele versioon 6
FaaS	*Function as a Service* – pilvepõhise teenuse kategooria
Garbage collection	Automaatne mälu haldamise vorm
GNOME	*GNU Network Object Model Environment* – osa GNU projektist ja osa vaba tarkvara, ehk avatud lähtekoodiga, liikumisest
GPS	*Global Positioning System* – ülemaailmne asukoha määramise satelliitnavigatsiooni süsteem
GWT	*Google Web Toolkit* – tarkvara arenduse tööriistakomplekt
Hot reload	Rakenduse töö jätkumine uute failide lisamisel, nii et uue versiooni muutused on nähtavad
HTML	*Hyper Text Markup Language* – veebibrauseris kuvamiseks mõeldud dokumentide standardne märgistuskeel
HTTP	*Hypertext Transfer Protocol* – protokoll teabe edastamiseks arvutivõrkudes
IDE	*Integrated Development Environment* – rakendus, mis pakub arendajatele keskkonda koodi kirjutamiseks
IEEE	*Institute of Electrical and Electronic Engineers* – ülemaailmne ühing ja organisatsioon
iOS	*iPhone Operating System* – Apple'i arendatav mobiilsete seadmete operatsioonisüsteem

J2EE	*Java 2 Platform Enterprise Edition* – keskkond ettevõtte tasemel rakenduste arendamiseks ja juurutamiseks
JPA	*Java Persistence API* – Java-rakenduste programmeerimisliidese spetsifikatsioon, mis kirjeldab relatsiooniandmete haldamist rakenduses
JSON	*JavaScript Object Notation* – lihtsustatud andmevahetusvorming
JSX	*JavaScript XML* – JavaScripti süntaksi laiendus
JVM	*Java Virtual Machine* – mootor, mis pakub käituskeskkonda Java-koodi või rakenduste juhtimiseks
Linq	*Language Integrated Query* – päringu süntaks C#-s ja VB.NET-is, et saada andmeid erinevatest allikatest ja formaatidest
MBaaS	*Mobile Backend as a Service* – tava, kus kasutatakse *backend* teenuste toiteks teenusepakkujat
MVC	*Model View Controller* ehk mudel-vaade-kontroller – rakendustes kasutatav arhitektuur
Native	Ökosüsteem, mille näiteks Apple või Google on valinud oma operatsioonisüsteemi jaoks rakenduste arendamiseks
NDK	*Native Development Kit* – tööriist, mis võimaldab programmeerida C/C++ keeles Android seadmetele
NPM	*Node Package Manager* – JavaScripti teekide haldaja
ORM	*Object-Relational Mapping* – põhimõte, mille eesmärk on objekt-orienteeritud koodi kaardistada ümber teise andmevormi
OS X	Apple arvutite operatsioonisüsteemi versioon 10
Overloading	Võimaldab klassil omada rohkem, kui ühte sama nimega funktsiooni, millel on erinev rakendus või erinev arv argumente
PHP	*Personal Home Page Tools* – skriptikeel
Push notification	Hüpikteatis
PWA	*Progressive Web Application* – interneti kaudu tarnitud rakendustarkvara tüüp, mis on loodud levima veebitehnoloogiate abil

RDBMS	*Relational Database Management System* – toode, mis kuvab andmete ülevaate ridade ja veergude kogumina
REST	*Representational State Transfer* – tarkvara arhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid
SDK	*Software Development Kit* – tarkvara arendamise tööriistade kogum ühes installitavas paketis
SOAP	*Simble Object Access Protocol* – veebiteenuste suhtlusprotokoll
SQL	*Structured Query Language* – loodud relatsiooniandmebaasi haldussüsteemis hoidvate andmete haldamiseks
Stackoverflow	Küsimuste ja vastuste veebisait professionaalsetele ja entusiastlikele programmeerijatele
Tasks	Programmeerimise põhiüksused, mida operatsioonisüsteem juhib
UC	*Use Case* – tegevuste või sündmuste loend, mis määratleb rolli ja süsteemi vahelise interaktsiooni eesmärgi saavutamiseks
UML	*Unified Modeling Language* – modelleerimiskeel süsteemi kujunduse visuaalseks kuvamiseks
URI	*Universal Resource Identifier* – tähemärkide jada, mis tuvastab üheselt konkreetse ressursi
URL	*Uniform Resource Locator* – internetiaadress
USB	*Universal Serial Bus* – arvutipordi tüüp
VOB	*Versioned Object Base* – tsentraliseeritud andmebaas, mis salvestab tarkvara konfiguratsioonihaldus süsteemi failide ja kaustade versiooni teavet
WebSocket	Arvutiside protokoll
WebView	Manustatav brauser
WORA	*Write once, run anywhere* – termin, mis osutab konkreetse programmi oletatavale võimele töötada kõigis operatsioonisüsteemides
XML	*Extensible Markup Language* – märgistuskeel, mis määratleb reeglite kogumi dokumentide kodeerimiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Viimastel aastatel on tarbijate nõudlus nihkunud tervislikumate, looduslikemate ja mahepõllumajanduslike valikute kasuks. Kuna tarbijad on hakanud oma toitumisotsuseid põhjalikumalt kaalutlema, siis on hakatud mõtlema ka muudele viisidele, kuidas tervist parandada. Üheks ilmseks viisiks on füüsilise aktiivsuse tõstmine. Tihtipeale aga ei leidu sõprusringkonnas kaaslasi, kes oleksid valmis liituma ja osa võtma spordialadest või vaba aja tegevustest, kus kaks inimest mängivad teineteise vastu või mille harrastamiseks on vaja kaaslasi või meeskonda.

Käesolev lõputöö analüüsib probleemi Eesti kontekstis ja eksisteerivate lahenduste all vaadeldakse keskkondi, mis on saadaval Eestis elavatele inimestele. Probleemi pakutud lahenduseks on mobiilirakendus, mis võimaldab inimestel kuulutusi üles panna huvist leida teatud vaba aja tegevuse harrastamiseks kaaslane. Rakenduse kaudu saab otsida inimesi samade huvidega ja nendega platvormi vahendusel kontakteeruda, organiseerimaks mõlemale või kõigile osalejatele sobiv aeg ja koht vaba aja tegevuse harrastamiseks.

1.1 Taust ja probleem

Bakalaureusetöö autor on sukeldumisega tegelev professionaal. Sukeldumisega seotud kogukonna tegevuste koordineerimine on aga alati olnud suureks peavaluks. Näiteks organiseeritakse paari sukeldumisklubi koostöö tulemusena väljasõite Rummu karjääri, selleks, et kliendigruppe sukelduma viia. Kuna aga enamus Eestis töötavaid sukeldumise professionaale on vabakutselised ja sukeldumist organiseerivad rohkem kui üks sukeldumisklubi, siis räägitakse teineteisest tihti mööda ja selgub, et kohale ei ole tulnud piisavalt instruktoreid klientide kohta. Puudub keskne koht, kus mitu klubi ja mitmed instruktorid saaksid oma aega ühise eesmärgi ümber planeerida. Kõnealuse probleemi lahendamiseks tuleks luua mobiilirakendus, mida saaks kasutada igal ajal, igal pool ja mis lihtsustaks vaba aja tegevuste organiseerimist.

1.2 Metoodika

Käesoleva lõputöö käigus selgitatakse esmalt lahti loodava infosüsteemi skoop, selle funktsionaalsed ja mittefunktsionaalsed nõuded. Seejärel vaadatakse olemasolevaid lahendusi ja analüüsitakse nende funktsionaalsust, tuues välja, kuidas eelpool kirjeldatud rakendus lahendaks eksisteerivate lahenduste puudujääke ja kuidas erineb kavandatav rakendus olemasolevatest.

Uue loodava mobiilirakenduse funktsionaalsuse modelleerimiseks on kasutatud kasutusjuhtude diagrammi, mis näitab seost kasutaja ja erinevate kasutusjuhtumite vahel, millega kasutaja seotud on. Selle koostamisel on kasutatud UML keelt ning diagramm on koostatud *Creately* veebitarkvaras. Kahte kasutusjuhtumit on lähemalt analüüsitud, kasutades blokkskeemi ja BPD-d protsesside visualiseerimiseks. Blokkskeem on koostatud *Creately* veebitarkvaras ja BPD veebipõhise tööriistaga *bpmn.io*.

Analüüsi teises pooles vaadatakse erinevaid võimalikke tehnilisi lahendusi, nii serveripoolseid kui ka kliendipoolseid tehnoloogiaid ja põhjendatakse serveri- ning kliendipoolsete tehnoloogiate valikuid lahenduse elluviimiseks.

Seejärel kirjeldatakse infosüsteemi arendust ja analüüsitakse kasutatud tehnoloogiaid, tööriistu, protsesse ja meetodeid lähemalt. Lõpetuseks antakse hinnang loodud infosüsteemile ja tuuakse välja edasi arendamise võimalused.

2 Loodava infosüsteemi skoop

2.1 Loodava rakenduse eesmärk

Eesmärgiks on tagada maksimaalne väärtus kasutajale. Seega peaks loodav teenus olema kättesaadav võimalikult suurele kasutajaskonnale ja lihtne ehk intuitiivne kasutada. Kasutaja peaks tundma, et rakendus on kasumlik ja pakub teatavat väärtust. Kõik eelmainitu tagab, et kasutajad oleksid valmis rakendust üha uuesti kasutama ja tutvusringkonnas inimestele edasi soovitama. Rakenduse esialgne siht ei ole kindlasti kasumi teenimine, vaid pigem brändi ehitamine, inimeste elu lihtsustamine ja lõputöö autori enda elust tuleneva probleemi lahendamine. Õilsaks eesmärgiks on inimeste elu paremaks muutmine.

2.2 Funktsionaalsete nõuete määramine

Lahendus probleemile peaks kindlasti olema mobiilne rakendus, kuna spordi ja muude vaba aja tegevustega tegelemiseks peab olema võimalik platvormi kasutada igal ajal ja igal pool. Suurt sülearvutit kalale või tennist mängima kaasa ei võeta, kindlasti on aga mobiil alati taskus. Süsteem peaks võimaldama kasutajatel luua püsiva isikliku kasutajakonto. Kasutaja peaks konto loomisel otsustada saama missugustest tegevustest ollakse huvitatud. Seda informatsiooni kuvatakse teistele kasutajatele eesmärgiga ühiste huvidega inimesi kokku viia. Igal registreerunud kasutajal peaks olema võimalik rakenduse avades lähemalt tutvuda erinevate inimestega, kes asuvad kasutaja läheduses, teatud raadiuse ulatuses. Seega peaks süsteem pidevalt suutma tuvastada kasutaja hetke asukohta. Kasutajal peaks olema juurdepääs üles pandud kuulutustele ehk sündmustele. Neid sündmuseid peaks saama sorteerida ja neile vastata, kirjutades kuulutuse üles panijale. Kirjutada peaks saama ka neile inimestele, kes on lähiümbruses, aga pole otsest kuulutust üles pannud – seda ettepaneku tegemise eesmärgiga. Iga kasutaja peaks saama kuulutusi üles panna. Lisaks sellele peaks kasutajal olema juurdepääs kalendrile, kust enda eelolevaid sündmusi näha ja redigeerida saab. Kasutajal peaks olema võimalik rakenduse sisest kalendrit sünkroniseerida mobiiltelefoni kalendriga. Iga sündmuse

siseselt peaksid sündmusega seotud liikmed saama tööd planeerida, asju sündmuse siseselt kalendrisse märkides. Nii nagu sündmuse sisene kalender peaks olema jagatav mitme kasutaja vahel, et aega selle sündmuse ümber organiseerida, peaks olema võimalik ka suhtluskanaleid luua mitme kasutaja vahel. Kõik see hõlbustab vaba aja organiseerimist kahe inimese või grupi inimeste vahel. Eesti olusid arvestades peaks lisaks eelmainitule olema võimalik näha ka detailseid ilmastiku tingimusi teatud regioonis. Näiteks sukeldumisega tegelemisel on ülioluline, et päike paistaks ja tuulekiirus oleks madal, sest sellest oleneb, kui kaugele sukelduja vee all näeb. Samal ajal kui lohesurfi või purjetamisega tegelevad inimesed otsivad tuule suuna ja tuule kiiruse prognoose. Seega oleks kasulik, et rakendusest näeks, kas homme või paari päeva pärast on tulemas vaba aja tegevusega tegelemiseks sobivad tingimused.

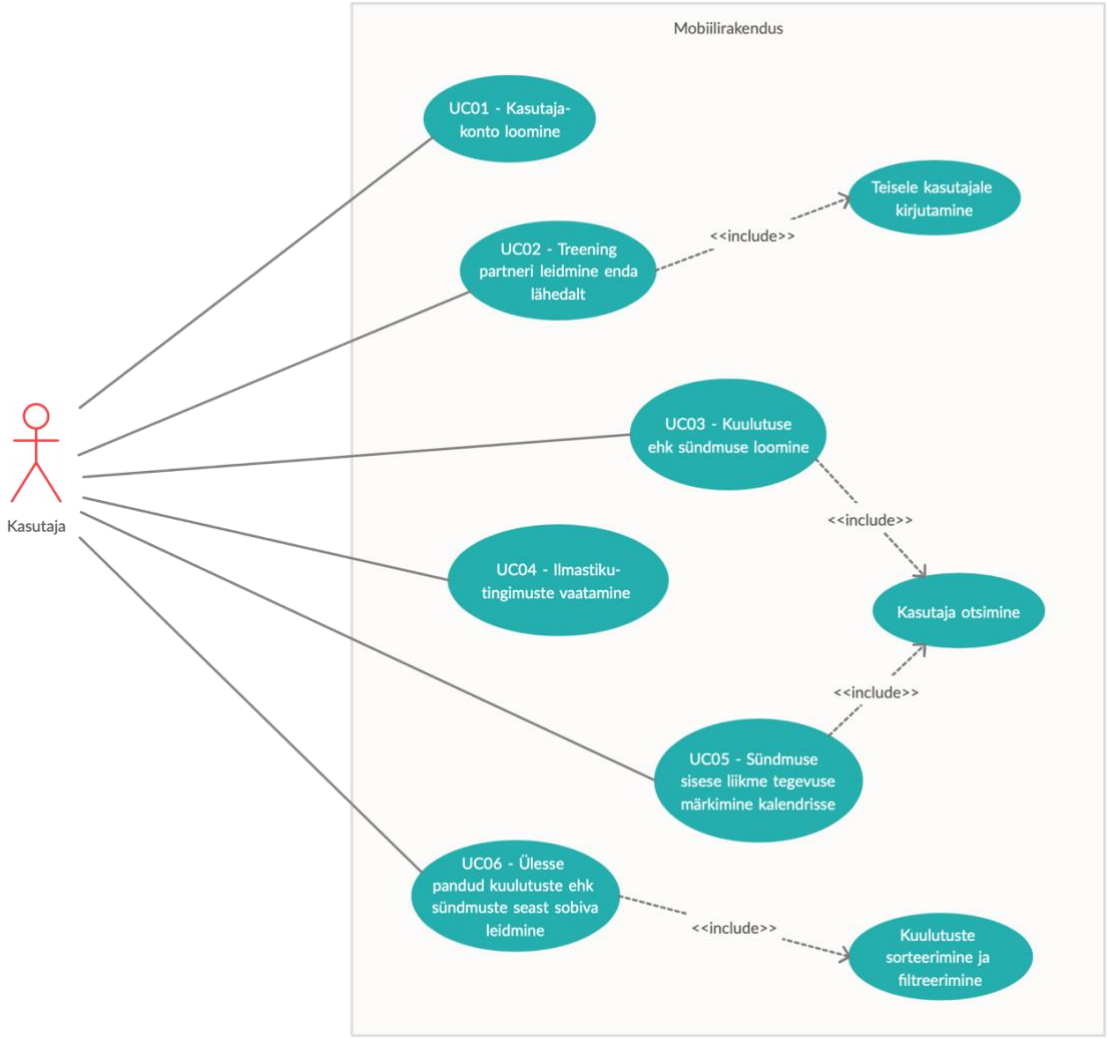
2.3 Mittefunktsionaalsete nõuete määramine

- Kasutajaga seotud andmed, peale kasutajanime ja vaba aja tegevuste huvide, on privaatsed ja pole teistele kasutajatele kätte saadavad.
- Paroolid ei tohi sisenemiskohas ega muul ajal olla nähtavad.
- Platvorm peab olema suuteline hakkama saama tuhande kasutajaga, 50 kasutajaga paralleelselt, mõjutamata selle jõudlust.
- Süsteem peab olema kohandatud värvipimedatele inimestele sel määral, et neil on kogu süsteemis kuvatav tekst ja muu teave sama hõlpsasti eristatav kui värvipimeduseta inimestel.
- Kasutajaliides peab kättesaadaval olema nii iOS kui ka Android operatsiooni-süsteemi peal.
- Tarkvara peab olema lihtsasti hooldatav.
- Tarkvara peab olema edasi arendusele mõeldes arusaadav ka teise keele kõnelejatele, mitte ainult eestlastele.
- Tarkvara peab olema lihtsasti edasi arendatav.

- Kasutajaliidese ja API rakenduse vaheline maksimaalne reageerimisaeg on kaks sekundit.
- Tarkvara peab olema kirjutatud järgides puhta koodi häid tavasid.
- Kasutaja asukohta ei avaldata teistele kasutajatele täpse asukohana, vaid umbkaudse raadiusena.
- Süsteemi seisakuid peab kolme kuu jooksul olema vähem kui 1 tund.
- Kasutajaliides peab järgima kasutajasõbraliku disaini printsiipe.
- Inimesed, kellel pole koolitust kasutajaliidese osas peavad olema võimelised toodet kasutama.
- Süsteemi ei tohi hoolduse jaoks välja lülitada rohkem kui üks kord 24-tunnise perioodi jooksul.
- Kasutajaliides peab kättesaadav olema mobiilis vaba aja tegevuste liikuva loomu tõttu.

2.4 Kasutusjuhtude mudel

Joonisel 1 on välja toodud kavandatava rakenduse kasutusjuhtude mudel.



Joonis 1. Mobiilirakenduse kasutusjuhud.

2.4.1 UC01 – Kasutajakonto loomine

Kirjeldus: Algab, kui rakendus avatakse esimest korda ja lõpeb teatisega õnnestunud kasutajakonto loomisest.

Tegutseja(d): Kasutaja

Eeltingimused: Kasutaja on mobiilirakenduse alla laadinud.

Peamine kasutajavoog:

1. Kasutaja avab rakenduse.
2. Kasutaja vajutab „Registreeru“ nupul.
3. Kasutaja sisestab vajalikud andmed.
4. Kasutaja vajutab nuppu „Loo konto“.
5. Kasutajat suunatakse „Lõpeta oma profiili loomine“ lehele.
6. Kasutaja kirjutab lühikirjelduse „Minust“ pealkirja alla ja valib loetelust tegevused, mis teda huvitavad, märkides ära tegevuse taseme – „tahan proovida“, „algaja“, „vahepealne“, „asjatundja“.
7. Kasutaja vajutab nuppu „Tehtud“.
8. Kasutajale näidatakse teadet sellest, et kasutajakonto on edukalt loodud.

Alternatiivne kasutajavoog:

6. Kasutaja ei märgi ära ühtegi tegevust kuvatavast loetelust või ei kirjuta lühikirjeldust endast.
7. Kasutaja vajutab nuppu „Tehtud“.
8. Kasutajale kuvatakse hoiatus ära täitmata väljadest.
9. Kasutaja kirjutab lühikirjelduse „Minust“ pealkirja alla ja valib loetelust tegevused, mis teda huvitavad, märkides ära tegevuse taseme – „tahan proovida“, „algaja“, „vahepealne“, „asjatundja“.
10. Kasutaja vajutab nuppu „Tehtud“.
11. Kasutajale näidatakse teadet sellest, et kasutajakonto on edukalt loodud.

Lõpetamise tulemus: Kasutaja saab mobiilirakendusse sisse logida.

2.4.2 UC02 – Treening partneri leidmine enda lähedalt

Kirjeldus: Algab pärast rakendusse sisse logimist või „Avasta“ nupule vajutusega ja lõpeb sellega, et jõutakse kellegagi kokkuleppele treeningule koos minemise osas.

Tegutseja(d): Kasutaja

Eeltingimused: Kasutaja on mobiilirakenduse alla laadinud. Kasutaja on kasutajakonto loonud.

Peamine kasutajavoog:

0.5. Kasutaja avab rakenduse.
0.6. Kasutaja vajutab „Logi sisse“ nuppu.
0.7. Kasutaja sisestab vajalikud andmed.
0.8. Toimub autentimine.
0.9. Kasutaja suunatakse rakendusse inimeste avastamise saki all olevale „Avasta“ lehele.

1. Kasutaja näeb ekraanil ühe teise kasutaja pilti, „Endast“ kirjeldust ja tegevusalasid, millest kasutaja on huvitatud.
2. Kasutaja liigub järgmise kasutaja juurde näpulükkega.
3. Seekord näeb kasutaja ekraanil inimest, kellel on temaga sama huvi ja kirjeldus vastab sellele, mida kasutaja otsib.
4. Kasutaja vajutab „Saada sõnum“ ikoonil.
5. Avaneb vestlusaken.
6. Kasutaja tervitab teist kasutajat ja teeb ettepaneku koos trenni minna.
7. Kasutaja saab vastuseks sõnumi nõusolekuga.

Alternatiivne kasutajavoog:

1. Kasutaja näeb ekraanil kirja „Kahjuks pole teie piirkonnas ühtegi vastet. Jagage rakendust sõna levitamiseks“.

Lõpetamise tulemus: Kasutaja saab luua sündmuse kokkulepitud ajaks ja lisada leitud treeningpartner kõnealuse sündmuse alla.

2.4.3 UC03 – Kuulutuse / sündmuse loomine

Kirjeldus: Algab „Loo sündmus“ nupule vajutusega ja lõpeb sellega, et „Minu sündmused“ all on uus sündmus.

Tegutseja(d): Kasutaja

Eeltingimused: Kasutaja on mobiilirakenduse alla laadinud. Kasutaja on kasutajakonto loonud ja sisse loginud.

Peamine kasutajavoog:

0.9. Kasutaja suunatakse rakendusse sisse logimisel inimeste avastamise saki all olevale „Avasta“ lehele.

1. Kasutaja vajutab „Sündmused“ sakil.
2. Kasutaja suunatakse „Sündmused“ lehele.
3. Kasutaja vajutab „Loo sündmus“ nupule.
4. Kasutaja täidab ära vajalikud andmeväljad.
5. Kasutaja vajutab nupule „Postita“.
6. Kasutaja suunatakse „Minu sündmused“ alla, kus äsja loodud kuulutus tulevasest sündmusest on näha.

Alternatiivne kasutajavoog:

5. Kasutaja otsib kasutajanimede abil välja tuttavaid inimesi, lisamaks neid sündmuse alla.
6. Kasutaja märgib ära, et sündmus on privaatne.
7. Kasutaja vajutab nupule „Postita“.
8. Kasutaja suunatakse „Minu sündmused“ alla, kus äsja loodud sündmus on näha.

Lõpetamise tulemus: Kasutaja saab enda postitatud sündmusi näha ja sündmuse alla teisi kasutajaid lisada, kes avaldavad soovi liituda. Loodud sündmuse all saavad sündmuse liikmed omavahel suhelda ja aega planeerida. Sündmus lisatakse kõigi seotud kasutajate kalendritesse ja „Minu sündmused“ alla automaatselt.

2.4.4 UC04 – Ilmastikutingimuste vaatamine

Kirjeldus: Algab „Avasta“ nupule vajutusega ja lõpeb sellega, et nähakse detailset ilmastiku ülevaadet.

Tegutseja(d): Kasutaja

Eeltingimused: Kasutaja on mobiilirakenduse alla laadinud. Kasutaja on kasutajakonto loonud ja sisse loginud.

Peamine kasutajavoog:

0.9. Kasutaja suunatakse rakendusse sisse logimisel inimeste avastamise saki all olevale „Avasta“ lehele.

1. Kasutaja vajutab „Ilm“ nupule, mis asub rakenduse ülemisel ribal.
2. Kasutaja suunatakse ilma prognoosi lehele, kus kuvatakse järgmise 7 päeva ilmaennustus.
3. Kasutaja vajutab päevale, mil tal on huvi teatud tegevusega tegeleda.
4. Kasutajale kuvatakse detailne ilmaennustus selleks päevaks.

Lõpetamise tulemus: Kasutaja on varustatud teadmisega, mis ajaks on kõige mõttekam sündmus organiseerida ja loob sündmuse vastavalt sellele.

2.4.5 UC05 – Sündmuse sisese liikme tegevuse märkimine kalendrisse

Kirjeldus: Algab „Kalender“ nupule vajutusega ja lõpeb sellega, et nähakse ülevaadet sündmusest koos kõigi selle alasündmustega ehk tööülesannetega.

Tegutseja(d): Kasutaja A, kasutaja B

Eeltingimused: Kasutajad on mobiilirakenduse alla laadinud. Kasutajad on kasutajakonto loonud. Kasutaja A on sisse loginud.

Peamine kasutajavoog:

0.9. Kasutaja A suunatakse rakendusse sisse logimisel inimeste avastamise saki all olevale „Avasta“ lehele.

1. Kasutaja A vajutab „Kalender“ sakil.
2. Kasutaja A suunatakse „Kalender“ lehele.
3. Kasutaja A vajutab kalendris olevale sündmusele.
4. Kasutaja A suunatakse sündmuse detail vaatesse.
5. Kasutaja A vajutab rakenduse paremal üleval nurgas olevale „+“ märgile.
6. Kasutaja A suunatakse „Lisa töö“ lehele.
7. Kasutaja A täidab ära vajalikud andmeväljad.
8. Kasutaja A vajutab nupule „Loo“.

9. Tööülesande külge ära märgitud kasutaja B saab teate loodud tööülesandest.
10. Kasutaja A suunatakse sündmuse detailvaatesse, kus äsja loodud tööülesanne on näha.

Alternatiivne kasutajavoog:

7. Kasutaja A ei märgi ära kasutajat, kellele käesolev tööülesanne on suunatud.
8. Kasutaja A vajutab nupule „Loo“.
9. Kasutajale A kuvatakse hoiatus ära täitmata väljadest.
10. Kasutaja A märgib ära kasutaja, kellele käesolev tööülesanne kuulub.
11. Kasutaja A vajutab nupule „Loo“.
12. Tööülesande külge ära märgitud kasutaja B saab teate loodud tööülesandest.
13. Kasutaja A suunatakse sündmuse detailvaatesse, kus äsja loodud tööülesanne on näha.

Lõpetamise tulemus: Kõik sündmusega seotud kasutajad saavad käesoleva sündmuse tööülesandeid näha ja muuta.

2.4.6 UC06 – Üles pandud kuulutuste ehk sündmuste seast sobiva leidmine

Kirjeldus: Algab „Sündmused“ nupule vajutusega ja lõpeb sellega, et nähakse detailset ülevaadet sündmusest, mis silma jäi ja huvi pakkus.

Tegutseja(d): Kasutaja

Eeltingimused: Kasutaja on mobiilirakenduse alla laadinud. Kasutaja on kasutajakonto loonud ja sisse loginud.

Peamine kasutajavoog:

0.9. Kasutaja suunatakse rakendusse sisse logimisel inimeste avastamise saki all olevale „Avasta“ lehele.

1. Kasutaja vajutab „Sündmused“ sakil.
2. Kasutaja suunatakse „Sündmused“ lehele.
3. Kasutaja sirvib üles pandud kuulutusi kerides lehekülge allapoole.

4. Kasutajale jääb silma sündmus tuleval kolmapäeval jalgpalli mängu asjus ja see on täpselt, mida kasutaja otsib.
5. Kasutaja klõpsab käesoleva sündmuse peale.
6. Kasutajale kuvatakse sündmuse detailsem ülevaade.

Alternatiivne kasutajavoog:

3. Kasutaja filtreerib välja sündmused, mis toimuvad nädalavahetuse päevadel.
4. Kasutaja sirvib üles pandud kuulutusi kerides lehekülge allapoole.
5. Kasutajale jääb silma sündmus, mis toimub laupäeval ja otsitakse jalgpallimänguks kaaslasi ning see on täpselt, mida kasutaja otsib.
6. Kasutaja klõpsab käesoleva sündmuse peale.
7. Kasutajale kuvatakse sündmuse detailsem ülevaade.

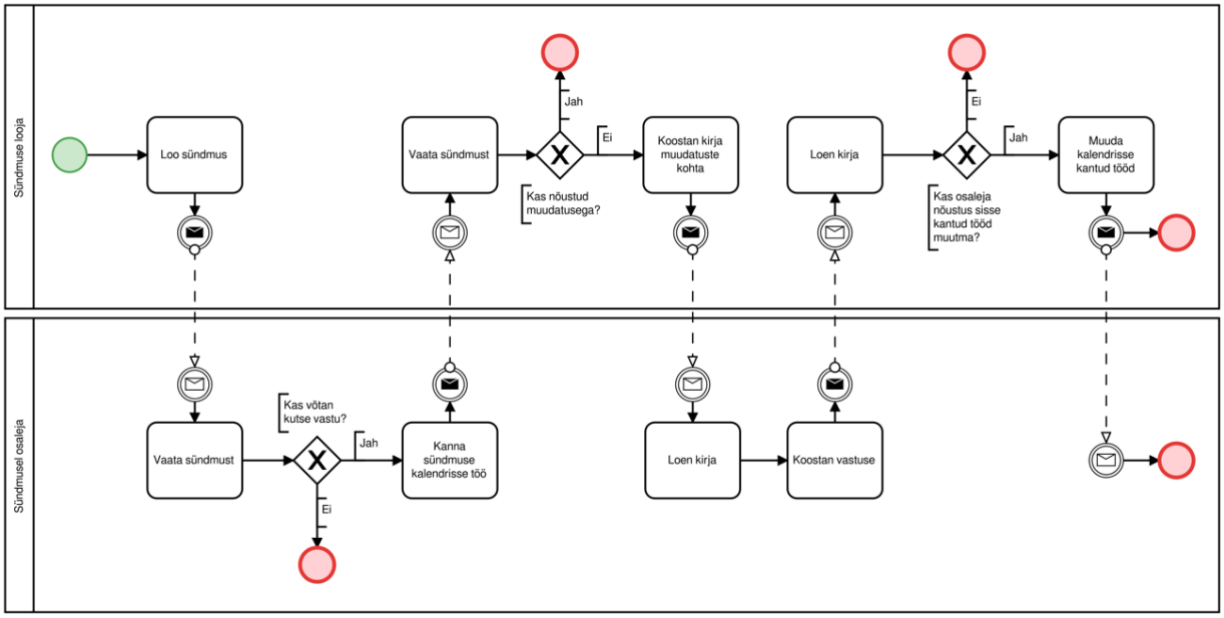
Lõpetamise tulemus: Kasutaja saab kirjutada sündmuse loojale, andmaks teada huvist liituda selle sündmusega.

2.5 Sündmuse aja planeerimine mitme kasutaja vahel

Kuna loodud sündmuse aja planeerimine sündmuse liikmete vahel toimub piiranguteta kõigi sündmusega seotud kasutajate vahel, siis on lõputöös kasutatud BPMN meetodit kõnealuse protsessi kirjeldamiseks.

BPMN ehk äriprotsesside modelleerimise märgisüsteem on skeemimeetod, mis modelleerib kavandatava äriprotsessi etappe otsast lõpuni. BPMN-i peamine eesmärk on tõhususe parandamine, uute asjaoludega arvestamine või konkurentsieelise saamine. BPMN toimib tegevuse protsessi hõlpsasti mõistetava visuaalse representatsioonina. See aitab inimestel, kes seda protsessi teostama hakkavad, saada üksikasjalikum ülevaade [1].

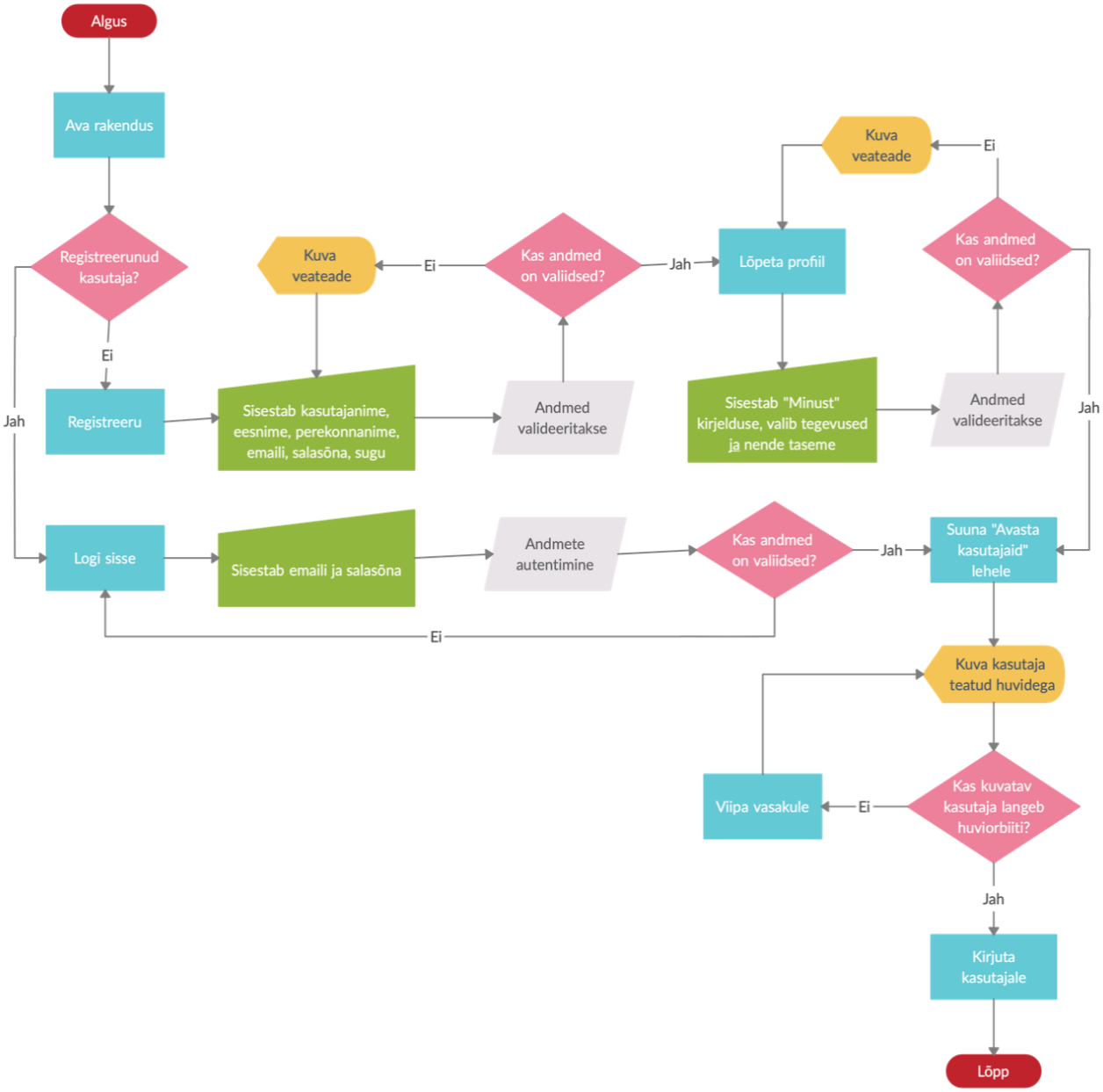
Järgnevalt on joonisel 2 välja toodud sündmuse aja planeerimise protsess kahe kasutaja vahel.



Joonis 2. Sündmuse aja planeerimise protsess.

2.6 Kaaslase leidmine teatud vaba aja tegevuse jaoks

Blokkskeeme kasutatakse paljudes valdkondades, sageli keerukate protsesside dokumenteerimiseks, uurimiseks, kavandamiseks, parandamiseks ja visualiseerimiseks selgetes ning hõlpsasti mõistetavates diagrammides. Kõige sagedamini kasutataksegi blokkskeeme uue projekti kaardistamiseks, kuna need muudavad protsessid hõlpsalt arusaadavaks. Nii on ka käesolevas lõputöös kavandatava rakenduse kaaslase leidmise protsess blokkskeemina üles märgitud. Seda võib näha joonisel 3. Protsess algab uue kasutaja registreerumisega ja lõppeb leitud kaaslasele kirjutamisega.



Joonis 3. Kaaslase leidmise protsessi illustreeriv blokkskeem.

3 Olemasolevate lahenduste analüüs

3.1 Facebook sündmused

Facebookis on sündmuste loomise funktsioon hästi üles ehitatud. Kasutaja saab valida erinevate kategooriate vahel, saab otsida sündmusi vastavalt sobivale kuupäevale või asukohale, oma sõprusringkonna siseselt või väliselt. Enamus sündmusi on aga loodud organisatsioonide või professionaalide poolt, leidmaks kliente enda organiseeritud üritustele. Isiklikke inimeselt inimesele sündmusi luuakse pigem privaatselt sõprusringkonna siseselt, näiteks sünnipäevade pidamiseks või väljasõitudeks ja ühisürituste organiseerimiseks.

Käesoleva bakalaureusetöö käigus ehitatav rakendus rõhub aga sellele, et üritusi kuulutatakse inimeselt inimesele, leidmaks uusi kaaslasi, mitte kliente, vaba aja tegevuse harrastamiseks. Ehk siis otsitakse inimesi väljaspoolt sõpruskonda, kellega sündmusi organiseerida. Ka Facebookist saab enda sõprusringkonnast väljapool olevate inimestega kontakteeruda, aga Facebookist otsides ei tea kasutaja, kes on potentsiaalselt huvitatud näiteks tennise mängimisest. Loodavas rakenduses on aga teada, et kasutajad on seal samadel eesmärkidel ja iga kasutaja konto juurde on märgitud selgesõnaliselt, mis on need tegevusalad, millest ollakse huvitatud.

Lisaks eelmainitule, ei ole Facebooki sündmuse juures võimalik aega planeerida. Uues loodavas rakenduses on aga selgelt võimalik sündmuse siseselt tööülesandeid kalendrisse märkida, organiseerimaks, kes sündmuse liikmetest mida ja mis ajal teeb. Seega on kõigil selge ülevaade iga liikme ajakavast ja planeeritavast sündmusest tervikuna. Samuti ei ole Facebookis ligipääsu ilmastikutingimuste täpsema prognoosi kohta. Facebooki siseselt on küll võimalik leida nädala ilma ülevaade, aga see sisaldab vaid temperatuuri ja lühikirjeldust – „Pilvine“, „Osaliselt pilves“. Täpsem info tuule suuna või kiiruse või sademete hulga jms kohta puudub.

3.2 Eventbrite

Eventbrite on USAst pärit ürituste haldamise ja piletimüügi veebisait. Teenus võimaldab kasutajatel sirvida, luua ja reklaamida kohalikke sündmusi. Teenus võtab ürituste korraldajatelt tasu veebipiletite müümise eest, välja arvatud juhul, kui üritus on tasuta [2].

Sarnaselt Facebooki sündmustega on kõnealuse veebilehe ja mobiili rakenduse üritused korraldajalt inimestele ja ürituse siseselt aja planeerimise funktsiooni ei ole. Üritusel toimuv ajakava kirjutatakse pika jutuna „Üritusest“ alapealkirja alla, selle asemel, et üksikasjalikult kalendrisiseselt ajagraafik välja tuua. Lisaks selle ei ole ka ilma prognoosi vaatamise võimalust sündmuse toimumise ajal, otsustamaks, kas ilmastikutingimused soosivad ürituse pidamist.

3.3 Flinder

Flinder on tõeliselt lihtne rakendus, mis soovitab asju teha vastavalt kasutaja asukohale, kellaajale ja ilmale. Horisontaalselt kerides on võimalik igasugu tegevusi näha ja kui ühe tegevuse juures seisma jääda ja vajutada nuppu „Kõlab hästi”, siis kuvatakse selle kategooria lähimate võimaluste loend. Näiteks „Mine spaasse“ puhul kuvatakse kasutajale loend lähimatest spaadest. *Flinder* on eksklusiivselt iOS toode [3], mis toetub pigem inspiratsiooni pakkumisele, andes informatsiooni edasi imetabaselt lihtsa ja elegantse kasutajaliidese kaudu. *Flinder*’i põhirõhk on ilmselgelt mitte ürituste organiseerimises ja teiste inimestega ühenduse leidmises, kuid see on kohe kindlasti eeskujuks, kui hästi saab inimesi innustada tegelema uute vaba aja tegevustega vastavalt asukohale, ilmale ja huvidele. Uus loodav lahendus peaks järgima kasutajaliidese lihtsat ja mugavat kasutuskogemust samaväärselt.

3.4 5F – Fitness Friends

5F ehk *Fitness Friends* missioon on lihtne: muuta inimesed aktiivsemaks: „Me kõik oleme seda kogenud; väljas sajab vihma, on liiga külm, liiga kuum, midagi valutab, väsinud, liiga hõivatud, pole piisavalt sportlik jne. Ehk siis treeningu vahele jätmiseks on lõputuid vabandusi ja paljud teevad seda. Kujutage aga ette, et mõni treeningsõber ootab teid treenima. Sõber, kes sinuga arvestab. Nüüd on motivatsioon üles tõusta ja sporti tegema minna palju kõrgem ning sporti on lihtsam ja lõbusam teha koos kaaslasega.

Treeningpartner on väga väärtuslik ja suurema tõenäosusega annate oma järgmisel treeningul 100% kuna motiveerite teineteist. Uue treeningrakenduse 5F abil on ülilihtne leida läheduses olevaid inimesi ja olla koos aktiivsemad [4].“

5F rakendus hõlmab teatud määral lõputöös käsitletavat lahendust probleemile, sest toetab inimeste leidmist, kes on kasutaja läheduses ja huvitatud samadest spordialadest. Kõikide lähedal olevate kasutajatega saab vabalt suhelda, aga puudub igasugune kalendri või n.ö. töö planeerimise võimalus. Lisaks ei ole ka selles rakenduses ilma prognoosi vaatamise võimalust, otsustamaks, kas ilmastikutingimused soosivad spordialaga tegelemist. *Fitness Friends* rakendus ei toetu sündmuste loomisele ja sündmuste ümber aja planeerimisele, aga kindlasti inspireerib inimesi rohkem aktiivsemad olema ja kaaslasi leidma vaba aja tegevuste harrastamiseks. Sarnaselt *Flinder*’ile on see eksklusiivselt iOS toode. Uus loodav lahendus peaks aga saadaval olema nii iOS kui ka Android kasutajatele. Androidi kasutajaid on 66.97% võrreldes iOS kasutajate 32.29 protsendiga Eestis [5], seega peaks lahendus probleemile kättesaadav olema mõlemas platvormis.

4 Tehnoloogia analüüs

Infosüsteemi arendamisel on võimalik valida paljude tehnoloogiate seast. Käesoleva diplomitöö raames on kõrvale jäetud CMS lahendused, kuna need piiravad lahenduse soovitud paindlikkust ning on loodud pigem standardlahenduste jaoks.

4.1 Mobiilirakenduste tehnoloogiate analüüs

Mobiiliplatvormide osas on vaieldamatult kaks suurt tegijat: Android ja iOS. Muud tehnoloogiad, näiteks Tizen, Blackberry või Windows Phone, on kas surnud või juba mõnda aega kasutusel olnud, aga neil ei ole mingeid väljavaateid märkimisväärse turuosa saavutamiseks. See võib eelarvamust tekitada, et arendajatel pole mobiilirakenduste loomisel palju võimalusi. Selline mõtteviis on aga tõest kaugel. Mobiilirakenduste loomisel kasutatakse palju erinevaid programmeerimiskeeli: C, C++, Java, Kotlin, Objective-C, Swift, JavaScript, TypeScript, C#, Dart, Ruby ja kindlasti ka muid [6].

Järgnevalt analüüsib autor erinevaid mobiilirakenduse arendamise võimalusi.

4.1.1 *Native* rakendused

Erinevalt veebirakendustest ei tööta *native* mobiilirakendused brauseris. *Native* rakendus on tarkvaraprogramm, mis on välja töötatud kasutamiseks konkreetsel platvormil või seadmel. *Native* rakenduse peab alla laadima platvormipõhistes rakenduste poodides, nagu *App Store* või *Google Play*. Kuna *native* rakendus on loodud kasutamiseks konkreetses seadmes ja selle operatsioonisüsteemis, siis on sellistel rakendustel võimalus kasutada seadmepõhist riist- ja tarkvara. *Native* mobiilirakendused suhtlevad otse *native* API-dega, ilma et oleks vaja vahetarkvara, näiteks pistikprogramme. Kuna sõltuvusi on vähem, on *native* mobiilirakendused kiiremad ja paremini reageerivad [7].

Native rakendused on teadaolevalt uskumatult rikka kasutajaliidesega ja usaldusväärsed. Nad on alati olemas ja kättesaadavad koduekraanidel, dokkides ja tegumiribadel. Nad töötavad sõltumata võrguühendusest ja võimaldavad kasutajale omaette kogemust. Nad saavad lugeda ja kirjutada faile kohalikust failisüsteemist, pääseda juurde USB või

Bluetooth-ühenduse kaudu ühendatud riistvarale ning suhelda isegi seadmesse salvestatud andmetega, nagu kontaktid ja kalendrisündmused. *Native* rakendustes saab teha asju, nagu näiteks pildistada, avalehel loetletud laule mängida või muus rakenduses olles laulu taas esitust hallata. *Native* rakendused tunduvad olevat seadme osana, milles nad töötavad [8] ehk siis nende kasutajakogemus on väga intuitiivne.

Native rakendused on kirjutatud samas programmeerimiskeeles, mida kasutatakse seadme ja selle operatsioonisüsteemi jaoks. Näiteks kirjutavad iOS arendajad rakendusi Objective-C või Swift keeles, samal ajal kui Androidi rakendusi luuakse Java baasil [9]. Android NDK abil on võimalik rakenduse kriitilisi osi ka otse „emakeeles“ arendada, kirjutades C või C++ keeles. Sel juhul peab aga teadlik olema platvormi iseärasustest [6]. Seega nõuab *native* rakenduste arendamine teistsuguseid oskusi ja tehnoloogiaid, kui mobiilse veebisaidi või veebirakenduse arendamine. Ei pea muretsema brauseri käitumise ja ühilduvuse pärast. Hea kasutajakogemuse pakkumiseks ja rakenduse funktsioonide täide viimiseks saab kasutada operatsioonisüsteemile omaseid iseärasusi ja funktsioone.

4.1.2 Veebirakendused

Veebirakendused on sisuliselt brauseri hallatavad rakendused. Koodi kirjutades ei peeta silmas platvormi, vaid pigem mõnda brauserit, mis selle platvormi peal töötab. Veebirakendused jooksevad veebibrauserites, seega ei pea arendajad veebirakendusi arendama mitme platvormi jaoks. Näiteks töötab üks Chrome’is töötav rakendus nii Windowsis kui ka OS X-is. Arendajad ei pea veebirakenduse värskendamisel tarkvara uuendatud versiooni kasutajatele levitama. Uuendades rakendust serveris, saavad kõik kasutajad juurdepääsu värskendatud versioonile.

Kokkuvõttes on need kõik brauseri poolt hallatavad HTML, CSS ja JavaScript lehed. *Native* API-dele, nagu kaamera, vibratsioon, aku olek või failisüsteem, puudub otsene juurdepääs, kuid mõnda neist saab kasutada veebi API-de kaudu. Veebi API-de suureks probleemiks on aga nende küpsusaste. Mõni brauser ei toeta paljusid neist ja teostamises on erinevusi, eriti erinevate mobiilibrauserite vahel.

Angular, React ja Vue on 2020. aasta seisuga tõenäoliselt populaarseimad veebiraamistikud. Täpsustuse huvides – Reacti peetakse siiski selle paindliku ja vähem nõudliku olemuse tõttu teegiks. Nimelt ei nõua React, et rakenduse struktuur järgiks

mingeid kindlaid standardeid, nagu näiteks kuidas oma faile organiseerida. React tegeleb ainult kasutajaliidese esitamisega, see ei ole niivõrd täisfunktsionaalne, kui seda on raamistikud. Angular seevastu on vägagi nõudliku olemusega raamistik. Vue elab kusagil nende kahe vahel [6].

4.1.2.1 Progressiivsed veebirakendused

Progressiivsed veebirakendused on sisuliselt tööriistade komplektid, mida kasutatakse veebirakenduse kasutajatele sama kogemuse andmiseks, mille nad saaksid mobiilirakenduse kasutamisel. Progressiivsed veebirakendused on ehitatud ja täiustatud kaasaegsete API-de abil, et pakkuda *native* rakendustele omaseid võimalusi, töökindlust ja alla laaditavust. Seega on võimalik jõuda ükskõik kelle kasutusse, ükskõik millisesse seadmesse ühe koodibaasiga. See tähendab, et veebirakendused saavad potentsiaalselt suurema levimuse kasutajate seas koos kõrgenenud kasutuse ajaga.

Alla laetud progressiivsed veebirakendused töötavad brauseri vahekaardi asemel eraldi aknas. Neid saab käivitada kasutaja avaekraanilt, dokist või tegumiribalt. Neid on võimalik seadmest otsida ja nende pealt teise rakenduse peale hüpata, muutes nad justkui tavaliseks rakenduseks või seadme osaks, kuhu nad on alla laetud.

Google on defineerinud PWA-dele kolm peamist omadust: need peavad olema usaldusväärsed, kiired ja kaasahaaravad. Kiirus on kriitiline selleks, et kasutajad saaksid rakendust hästi kasutada. Kasutajad armastavad rakendusi, mis reageerivad interaktsioonile ühe silmapilguga, ja kogemusi, millele nad võivad alati loota.

Progressiivsete veebirakenduste aluseks olevad funktsioonid on loodud töötama sõltumata seadme ühenduse olekust. See hõlmab nii võrguühenduseta režiimi kui ka halba ühendust. Samuti pakuvad need märkimisväärselt tajutavat jõudluse tõusu, kuna rakendused käivitatakse lokaalsesse vahemällu salvestatud andmeid kasutades, mis välistab sünkroonse sisu alla laadimise viivitusi. Erinevalt tavalistest veebirakendustest on üks PWA-de tuntud omadustest *Push Notifications* – hoiatuse stiilis teated, mis kuvatakse ülemisel teavitusribal või selle alal, ka lukustuskuvadel. Neil on võim meelitada kasutajad rakenduse juurde tagasi [6] [8].

4.1.3 Hübriidrakendused

Mõistet hübriid kasutatakse tavaliselt ka kõigi platvormiüleste lahenduste puhul. Hübriidrakendused on *native* ja veebirakenduste kombinatsioon. Hübriidrakenduse sisemised toimingud sarnanevad veebirakendusega, kuid see laetakse alla nagu *native* rakendus. Hübriidrakendustel on juurdepääs seadme sisestele *native* API-dele, mis tähendab, et nad saavad kasutada selliseid ressursse nagu kaamera, failihoidla, GPS jt [10].

Kui otsustada rakendust levitada rakenduse poodide kaudu, seal juures kasutamata *native* keeli või tööriistu, siis on kaks võimalust: hübriid *native* pinu kasutamine või hübriid veebi pinu kasutamine. *Native* pinu genereerib *native* koodi tõlgituna sellest keelest, milles rakendus on tegelikult kirjutatud. See omakorda käivitatakse platvormil, kasutades tavaliselt samu *native* kasutajaliidese juhtelemente. Veebi pinu kasutab aga veebimootorit elementide esitamiseks ja loogika täideviimiseks brauseris, ilma selle ümber olevate tööriistade ja otsinguribata, isegi kui see sisaldub *native* paketi sees, mida levitatakse rakenduste poodide kaudu [11].

4.1.3.1 Veebi pinu hübriidrakendused

Veebi pinu hübriidrakendused on üles ehitatud HTML, JavaScript ja CSS abil. Arendajad loovad ühe koodibaasi ja teevad seejärel väikeseid muudatusi, et kohandada rakendus igale platvormile. Veebi pinu hübriidrakendused juhivad veebirakendust tavaliselt konteineri või *WebView* kaudu – brauser, mis võib olla mobiilirakenduse sees [10].

PhoneGap on veebi pinu hübriidrakenduste platvormide seast ilmselt kõige tuntum ja veebiarendaja jaoks kõige lihtsam alustada. PhoneGap pakub *native* pistikprogramme, mis võimaldavad kasutada kõiki seadme funktsioone, sealhulgas kiirendusmõõturit, kaamerat, kompassi, failisüsteemi, mikrofoni, pilte, võrku, teatiseid, geograafilist asukohta ja faili hoidlat. Cordova on PhoneGapi avatud lähtekoodiga baas ja mootor, mida toetab Adobe. Cordova võimaldab luua brauserite üleseid mobiilirakendusi Javascripti, HTML-i ja CSS-iga. Teised tuntumad veebi pinu hübriidrakenduse tööriistad on Ionic, mis on avatud lähtekoodiga kasutajaliidese tööriistakomplekt, ja Ionicu meeskonna poolt välja lastud Capacitor [12].

4.1.3.2 *Native* pinu hübriidrakendused

Native pinu hübriidrakendused on suhteliselt uus ja sageli valesti mõistetud rakenduste kiht. Nende puhul ristuvad veebirakendused *native* komponentidega. Kuigi Appcelerator Titanium on juba pikka aega kasutuses olnud, leiduvad mõned suhteliselt uued konkurendid, kes õigustavad selle muutmist täiesti eraldiseisvaks kategooriaks mobiilirakenduste hulgas.

Rakenduse esitamiseks ja käitamiseks ei kasutata *WebView*-d. JavaScripti kompileeritakse, kuid probleem on selles, et see ei muuda JavaScripti millekski, mida Androidi või iOS-i operatsioonisüsteemid saaksid otseselt mõista. Strateegiaks on saata JavaScripti mootorid koos koodiga, tavaliselt V8 Androidi ja JavaScriptCore iOS-i puhul. See lähenemisviis on parema kasutajaliidese teostusega, kuna kõik komponendid on samad, mida ka *native* rakendus kasutaks. Mõnikord ei ole võimalik *native* pinu hübriidrakendusi eristada platvormi ametliku IDE ja keelega loodud rakendustest [6].

Native pinu hübriidrakenduste raamistike alla käivad NativeScript, Appcelerator Titanium ja Facebooki meeskonna poolt loodud React Native. Need raamistikud ei sõltu tavaliselt HTML-i märgistuskeelest. Nad tuginevad ainult JavaScriptile või TypeScriptile. Xamarin on veel üks lahendus *native* pinu lähenemisviisi elluviimiseks, kuid selle suureks erinevuseks on see, et JavaScripti ja muude veebikeelte *native* keelde kompileerimise asemel kompileeritakse see C#-st [11].

4.2 Serveripoolsete tehnoloogiate analüüs

Tavaliselt koosneb *backend* kolmest põhiosast: server, mis võtab vastu päringuid; rakendus, mis on siis kliendirakendusest eraldiseisev serveris töötav rakendus, mis kuulab päringuid, hangib andmebaasist teavet ja saadab vastuse kliendile; andmebaas, mida kasutatakse andmete korraldamiseks ja säilitamiseks [13].

Valikuvõimalus on lai *backend* rakenduse keelte osas. Tuntumateks ja kõige rohkem kasutatavamateks keelteks lõputöö kirjutamise ajal on Ruby, Python, Java, PHP, C# (.NET keskkonnad), JavaScript (Node.js) ja SQL andmebaaside haldamiseks.

4.2.1 Serveri poolne arhitektuur

Võimalusi, millist serveri poolset arhitektuuri mobiilirakenduse ehitamisel kasutada, on mitu. Saab kasutada kas traditsioonilist lähenemist, ehitades veebi API ja serveripoolne tarkvara üles nullist või ehitada serveri pool üles teenusena (MBaaS), kasutades ära pakutavaid komponente. Samuti on võimalus ehitada mobiilirakendus üles serverita [14].

4.2.1.1 Traditsiooniline veebi API

Aastaid on API-maailm kahele põhi API arhitektuurile toetunud – SOAP ja REST. SOAP mõeldi välja 1998. aastal ja REST 2000. aastal ning paljude aastate jooksul on need domineerinud kõike, mis on seotud API-dega. 2015. aastal avaldas Facebook aga uue andmepäringute keele GraphQL, mis on väidetavalt REST-i ja SOAP-i alternatiiv. Ehkki kõiki neid kolme kasutatakse API arhitektuuri loomiseks, on neil oma olemuselt mõned suured erinevused. SOAP on protokoll, REST on arhitektuuristiil, samas kui GraphQL, nagu näha selle nimes olevast QL-ist, on päringute keel (*Query Language*) [15].

Traditsioonilise kohandatud *backend*-i eelised:

- *Backend*'i saab täielikult kohandada projekti vajadustega. Kohandatud *backend* võimaldab luua sõna otseses mõttes mis tahes funktsioone. Ainulaadsed funktsioonid või kolmanda osapoole integreerimine nišiteenustega pole probleem.
- Äärmiselt eskaleeritav. Kohandatud *backend*'i saab hõlpsasti kohandada kõigi aja jooksul rakendatud muudatustega. Isegi kui rakendus kasvab, lisandub uusi funktsioone või suureneb kasutajate arv, siis saab rakenduse *backend* sellega hakkama.
- *Backend* ei sõltu teenusepakkujatest. See töötab seni, kuni otsustatakse see sulgeda. See on MBaaS-i ees väga suur eelis. MBaaS-i kasutamine nõuab, et usaldatakse täielikult välist teenust, mis võib igal ajal haihtuda [16].

Traditsioonilise veebi API puuduseks on aga see, et on vajadus serveritarkvara ja riistvarahalduse järele. Tarkvararakenduse majutamine Internetis hõlmab mingisuguse serveritaristu haldamist. Tavaliselt tähendab see virtuaalset või füüsilist serverit, mida tuleb hallata, samuti operatsioonisüsteemi ja muid veebiserveri majutamise protsesse, mis on rakenduse käivitamiseks vajalikud.

4.2.1.2 MBaaS – mobiili *backend* teenusena

Enamik mobiilirakendusi sõltub sellistest teenustest nagu API, kasutajate ja andmete haldamine, failide salvestamine, hüpik teatised, sotsiaalse meedia integreerimine ja eskaleerimine. MBaaS on see, mis majutab kõiki neid olulisi teenuseid. See on pilvepõhine andmetöötluse arhitektuur, mis pakub mobiilirakendustele juurdepääsu kõigiks tööks vajalikele ressurssidele. Üldiselt on MBaaS-i platvormi eesmärk pakkuda arendajatele vahendeid oma kliendipoolse rakenduse ühendamiseks *backend* API-dega ja pilvepõhise ladustamise süsteemiga. Selle tulemusena vabastatakse arendajad serveritega seotud ülesannetele mõtlemisest, muretsemisest, haldamisest või täitmisest. See võimaldab keskenduda serveritega tegelemise asemel parema kasutajakogemuse pakkumisele.

MBaaS-i platvorm pakub tavaliselt mitmeid põhitoiminguid, näiteks kasutajahaldust, hüpikteatiste tuge, sotsiaalvõrkude API-sid sotsiaalmeediasse sisse logimiseks ja postitamiseks jms. MBaaS-i pakkujad kasutavad kohandatud tarkvara arenduskomplekte (SDK-sid), et võimaldada arendajatel oma API-sid ühendada erinevate kasutajaliideste külge nagu iOS ja Androidi rakendused, mis on ehitatud ükskõik millise mobiilirakenduse arenduse tehnoloogia abil, nagu näiteks React Native, Ionic, Flutter, Unity jne. Need SDK-d võivad sisaldada ka juurdepääsu teenusepakkuja eelmonteeritud API-dele, näiteks sisse logimise API-d, hüpikteatiste API-d ja andmesideteenuste API-d. Samuti ei pea muretsema selle pärast, kuidas *backend*'i eskaleerida või kas *backend* töötab ilma tõrgeteta igal ajahetkel – kõiges selles saab loota teenusepakkujale.

MBaaS-teenuste pakutavate funktsioonide sügavus ja kvaliteet võib aga olla väga erinev. Mõni võib pakkuda ülal nimetatud minimaalseid funktsioone, jättes ülejäänud *backend* arendajatele, näiteks API loomine ning andmebaasi arendamine ja haldamine. Rohkem funktsioonirikkad pakkujad annavad arendajatele tööriistad API-de ehitamiseks, testimiseks ja juurutamiseks MBaaS-i platvormis endas [17] [18].

4.2.1.3 Serverita *backend*

Serverivaba arhitektuur on tuntud ka kui FaaS ehk funktsioon teenusena. See on tarkvara kujundamise muster, kus rakendusi majutab kolmanda osapoole teenus. See välistab arendaja vajaduse serveritarkvara ja riistvarahalduse järele. Rakendused on jaotatud üksikuteks funktsioonideks, mida saab eraldi käivitada ja individuaalselt eskaleerida.

Virtuaalserveri kasutamine pilveteenuse pakkujalt nagu Amazon või Microsoft tähendab küll füüsilise riistvaraga seotud probleemide kõrvaldamist, kuid nõuab siiski operatsioonisüsteemi ja veebiserveri tarkvara protsesside teatavat juhtimist. Serverita arhitektuuri puhul keskendutakse puhtalt rakenduse koodi üksikutele funktsioonidele [19]. Pilveteenuse pakkuja käitab serverit ja haldab dünaamiliselt ressursside jaotust. Eskaleerimine, mahutavuse planeerimine ja hooldustoimingud võivad arendaja eest olla peidetud. Hind kujuneb rakenduse tarbitud ressursside tegelikul hulgal, mitte eelnevalt ostetud mahutavusühikutel.

Serverita andmetöötlus võib lihtsustada koodi tootmisesse laskmist. Serverita koodi saab kasutada ka koos koodiga, mis on kasutusele võetud traditsioonilistes stiilides, nagu näiteks mikroteenused. Teise võimalusena saab rakendused kirjutada puhtalt serverivabadeks, kasutamata igasugust ette nähtud serverit [20].

FaaS-i peamised omadused on järgmised:

- See on sõltumatu ja serveripoolne ning sellel on loogilised funktsioonid. FaaS sarnaneb funktsioonidega, mida on arendajad harjunud programmeerimiskeeltes kirjutama; väikesed, eraldi loogikaühikud, mis võtavad sisse argumente, töötlevad sisendit ja tagastavad tulemuse.

- Olekuta. Serverita *backend*-i puhul on kõik olekuta. Mistahes sama funktsiooni kaks kutsumist võivad käia täiesti erinevatel konteineritel. Üks ei tea teisest midagi.

- Lühiajaline. FaaS on loodud kiireks käivitamiseks, oma töö tegemiseks ja siis uuesti sulgemiseks. Need ei seisa kasutamata. Pärast toimingu täitmist, lammutatakse selle aluseks olevad konteinerid ära.

- Sündmuste poolt käivituvad. Kuigi funktsioone saab otse käivitada, käivituvad need tavaliselt pilveteenuste sündmustest, nagu näiteks HTTP-päringud, uute andmete sisestused andmebaasi või sissetulevate sõnumite teatised.

- Eskaleeritav. Olekuta funktsioonide abil saab mitu konteinerit initsialiseerida, võimaldades just nii palju funktsioone käitada, et kogu sissetulevate päringute hulk saab pidevalt teenindatud.

- Täielikult pilve tarnija hallata. Kõige tuntumad FaaS-lahendused on AWS Lambda, Azure Functions, IBM OpenWhisk ja Google Cloud Functions. Iga pakkuja toetab tavaliselt mitmesuguseid keeli ja käitusaegu [21].

4.2.2 Serveripoolse rakenduse keeled ja raamistikud

4.2.2.1 Ruby

Ruby on interpreteeritud ja kõrgetasemeline programmeerimiskeel. Selle disainis ja arendas 1990ndate keskel välja Jaapanist pärit Yukihiro „Matz“ Matsumoto. Ruby on dünaamiliselt tüübitud ja kasutab *garbage collection*'it. See toetab mitut programmeerimise paradigmat, sealhulgas protseduurilist, objektorienteeritud ja funktsionaalset programmeerimist. Looja sõnul mõjutasid Ruby't Perl, Smalltalk, Eiffel, Ada, Basic ja Lisp [22].

Ruby'l on funktsioonid, mis on sarnased Smalltalki, Perli ja Pythoni omadega. Perl, Python ja Smalltalk on skriptikeeled. Kuid Ruby süntaksi kasutamine on palju lihtsam, kui näiteks Smalltalki süntaksi kasutamine. Ruby on avatud lähtekoodiga ja veebis vabalt saadaval, kuid selleks on vaja litsentsi. Ruby abil saab kirjutada CGI skripte ja seda saab HTML-i sisse panna. Ruby'l on puhas ja lihtne süntaks, mis võimaldab uuel arendajal seda väga kiiresti ja hõlpsalt õppida, sellel on sarnane süntaks paljude programmeerimiskeeltega, nagu näiteks C++ ja Perl. Ruby on lihtsalt eskaleeritav ja Ruby'ga kirjutatud suured programmid on hõlpsasti hooldatavad. Ruby't saab kasutada Interneti- ja sisevõrgurakenduste arendamiseks ja seda saab hõlpsalt ühendada DB2, MySQL, Oracle või Sybase'i andmebaasiga [23].

4.2.2.2 Python

Python on Guido van Rossumi poolt välja töötatud programmeerimiskeel, mis ilmus esmakordselt 1991. aastal. See on avatud lähtekoodiga kasutajasõbralik ja hõlpsasti kasutatav keel, millel on rõhuasetus koodi loetavusel. Python on tuntud lihtsa süntaksi ja lühikese koodipikkuse poolest. Python on dünaamiliselt trükitud keel. Pythonis kirjutades ei pea muutujate tüüpe määrama, kuna interpretaator järeldab need tüübid ise ja käitluse ajal tehakse vastavad kontrollid. Selle tulemuseks ongi lihtsam süntaks, mis on üsna sarnane inglise keelega. Lisaks ei kasuta Python ümbritsevaid sulgusid ja järgib taande reegleid, mis muudab koodi üsna hõlpsasti loetavaks ja algajatele sõbralikuks [24]. Selle loetavus muudab sujuvamaks ka sama projekti kallal töötavate

arendajate suhtluse. See tähendab, et kui mõni teine arendaja töötab koodi hilisemate täienduste kallal, ei tohiks neil probleeme olla algse koodi mõistmisega. Lisaks on Python platvormist sõltumatu keel, mis tähendab, et Pythoni abil loodud tarkvara saab kasutada paljudes erinevates operatsioonisüsteemides ilma interpretaatori vajaduseta.

Kuigi Pythonit peetakse võrreldes teiste serveripoolsete programmeerimiskeeltega, nagu näiteks Java või C++, aeglaseks, pole see asjaolu Pythoni kasvu aeglustanud. Seda tõestab ka Elektri- ja Elektroonikainseneride Instituudi (IEEE) programmeerimiskeelte edetabel, kus on Python tõusnud edetabelite tippu, olles populaarsuselt esikohal. Selle populaarsus ei piirdu ainult IT-tööstusega, Pythonit kasutatakse laialdaselt ka akadeemilistes ringkondades. Pythoni populaarsus tähendab, et sellel on suur ja äärmiselt aktiivne kogukond, kes on aidanud üles ehitada arvestatava arvu spetsialiseeritud teeke ja raamistikke, mis pakuvad lisatööriistu heaks ja kiireks arendustegevuseks [25].

Kuna Python on süntaktiliselt väga lihtne, kuid samas täieõiguslik programmeerimiskeel, on see saanud populaarseks valikuks erinevate erialade inimeste vahel, kes soovivad katsetada masinõppega ja tuua AI võimsust enda vastavatesse valdkondadesse. Sellepärast toimub suur osa AI arendamisest ja masinõppest just Pythoni abil, koos selle tohutu ökosüsteemi ja teekidega. Sinna hulka kuuluvad TensorFlow, Keras, Sickit-Learn ja Facebooki PyTorch. See on vaieldamatult kõige populaarsem keel masinõppe valdkonnas.

Kaks kõige populaarsemat Pythoni veebiarenduse raamistikku on aga Django ja Flask. Flask on mikro veebiraamistik, mis tagab vajalikud põhifunktsioonid nagu päringute marsruutimine. Django on aga rohkem kiidetud valik ja aitab luua võimsaid serveripoolseid programme, kasutades ära selle raamistiku pakutud tõhusust ja turvalisust. Django on varustatud võimsa ORM-kihiga, mis hõlbustab andmebaasidega suhtlemist ja andmetega mitmesuguste toimingute tegemist [24].

4.2.2.3 Java

Java on staatiliselt trükitud ja objektorienteeritud programmeerimiskeel. Java on mõeldud olema WORA – kirjuta üks kord, käita ükskõik kus – keel. See on Java virtuaalmasina (JVM) abil mõeldud töötama mis tahes platvormil võimalikult väheste sõltuvustega. Paljude arenduskeelte korral loob programmi kompileerimine koodi, mis võib erinevalt töötada erinevate arvutite peal. JVM-i tõttu pole see aga Java jaoks probleem. Java

virtuaalmasin toimib nagu keskmine vahekiht, mis suudab koodi käitada igas arvutis, olenemata sellest, kus nimetatud kood kompileeriti. Seega on Java väga mitmekülgne ja selle kasutus ulatub nutitelefonidest nutikaartideni. Seda on arendajad kasutanud üle 20 aasta [26].

Ehkki Java on laua- ja äritarkvara arendajate seas äärmiselt populaarne, on see vähem algajasõbralik keel, kui Python. Java järgib rangeid süntaksireegleid, mis on staatiliselt trükitud keele omapära. Muutujate tüübid peab selgesõnaliselt deklareerima ja seda ei tohiks mingil juhul märgata jätta, sest vastasel juhul kood ei kompileeri. Kuigi see pole algajatele just kõige lihtsam asi, leiavad mõned arendajad staatiliselt trükitud keelte selgusega lohutust, kuna suurte koodibaaside korral ei tunne nad end taande reeglite järgimisel mugavalt [24].

Java toetab selliseid põhimõtteid nagu andmete abstraktsioon, polümorfism, kapseldamine, *overloading* ja pärimine. *Backend* veebiarendajate sõnul muudab see keele sama võimsaks kui seda on C++. Java API pakub kõike, mida üks edukas programm peab teha oskama; sisendi/väljundi käitlemine, võrgustike loomine, utiliidid, päringute haldamine, andmebaasi ühendamine, XML-i analüüsimine ja lugemine, andmete ja aja käsitsemine jms. Java avatud lähtekoodiga teeke on umbes kümme tuhat. Apache Commons, Spring MVC, Hibernate on mõned neist. Samuti on Java ökosüsteemis nii mõnigi silmapaistev tööriist ja raamistik, nagu näiteks Eclipse IDE, Maven ja Spring – kõige tuntum Java serveripoolse arenduse raamistik. Ükskõik, kas tegemist on Windowsi, Androidi või iOS-i nutitelefoniga, Java-põhiseid rakendusi on kõikjal [27].

4.2.2.4 PHP

PHP on laialt kasutatav avatud lähtekoodiga skriptikeel, mis sobib staatiliste ja dünaamiliste veebisaitide või veebirakenduste arendamiseks ja mida saab HTML-i sisse panna. Skriptide eesmärk on tavaliselt rakenduse jõudluse parandamine või tavapäraste toimingute tegemine. HTML-i väljutamiseks mõeldud käskude asemel, nagu on C-s või Perlis, koosnevad PHP-lehed HTML-ist, kuhu sisse on sängitatud PHP kood. PHP-kood on suletud spetsiaalsetesse algus- ja lõppjuhistesse „*<?php*“ ja „*?>*“, mis võimaldavad hüpata „PHP-režiimist“ sisse ja välja.

Mis eristab PHP-d millestki sellisest, nagu seda on kliendipoolne JavaScript, on see, et kood käivitatakse serveris, genereerides HTML-i, mis seejärel saadetakse kliendile.

Klient saab HTML-i skripti käitamise tulemusena, kuid ei tea, mis oli selle aluseks olev kood. See tähendab, et PHP peab installitud olema ainult serverisse ja serverilt ressursse taotlevatel kliendi arvutitel ei pea PHP installitud olema, piisab ainult veebibrauserist.

Parim asi PHP kasutamisel on see, et see on äärmiselt lihtne, kuid pakub professionaalsele arendajale palju lisafunktsioone [28]. PHP on platvormide ülene; see tähendab, et PHP-s kirjutatud rakendust saab juurutada paljudes erinevates operatsioonisüsteemides, nagu Windows, Linux, Mac OS jne. PHP-l on sisseehitatud MySQL-i tugi, kuid see ei tähenda, et ei saaks kasutada ka teisi andmebaasihaldussüsteeme. PHP-d värskendatakse regulaarselt, et olla kursis kõige uuemate tehnoloogiatrendidega [29].

4.2.2.5 .NET keskkonnad

Enam kui 10 aastat tagasi avalikkusele tutvustatud .NET raamistik on funktsioonirikas ja põhjalikult n.ö. lahingu testitud. Kui .NET-i algusaegadel oli tavaline, et *native* arenduse ühendamine hallatava koodiga oli vajalik, siis toetatakse valdavat osa arendusülesannetest praegu valmislahendusena. Isegi sellised ettevõtted nagu Oracle on oma toodetega liidendamiseks välja lasknud komponendid, mis on 100-protsendiliselt .NET-i hallatav kood.

.NET API on järjepidev, hästi dokumenteeritud ja seda kasutavad miljonid. Kuigi Microsofti serveripoolne tarkvara on jäänud peaaegu muutumatuks, kasutatakse endiselt samu mustreid ja komponente, nagu näiteks *Dependency Injection*, *Tasks*, *Linq*, EF või ADO, nägi siiski kliendipoolne osa ehk ASP.NET põhimõttelist nihet "tee seda Microsofti moodi" pealt "tee seda oma viisil ja kasuta platvormina ASP.NET-i" peale. Täna on ASP.NET MVC-põhine raamistik, mis sisaldab tugevat autentimise, komplekteerimise ja marsruutimise infrastruktuuri ning integreerub paljude Microsofti alla mittekuuluvate tehnoloogiatega, nagu näiteks Bootstrap või AngularJS. ASP.NET-i veebilehed näevad kenad välja paljudes eri vormides, alates telefonidest kuni lauaarvutiteni, ja selle veebi API võimalused muudavad veebiteenuste arendamise imelihtsaks. Raamistik on mitu aastat avatud lähtekoodiga olnud, nii et probleemide taha takerdudes on allikas vabalt GitHubis saadaval [30].

Peamine erinevus Java EE ja .NET vahel on see, et Java võib töötada mis tahes operatsioonisüsteemis, samal ajal kui .NET töötab ainult erinevates Windowsi versioonides. Mõlemad platvormid pakuvad tuge tavalistele programmeerimiskeeltele

nagu PHP, Ruby ja Python. Java arendajad saavad kasutada ka selliseid keeli nagu Java, JavaScript, Clojure, Groovy ja Scala, samal ajal kui .NET arendajad saavad programmeerida C#, F#, VB.NET ja C++ keeles. J2EE raamistik seab vaikeväärtusena keeleks Java, samas kui .NET ühildub keeltega nagu C#, F# ja VB.NET. See on paindlikum lähenemisviis ärirakenduste arendamiseks, ehkki see nõuab mitmekesisemaid programmeerimise oskusi [31]. .NET-i abil serveripoolsete rakenduste loomiseks on Microsofti poolt toetatud kaks rakendust: .NET Framework ja .NET Core. Mõlemal on palju samu komponente ja koodi saab jagada ka nende kahe vahel. Neil on aga põhimõttelisi erinevusi ja valik ühe või teise kasuks sõltub sellest, mida soovitakse saavutada [32].

4.2.2.6 Node.js

Node.js on rakenduste käituskeskkond, mis võimaldab serveripoolseid rakendusi JavaScriptis kirjutada. Tänu oma ainulaadsele sisend-väljund mudelile paistab see silma just sellistes eskaleeritavates ja reaalaja olukordades, mida serveritelt üha enam nõutakse. See on kergekaaluline ja tõhus ning selle võime JavaScripti kasutada nii kliendi- kui ka serveripoolel avab uusi arenemisvõimalusi. JavaScript on olnud üks populaarsemaid kliendipoolseid programmeerimiskeeli ja üks kõige sagedamini kasutatavaid *frontend* veebiarenduse tööriistu [33]. JavaScript, mis on peaaegu sama vana kui Java, on interpreteeritud keel, mida on kliendi poolel kasutatud pikka aega enne, kui ilmus Node.js. Node.js oli esimene tööriist, mis võimaldas arendajatel kasutada JavaScripti serveripoolel oma „JavaScript kõikjal" kontseptsiooniga. Node.js ilmus esmakordselt 2009. aastal ja on nüüdseks avatud lähtekoodiga kogukonna projekt. Kui JavaScripti kasutatakse nii serveri- kui ka kliendipoolel, on koodi vajadusel lihtsam migreerida. Kuna kasutatakse sama keelt, töötab algselt brauserile loodud loogika serveris minimaalsete kohandustega [34].

Node.js on tänu oma asünkroonsele mitteblokeerivale sisendi/väljundi töötlemisele ja sündmustele orienteeritud olemuse tõttu hea valik kaasaegsete lahenduste loomiseks, mis põhinevad mikroteenustel, sündmuste järjekordadel, tööülesannetel ja *WebSocket*'il. Node.js sobib suurepäraselt keerukate rakenduste jaoks, kuid tarkvara korral, mis nõuab suurte arvutuste käitamist, võib see toimida vähem tõhusalt. Suured arvutused blokeerivad sissetulevaid päringuid, mis võib vähendada jõudlust [35]. Node.js on loodud palju aastaid hiljem kui Python, Java või PHP, mis muudab selle kaasaegsemaks ega nõua

ühilduvuse säilitamist eelmise ajastu lahendustega. Node.js paketihaldur NPM teeb tarkvaraarenduse lihtsamaks ja kiiremaks, kuna see võimaldab kasutada teiste arendajate loodud avatud lähtekoodiga pakette [36]. Kuna aga enamasti on tegemist avatud lähtekoodi ökosüsteemiga, siis pole arvukate tööriistade üle järelevalvet teostatud ning seetõttu võib nende kvaliteet puudulik olla ja programmeerimisstandardeid mitte täita. Node.js teegid ja mustrid on vähem stabiilsed ning küpsed, kui mõne vanema programmeerimiskeele või raamistiku omad. Mõne arendaja poolt võib puuduseks pidada ka asjaolu, et Node.js pole standardiseeritud, seal pole stiili osas palju ettekirjutusi või nõudeid. Node.js-ga peab põhimõtteliselt kirjutama kõik nullist. See võib põhjustada tootlikkuse langust, aeglustades tööd [35].

4.2.3 Andmebaaside valiku võimalused

Andmebaaside analüüsi aluseks on võetud kaks andmebaasi mudelit: relatsioonilised ja mitte-relatsioonilised. Mitte-relatsiooniliste andmebaaside tehnoloogiate rohkuse tõttu on kirjeldatud ka selle alla kuuluvaid peamisi kategooriaid.

4.2.3.1 Relatsioonilised andmebaasid

E.F Coddi poolt 1970. aastal määratletud relatsiooniandmebaas on digitaalne andmebaas, mis põhineb andmete relatsioonimudelil. MySQL, PostgreSQL või SQL Server on suurepärased relatsioonandmebaasi näited. Nende nimed sisaldavad ka vihjet: andmetele juurdepääsemiseks ja nendega manipuleerimiseks peab teadma SQL-i (*Structured Query Language*). SQL-il on väljakujunenud standardid ja see võimaldab andmeid hõlpsalt töödelda. Andmeid hoitakse tabelites, mis sisaldavad ridu ja veerge. Iga rida tähistab individuaalset kirjet ja veerg tähistab välja, millele on omistatud konkreetne andmetüüp. Tabeleid, mis sisaldavad seotud teavet, saab siduda primaarsete ja võõrvõtmetega [37] [38].

Relatsioonilised andmebaasid on kõige levinumad ja laialdasemalt kasutatavad – seda ka mõjuvatel põhjustel. Mudel, ehkki paindumatu, on usaldusväärne, väldib andmete dubleerimist ja võimaldab arendajatel töötada kindlate ja ennustatavate tulemustega. Lisaks sisaldab see mudeli tüüp paljusid kõige küpsemaid saadaolevaid andmebaase, näiteks PostgreSQL või MySQL. Sellest tulenev stabiilsus ja turvalisus on tõenäoliselt mis tahes andmebaasi valimisel kõige olulisem tegur. Relatsioonilisel andmebaasil põhineva lähenemisviisi peamine eelis on võime luua tabelite liitumisel

tähenduslikku informatsiooni. Tabelite ühendamine võimaldab mõista seoseid andmete vahel või seda, kuidas on tabelid omavahel seotud. SQL sisaldab võimalust päringute kaudu andmeid kokku lugeda, lisada, grupeerida ja ka ühendada. SQL suudab täita põhilisi matemaatika ja vahesumma funktsioone ning loogilisi teisendusi. Analüütikud saavad tulemusi järjestada kuupäeva, nime või mis tahes veeru järgi. Need omadused muudavad relatsioonilise lähenemise tänapäeval kõige populaarsemaks päringutööriistaks ettevõtluses.

Relatsiooniandmebaasidel on teiste andmebaasivormingutega võrreldes mitmeid eeliseid:

- Paindlikkus. SQL-il on sisseehitatud keel tabelite loomiseks nimega *Data Definition Language* (DDL). DDL võimaldab lisada uusi veerge, lisada uusi tabeleid, seoseid ümber nimetada ja muid muudatusi teha isegi andmebaasi töötamise ajal ja päringute ajal. See võimaldab muuta skeemi või seda, kuidas modelleeritakse andmeid.

- Vähendatud liigrohkus. Relatsioonilised andmebaasid välistavad andmete liiasuse. Näiteks teave ühe kliendi kohta kuvatakse ühes kohas – üks kanne kliendi tabelis. Tellimuste tabel peab salvestama ainult lingi kliendi tabelisse. Liiasuse vältimiseks andmete eraldamise tava nimetatakse normaliseerimiseks.

- Varundamise ja taastamise lihtsus. Relatsioonilised andmebaasid on transaktsioonilased, tagamaks, et kogu süsteemi olek on igal hetkel terviklik. Enamik relatsiooniandmebaase pakub lihtsaid ekspordi ja impordi võimalusi, muutes varundamise ja taastamise triviaalseks. Ekspordid võivad toimuda isegi andmebaasi töötamise ajal, muutes rikke taastamise lihtsaks [39].

4.2.3.2 Mitte-relatsioonilised andmebaasid

Mitte-relatsiooniliste andmebaaside kasutus on samuti märkimisväärselt tõusnud. Selle peamiseks põhjuseks on kasvav vajadus struktureerimata andmesalvestuse järele. Suure hulga andmete ajastul peame sageli tegelema teabe mitmekesisusega. Nüüd võivad andmed tähendada ka pilte, videoid ja isegi postitusi sotsiaalmeedia võrkudes. Tabeliväliste andmetega töötamiseks on vaja mitte-relatsioonilisi andmebaase. Arendajad nimetavad neid mõnikord NoSQL-i andmebaasideks. Erinevalt relatsioonilistest, ei toeta nad SQL päringuid.

On olemas nelja tüüpi andmebaase, mis ei kasuta relatsioonimudelit. Valiku põhjal saab andmeid salvestada dokumentide, võtmeväärtus paaride, graafikute või veergude perekondadena [38].

4.2.3.2.1 Võti-väärtus andmebaasid

Kasutavad põhiandmemudelina assotsiatiivset massiivi, nagu seda on sõnastik ehk *dictionary* või kaart ehk *map*. Andmed on esitatud võtme-väärtus paaride kogumina ja võti kuvatakse kogumikus maksimaalselt üks kord. Talletada saab väärtusi täisarvuna, sõnena, JSON-i struktuurina või massiivina, koos võtmega, mida kasutatakse sellele väärtusele viitamiseks. Arendajad kasutavad võti-väärtus andmebaase enamasti siis, kui nende käsitletavad andmed pole liiga keerulised ja kiirus on prioriteediks. Näiteks on see suurepärane valik konfiguratsiooniandmete salvestamiseks. Salvestatud andmetele ei omistata skeemi ja andmebaas ise on relatsioonilise andmebaasiga võrreldes palju kergekaalulisem. Alates 2019. aastast on populaarseim võti-väärtus andmebaas Redis [37] [38].

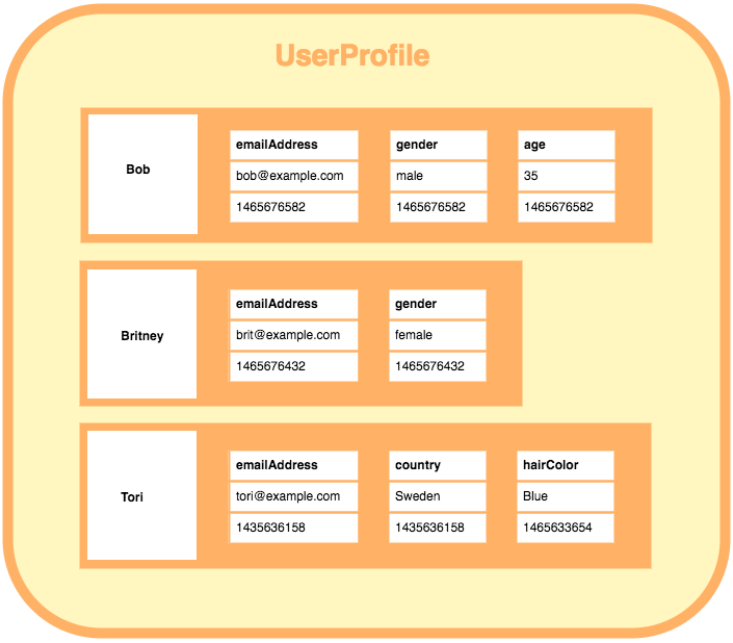
4.2.3.2.2 Graafandmebaasid

Graafandmebaas kasutab andmete esitamiseks ja salvestamiseks graafistruktuure. See võimaldab kasutajatel liikuda kiiresti kõigi ühendatud väärtuste vahel ja leida suhetest ülevaadet. Graafandmebaasi kasutades on käsitsemiseks kahte tüüpi andmeid. Sisuüksused (*nodes)* tähistavad andmebaasi üksusi ja servad määratlevad nende seoseid. Kõigist andmebaasitüüpidest on see parim variant juhul, kui seosed ja nende analüüs on esmatähtsad. Graafilistel andmebaasidel on aga üks selge puudus: kuigi andmetele on juurdepääsuks vaja päringukeelt, ei saa kasutada ei SQL-i ega ka mõnda muud üldtunnustatud lähenemisviisi. Standardiseerituse puudumine tähendab, et enamikku päringukeeli saab kasutada ainult ühes või vähestes graafandmebaasides [37] [38].

4.2.3.2.3 Veeruhoidla põhised andmebaasid

Veeruhoidla andmebaasid kasutavad mõistet *keyspace*. *Keyspace* on nagu skeem relatsioonimudelis. See sisaldab kõiki veeruperekondi, mis sisaldavad ridu ja veerge andmete salvestamiseks ja korraldamiseks. Veergude perekond koosneb mitmest reast. Tabelitega võrreldes on siiski selge erinevus: veerg ei ulatu üle kõigi ridade. Selle asemel sisaldub see rea sees, mis tähendab ka seda, et erinevatel ridadel võivad olla erinevad

veerud. Näidet veeruperekonnast võib näha joonisel 4 [40]. Lisaks veergudele on igal real ka identifikaator, mida nimetatakse võtmeks ja igas veerus on nimi, väärtus ja ajatempel. Mis teeb sellest hea valiku suurte andmehulkade haldamiseks, on kiire jõudlus, tõhus andmete pakkimine ja suur eskaleerimise võime [37] [38].



Joonis 4. Veeruperekonda illustreeriv näide.

4.2.3.2.4 Dokumendile orienteeritud andmebaasid

Dokumentidele orienteeritud andmebaas ehk dokumendihoidla on arvutiprogramm, mis on loodud dokumentidele orienteeritud teabe ehk poolstruktureeritud andmete hoidmiseks, otsimiseks ja haldamiseks. Andmeid hoitakse dokumendikogus, kasutades tavaliselt JSON-, XML- või BSON-vorminguid. Ühes kirjes võib olla nii palju andmeid, kui vaja, mis tahes eelistatud andmetüübis või -tüüpides. Piirangud puuduvad. Ühel dokumendil on teatud sisemine struktuur, kuid see võib dokumentide vahel erineda. Dokumente saab ka üksteise sisse panna.

Kõigist mitte-relatsioonilistest andmebaasitüüpidest on kõige populaarsemad just dokumendihoidlad. Parimaks näiteks on MongoDB, millel on praegu kogu maailmas üle 400 miljoni alla laadimise. Esialgu tutvustati seda 2008. aastal ja nüüdseks kasutavad seda sellised tööstushiiglased nagu Barclays ja Bosch [37] [38].

5 Tehnoloogia valik lahenduse elluviimiseks

Valikute tegemisel on arvesse võetud järgmisi tegureid:

- Projekti nõuded, probleemi domeen. Valitud tehnoloogia peaks sõltuma probleemist, mida lahendatakse. Mõnda asja saab paremini teha ühes keeles, kui teises. Igal tehnoloogial on oma ärikasutus ja seda eelist peaks ära kasutama. Näiteks .NET on mitmekülgse kommerts veebirakenduse loomiseks eelistatud valik. Kui tahetakse teha suurandmete analüüsi, näiteks ilmaolude prognoosimist, võib Python olla parim valik. Kui rakenduse ettevõtte tasemel turvalisus on kriitiline, peaks tõenäoliselt mõtlema Java kasutamisele [41].

- Tehniline kompetentsus. Tehnoloogia valik peaks sõltuma arendaja tehnilisest võimekusest. Uue tehnoloogia õppimise asemel on eelistatav kasutada tehnoloogiaid, mida arendaja tunneb.

- Ajapiirang võrreldes rakenduse suuruse ja keerulisusega.

- Turvalisus. Turvalisust ei tohiks kunagi unarusse jätta. Sõltuvalt sellest, milliste andmetega infosüsteemis töötatakse, võib turvalisus isegi kõige olulisem tegur olla. Tehnoloogiate valimisel tuleb olla ettevaatlik, kuna paljud pole piisavalt turvalised.

- Eskaleerimise võimalused, tuleviku muudatused. Asjad muutuvad palju kiiremini, kui 20 aastat tagasi. Pikka aega töötasid inimesed peamiselt lauaarvutites ja Windowsiga. Järgmise 10 aasta jooksul pole see tõenäoliselt nii. Valitud tehnoloogia peaks jääma kasutusse vähemalt 5–10 aastaks: see on pikk aeg ja ei saa olla kindel, kuidas asjad selle aja jooksul arenevad. Sellepärast peab olema valmis tehnoloogiate muutmiseks, kui selleks peaks vajadus tekkima. Seega peab arvestama, et valitud tehnoloogia pinu tööriistad võimaldaksid eskaleerimist. Eskaleerida saab nii vertikaalselt, lisades uute toimingute jaoks täiendavat tarkvara, kui ka horisontaalselt, töötlemisüksuste või füüsiliste masinate lisamisega oma serverisse või andmebaasi. Tuleb mõelda kohandatavusele,

väikeste pakendite peale ja huvide eraldamisele. Teenused, serveripool ja kasutajaliidesed tuleks eraldada väiksemateks rakendusteks ja mikroteenusteks. Sel juhul saab tehnoloogiaid hõlpsalt ümber vahetada.

- Kogukonna tugi ja dokumentatsioon. Kui suur on keele või raamistiku taga olev ökosüsteem? Mitu *Stackoverflow* küsimust, konverentsi ja veebipõhist õpetust on kõnealuse tehnoloogia kohta? Kas huvi tarkvara vastu kasvab endiselt? Mida suurem on kogukond, selle tugi ja dokumentatsioon, seda suurem on tõenäosus, et tehnoloogia püsib pikka aega ja arendamise käigus tekkivatele küsimustele leiab vastuse.

5.1 Serveripoolne tehnoloogia

Uue loodava serveripoolse rakenduse puhul on eelkõige kasutusele võetud traditsioonilise veebi API lähenemine. Seda seetõttu, et lõputöö autoril on õpingutest tulenevalt traditsioonilise API arendamise kogemus ja see annab võimaluse infosüsteemi täielikult vajadustele kohandada ning hiljem vajadusel laiendada või muudatusi sisse viia. Samuti tagab see kõrgendatud turvalisuse taseme, kuna arendajal on suurem kontroll nii loodud rakenduse üle kui ka andemete üle, mida rakendus käsitleb. Tarkvara juurutamisel on aga mõistlik suuremal või vähemalt määral pilvetarnijate teenuseid kasutada, et serveritega seotud muredest vähem sõltuv olla.

5.1.1 Suhtlusprotokolli valik

Rakenduste vahelise suhtluse meetodiks üle veebi on valitud REST, mis võimaldab andmeid HTML, JSON, XML või lihtteksti kujul üle kanda, samas kui SOAP lubaks ainult XML-i saata. SOAP-i kasutatakse pigem ettevõtte tasemel tarkvara arenduseks, mis vajab suurt turvalisust, nagu näiteks finants ettevõtte teenused. REST-i arhitektuur on paindlikum. Kuna see koosneb ainult lõtvadest juhistest, siis on arendajal võimalik neid soovitusi omal moel rakendada. REST-i API-sid on lihtne üles ehitada ja laiendada, kuid need võivad olla ka massilised ja keerulised – kõik sõltub sellest, kuidas nad on üles ehitatud, millele lisatud ja mida nad on loodud tegema. Põhjused, miks võiks luua REST API, hõlmavad ressursipiiranguid, vähem turvanõudeid, brauseri kliendi ühilduvus ja andmete terviklikkus. Kõik eelmainitu kehtib veebiteenuste osas ja seda on ka uus loodav rakendus. REST API-liidesed põhinevad URI-del ja HTTP-protokollil ning

REST võimaldab kasutada JSON-it andmevormingu jaoks, mis ühildub brauseriga [42]. Sellele lisaks on RESTful veebiteenused parema jõudlusega ja bakalaureusetöö autoril on kogemus just seda tüüpi suhtlusprotokolliga.

5.1.2 Programmeerimiskeele valik

Serveripoolse keele osas on valitud Java, kuna Javaga seotud abivahendid on avatud lähtekoodiga ning tasuta kasutatavad. Java ökosüsteemi kuuluvad mitmed tööriistad ja raamistikud, mis teevad arendustöö lihtsamaks ja kiiremaks. See on objektorienteeritud, mis võimaldab koodi mooduliteks eraldada ja korduv kasutada. Lisaks on see platvormist sõltumatu, tehes võimalikuks hõlpsalt ühest arvutisüsteemist teise liikuda. Javal on hea kogukonna tugi, dokumentatsioon ja lõputöö autor tunneb ennast kõige kindlamalt just selles keeles.

5.1.3 Andmebaasi valik

Andmebaaside osas on valik langenud relatsioonilise andmebaasi kasuks, kuna relatsioonilised andmebaasid on rakendusi juba aastaid usaldusväärselt teeninud, nad on enim kasutatud ehk suure kogukonna toega ja pakuvad funktsioone, mis on tänapäeval endiselt kriitilised. Nad on küpsed ja hästi mõistetavad, seega suuresti usaldatud andmebaaside valik. Relatsioonilise andmebaasi puhul on võimalik andmetele lihtsalt juurde pääseda standardiseeritud keele SQL kaudu. Seega saab andmeid manipuleerida ja teiste kasutajatega koostööd teha, teisi sõnu: relatsioonilised andmebaasid on paindlikud. Lisaks puudub autoril kogemus mitte-relatsiooniliste andmebaasidega täielikult.

Järgnevalt on välja toodud mõned tuntumad relatsioonilised andmebaasid:

- SQLite. See on iseseisev, failipõhine ja täielikult avatud lähtekoodiga RDBMS, mis on tuntud oma teisaldatava loomu, töökindluse ja tugeva jõudluse poolest, isegi vähese mäluga keskkondades. Selle tehingud vastavad ACID-le isegi siis, kui süsteem jookseb kokku või toimub elektrikatkestus. SQLite'i kirjeldatakse mõnikord ka kui nullkonfiguratsiooni andmebaasi. Ehkki SQLite'i andmebaasidele pääseb juurde ja saab päringuid teha mitme protsessi kaudu, saab andmebaasis muudatusi teha igal ajahetkel ainult üks protsess. See tähendab, et SQLite toetab suuremal määral samaaegset andmete käsitlemist, kui enamik

teisi sisseehitatud andmebaaside haldussüsteeme, kuid mitte nii hästi nagu kliendi/serveri RDBMS-id, nagu seda on MySQL või PostgreSQL. Kuna SQLite loeb ja kirjutab otse arvuti kettafaili, on ainsad rakendatavad juurdepääsuõigused selle aluseks oleva operatsioonisüsteemi tüüpilised juurdepääsuõigused. See teeb SQLite'ist halva valiku rakenduste jaoks, mis vajavad mitut spetsiaalsete juurdepääsuõigustega kasutajat [43].

- MySQL. See on funktsioonirikas toode, mis annab jõudu paljudele maailma suurimatele veebisaitidele ja rakendustele, sealhulgas Twitter, Facebook, Netflix ja Spotify. MySQL-iga alustamine on suhteliselt lihtne, suuresti tänu põhjalikule dokumentatsioonile, suurele arendajate kogukonnale ja MySQL-iga seotud ressursside rohkusele veebis. MySQL on loodud kiiruse ja töökindluse tagamiseks täieliku SQL standardi järgimise arvelt. Otsustades mitte rakendada teatud SQL-i funktsioone, said MySQL-i arendajad kiirust prioriteediks seada. MySQL tuleb koos skriptiga, mis aitab parandada andmebaasi turvalisust sellega, et lubab määratleda juurkasutaja parooli, eemaldades anonüümsed kontod ja eemaldades testandmebaasid, mis on vaikimisi kõigile kasutajatele juurdepääsetavad. Lisaks, erinevalt SQLite'ist, toetab MySQL kasutajahaldust ja võimaldab juurdepääsuõigusi anda kasutajalt-kasutajale. MySQL-i duplikaat-tugi võib hõlbustada horisontaalset laiendamist. Lisaks sellele on üleminek kommertslikuks MySQL-i tooteks suhteliselt sirgjooneline protsess [43].

- PostgreSQL. Seda kutsutakse kõige arenenumaks avatud lähtekoodiga relatsiooniliseks andmebaasiks maailmas. See loodi eesmärgiga olla väga laiendatav ja standarditele vastav. PostgreSQL on objekt-relatsiooniline andmebaas, mis tähendab, et kuigi see on peamiselt relatsiooniline andmebaas, sisaldab see ka funktsioone, nagu näiteks tabeli pärilikkus ja funktsioonide *overloading*, mida sagedamini seostatakse objekt-andmebaasidega. Postgres on võimeline tõhusalt toime tulema korraga mitme ülesandega. PostgreSQL loob iga uue kliendiühenduse jaoks uue protsessi. Igale uuele protsessile on eraldatud umbes 10MB mälu, mis võib paljude ühenduste korral kiiresti suureks kasvada. PostgreSQL on tavaliselt väiksema jõudlusega, kui teised RDBMS-d, nagu näiteks MySQL. PostgreSQL on MySQL-ist populaarsuse poolest ajalooliselt maha jäänud. Selle tagajärjel on endiselt vähem kolmanda osapoole tööriistu, mis aitaksid PostgreSQL andmebaasi hallata [43].

- MariaDB. Avatud lähtekoodiga relatsioonilise andmebaasi haldussüsteem, mis on laialt levinud, MySQL andmebaasi tehnoloogiaga ühilduv ja asendatav. See loodi MySQL-i tarkvaraharuna arendajate poolt, kes mängisid algse andmebaasi üles ehitamisel võtmerolli. Nad koostasid MariaDB 2009. aastal vastuseks sellele, et Oracle Corp. omandas MySQL-i. Võib öelda, et MariaDB on MySQL-i täiustatud versioon, kuna see sisaldab olemasoleva SQL-i mitmesuguseid optimeeritud funktsioone. MariaDB põhineb SQL-il ja toetab ACID-tüüpi andmetöötlust, tagades tehingute järjepidevuse, eraldatuse ja vastupidavuse. Muude funktsioonide hulgas toetab andmebaas ka JSON API-sid, paralleelset andmete replikatsiooni ja mitut ladustusmootorit [44].

Võttes arvesse, et ehitatav rakendus vajab erinevate kasutajagruppide ja juurdepääsuõiguste rakendamist, kui ka teatavat turvalisuse taset arvestamaks kasutajate isiklike andmete hoiustamisega, siis serverita andmebaas nagu SQLite ei ole kindlasti heaks valikuks kõnealuse rakenduse puhul. Oma populaarsuse, kasutusmugavuse, turvalisuse ja ka kiiruse tõttu on parimaks valikuks MySQL andmebaas. MariaDB-l, mis on MySQL-i täiendatud versioon, puuduvad MySQL-iga võrreldes mõned MySQL 5.6 funktsioonid ja kõik MySQL 5.7 funktsioonid. MySQL-i eeliseks on see, et suurettevõte arendab ja toetab seda. Nad edendavad MySQL-i mastaapsust ja funktsioone kiiremini, kui teised loetletud valikud. Seda on lihtne käitada ja hooldada, see on oma olemuselt küps ja hästi dokumenteeritud.

5.2 Kliendipoolne tehnoloogia

Võrdlemaks *native*, hübriid ja veebirakenduse lähenemisviise kliendipoolse rakenduse ehitamiseks, on arvesse võetud 7 erinevat otsustamise aspekti ja selle põhjal võrdlev tabel koostatud. Tabelis 1 on rohelise taustaga tähistatud plusse, punasega miinuseid ja halli taustaga neutraalseid külgi.

Tabel 1. Mobiiliarenduse tehnoloogiate võrdlus.

	Native	**Hübriid**	**Veeb**
Koodi taaskasutatav	Ühele platvormile arendatud kood töötab ainult selle platvormi jaoks	Enamik hübriid tööriistadest võimaldavad ühe koodibaasi ülekandmist teistele peamistele mobiilplatvormidele	Ainsateks probleemideks on brauseri ühilduvus ja jõudlus
Juurdepääs seadmele	Platvormi SDK võimaldab juurdepääsu kõigile seadme API-dele	Olenevalt tööriistast pääseb juurde paljudele veebirakenduste jaoks suletud seadme API-dele	Pääseb juurde vaid mõnele seadme API-le, näiteks geograafilisele asukohale, kuid nende arv kasvab
Kasutajaliidese disain	Platvorm sisaldab tuttavaid, originaalseid kasutajaliidese komponente	Kasutajaliidese raamistike abil on võimalik saavutada üsna loomulik (*native*) välimus	Kasutajaliidese raamistike abil on võimalik saavutada üsna loomulik (*native*) välimus
Levitamine	Rakenduste kauplused pakuvad turunduseeliseid, kuid neil on ka nõudeid ja piiranguid	Rakenduste kauplused pakuvad turunduseeliseid, kuid neil on ka nõudeid ja piiranguid	Väljalaskel pole piiranguid, kuid rakenduste poe eelised puuduvad
Jõudlus	*Native* koodil on otsene juurdepääs platvormi funktsionaalsusele, mille tulemuseks on parem jõudlus	Keerukate rakenduste puhul takistavad abstraktsioonikihid sageli *native* jõudluse saavutamist	Jõudlus põhineb brauseril ja võrguühendusel
Rahaline tulu	Rohkem rahalise tulu teenimise võimalusi, kuid rakenduste poed võtavad protsendi	Rohkem rahalise tulu teenimise võimalusi, kuid rakenduste poed võtavad protsendi	Ei vaja poodide komisjoni tasusid ega seadistuskulusid, kuid raha teenimise meetodeid on vähe
Arendamise lihtsus	Kolmest valikust kõige raskem, eriti kui arendatakse mitme platvormi jaoks. Suur õppimiskõver	Sarnaneb veebi arendamisele, kuid hübriid tööriistade jaoks on vaja lisaoskusi	Kõige lihtsam ühe koodibaasi ja väikese õppimiskõvera tõttu – tehnoloogiad kattuvad varem õpituga

Tabelist on näha, et hübriid mobiiliarenduse puhul on kõige vähem miinuseid. Sellest tulenevalt on kliendipoolse rakenduse arenduseks valitud hübriidne lähenemine. Nagu on eelnevalt analüüsis välja toodud, võib hübriidarendusi jaotada veebi pinu hübriidrakendusteks ja *native* pinu hübriidrakendusteks.

Veebi pinu kasutamisel on oma eelised: saadaval on palju avatud lähtekoodiga raamistikke, veebi ja rakenduse vahel on võimalik koodi jagada ning saab ära kasutada lõputöö autori praegust veebikeelte oskust. Siiski ilmneb ühilduvuse, jõudluse ja mälukasutusega seotud probleeme. Kuna veebi pinu mobiilirakenduste arendusraamistiku käsutuses on ainult *WebView*, siis on ka arendaja seotud *WebView* piirangutega [10]. Kasutajate jaoks on oluline rakenduse loomulik välimus ja hea kasutajakogemus. Kuna aga veebi pinu hübriidrakendused on tavaliselt lihtsalt veebirakendused, siis võivad mõned asjad kasutajatele alguses imelikud tunduda. Probleemideks võivad olla visuaalse tagasiside puudumine puutepiirkondades ja ekraani kerimisvõimalused, mis ei tundu nii sujuvad, kui *native* rakenduste puhul, kuna puutesündmuste viivitus on 300ms. Ehkki kõigile neile küsimustele leidub lahendusi, et veebi pinu rakendus tunduks võimalikult ligilähedane *native* rakendusele, siis peab selleks tegema täiendavaid pingutusi. Cordovas näiteks pole juurdepääsu *native* juhtelementidele. *Native* välimuse ja tunde loomise huvides on kaks võimalust: saab luua *native* juhtelemendid, näiteks nupud ja sisestuselemendid, ise HTML-i ja CSS-iga, või tuleb rakendada *native* mooduleid, mis pääsevad neile *native* juhtelementidele ligi [45].

Native hübriid pinu lähenemisviis, mis kasutab samuti veebitehnoloogiaid, kompileerib rakenduse *native* koodiks ja ei kasuta *WebView*'d vaadete esitamiseks. *Native* pinu lahendused loovad rakendusi, mida ei ole mõnikord võimalik eristada platvormi ametliku IDE ja keelega loodud rakendusest. Seega on kliendirakenduse loomiseks valitud just *native* hübriid pinu lähenemisviis.

5.2.1 Platvormide üleste *native* hübriidraamistike võrdlus

Järgnevalt on välja toodud 5 kirjutamise ajal kõige tuntumat *native* hübriid pinu arendusraamistikku, mis toetavad platvormide ülest arendust. Seejärel on põhjendatud kliendipoolse raamistiku valikut.

5.2.1.1 Titanium

Titanium on täielikult avatud lähtekoodiga platvorm, mille on välja töötanud ettevõte nimega Appcelerator, mis on omakorda osa Axway'st. Titaniumi ja selle CLI abil saab JavaScriptiga luua *native* välimusega mobiilirakendusi. Esimeseks kihiks on JavaScript, mis räägib Titanium SDK-ga. Titanium SDK on omakorda ehitatud Objective C-ga või Androidi puhul Javaga ja see siis loob vastutasuks arendaja poolt loodud komponentidele *native* elemendid. Tulemuseks on 100% *native* ekraanikomponent. See käitub nagu operatsioonisüsteemile omane aken ja näeb välja nagu *native* aken. Tegelikkuses ongi see *native* komponent; esialgne kood oli olemas puhtalt selleks, et luua *native* element. JavaScripti kompileeritakse koos rakendusega, seega on rakenduse algne pakett suurem, kui see vastaval operatsioonisüsteemi platvormil on. See ei mõjuta rakenduse toimimist aga mingil viisil [46].

5.2.1.2 React Native

React on JavaScripti teek, mis kuulub ametlikult Facebookile. React Native on osa Reactist, mis töötab selle võimsa mobiiliraamistiku all. React Native'i abil saab luua ehtsaid *native* mobiilirakendusi, kus React Native'i komponendid teisendatakse *native* iOS ja Androidi komponentideks JavaScripti ja Reacti raamistiku kasutamise tulemusena. Algselt töötas Facebook välja React Native'i, et toetada ainult iOS-i, kuid oma hiljutise Androidi operatsioonisüsteemi toega saab raamistik nüüd mõlema platvormi jaoks mobiilseid kasutajaliideseid luua. React Native'it on äärmiselt lihtne kasutama hakata, eriti arendajatele, kes tunnevad JavaScripti või Reacti varasemalt [47].

5.2.1.3 NativeScript

NativeScript on avatud lähtekoodiga raamistik rakenduste arendamiseks Apple iOS ja Androidi platvormidele. Algselt töötas selle välja Progress. NativeScripti rakendused on ehitatud JavaScriptiga või mõne muu keelega, mis JavaScriptiks ümber tõlgendub, nagu näiteks TypeScript. NativeScript toetab JavaScripti raamistikke nagu Angular ja Vue. NativeScriptiga loodud mobiilirakendused annavad tulemuseks täiesti *native* rakendused, mis kasutavad samu *native* API-sid just kui oleksid need välja töötatud Xcode'is või Android Studios [48].

5.2.1.4 Xamarin

Tarkvarahiiglase Microsofti poolt omandatud Xamarin on üks intelligentsemaid raamistikke, mida mobiilarendajad kasutada saavad. Selle suureks erinevuseks on see, et JavaScripti ja muude veebikeelte *native* mobiilkeelde kompileerimise asemel kompileeritakse see C#-st. See tähendab, et .NET-i arendajad saavad oma oskusi kasutada ja platvormide üleseid rakendusi luua Androidi, Windowsi ja iOS-i jaoks eelistatavast ökosüsteemist lahkumata. Mis veel teeb Xamarini ainulaadseks on see, et tal on olemas vahendid Java, C++ ja Objective-C teekide kaasamiseks. Xamarini teekide kogum on tohutu. Raamistik vähendab kulusid ja põhjustab minimaalseid eelarveprobleeme. Xamarin konkureerib sageli React Native'iga kõrgeima koha nimel. Põhjuseks on see, et arendajad otsivad loomingulisi viise hübriidrakenduste loomiseks, mis tasakaalustaksid võrdses proportsioonis loovuse ja eelarve. On teada, et see annab *native* rakendusega võrdse jõudluse ja kasutajakogemuse. Oma *native* kasutajaliidese juhtelementidega näitavad Xamarini rakendused absoluutset võimekust. Hiljuti pakkusid nad välja uue arendusmeetodi, mis võimaldab koodi jagada platvormide vahel. See on aidanud arenduskiirust parandada ja hooldust lihtsustada. Arendajad saavad ühist loogikat kasutada, kuid kasutajaliidese osad jäävad iga platvormi jaoks individuaalseks. Xamarin ei tegele keeruka graafikaga, vaid sobib pigem lihtsate rakenduste jaoks. See tehnoloogia valitakse tavaliselt ettevõtlusele suunatud projektide jaoks. Xamarini ametlikul foorumil ja ka kolmandate osapoolte veebisaitidel on kogukonna tugi tohutu [49].

5.2.1.5 Flutter

Google'i poolt 2018. aasta veebruaris *Mobile World Congress*'il välja kuulutatud Flutter on kaasaegne arenduskomplekt, mis on loodud mobiilirakenduste ehitamiseks Androidi, iOS-i ja Google Fuchsia jaoks. Flutter on avatud lähtekoodiga rakenduste arendamise raamistik, mis põhineb Dart programmeerimiskeelel. Üks suurepäraseid asju Flutteri juures on *hot reload*. See tähendab, et arendajad näevad rakenduses kõiki koodimuudatusi koheselt. Selle mõte on hoida rakendus töös ja sisestada töö ajal muudetud failide uusi versioone. See võimaldab arendajatel kiiresti uusi detaile lisada, vigu parandada ja katsetada kasutajaliidese elementidega. Flutteril on rikkalik komponentide komplekt ja ulatuslikud võimalused keerukate kohandatud komponentide loomiseks, mis on rakenduse vaate ja liidese jaoks äärmiselt oluline. Kuna Flutteri taga on suur nimi, saab loota arendajate kogukonna toetusele. Selle miinused on aga piiratud teekide arv ja pidev integratsioon [50].

5.2.2 Kliendipoolse raamistiku valik

Kliendipoolse raamistiku valikul elimineeris autor esialgu Flutteri ja Xamarini, kuna nende puhul oleks arendustegevus käinud kas täiesti uue programmeerimiskeelega või keelega, milles töö autor ennast nii tugevalt ei tunne. Tänu varasemale töökogemusele ja õpingute käigus omandatule on autoril aga JavaScripti ja Reacti kogemus. Võrreldes ülejäänud kolme platvormi, siis igaühel on omad plussid ja miinused ning on palju asju millega näiteks React Native saab paremini hakkama, kui Titanium, ja vastupidi. Seega oli kliendipoolse raamistiku valik tulenev puhtalt sellest, mida autor soovis juurde õppida ja milles kogemust saada – React Native.

5.3 Arenduskeskkonna valik

Arenduskeskkonnaks võib pidada protsesside ja programmeerimisriistade kogumit, mida kasutatakse programmi või tarkvaratoote loomiseks. Mõiste võib mõnikord tähendada ka füüsilist keskkonda. IDE ehk integreeritud arenduskeskkond on protsesside ja tööriistade koordineeritud kogum, et pakkuda arendajatele arendusprotsessi korrapärast liidest ja mugavat vaadet [51]. See on rakendus, mis pakub arendajatele tarkvara arendamiseks terviklikke võimalusi. IDE koosneb tavaliselt lähtekoodi redaktorist ning ehituse automatiseerimise ja tarkvara silumise tööriistadest. Enamik IDE-sid võimaldavad ka versioonihaldusega integreerimist, mis võimaldab salvestada ja hallata arendatava rakenduse mitut versiooni. Versioonihaldus on mõeldud arendajate töö koordineerimiseks, kuid seda saab kasutada ka lihtsalt muutuste jälgimiseks, mis tahes failide komplektis. Kui midagi valesti läheb, saab võrrelda erinevaid koodiversioone ja naasta mõne eelmise versiooni juurde. See võimaldab ühtset pilvepõhist asukohta, kus hoida oma koodi ja annab võimaluse näha, kuidas projekt on arenenud [52].

5.3.1 Versioonihaldustarkvarad

Versioonihaldustarkvaradest on võrdluseks välja toodud laialt levinud ja interneti avarustes enim mainitud koodihalduse keskkonnad. Seejärel on põhjendatud versioonihaldustarkvara valikut.

5.3.1.1 GitHub

GitHubil on suurepäraseid funktsioone, nagu väljaannete jälgija, vigade jälgimissüsteem, lähtekoodide haldus ja oma sisseehitatud CI/CD tööriist, mis ilmus 2019. aasta novembris. Github pakub ka *wiki*'sid, märgenduskeelel põhinevaid README faile ja muud dokumentatsiooni. Juba teisel aastal jõudis Github 100 000 kasutajani koos 135 000 avaliku hoidlaga. Microsoft soetas 2018. aastal Githubi 7,5 miljardi dollari eest. Sellega vähenes Githubi aktiivsete kasutajate arv. Just siis sai GitLab üsna populaarseks ja temast sai Githubi tugev konkurent. Vaatamata sellele on Github endiselt kõige populaarsem git-hoidlate platvorm maailmas. Githubil on 2020. aasta jaanuari seisuga üle 40 miljoni kasutaja ja 100 miljon hoidlat [53].

5.3.1.2 Gitlab

GitLabi asutasid Dmitriy Zaporožets ja Valeri Sizov 2011. aastal Ukrainas. Dmitriy vajas tõhusat koostöövahendit. Ta teadis, et projektide edukaks lõpule viimiseks on oluline suhtlus. GitLabil on enamasti olemas kõik samad funktsioonid, mida Github pakub. GitLab kavandati algselt sisseehitatud CI/CD tööriista abil, mis teeb sellest tänapäeval ühe populaarseima CI/CD raamistiku. GitLab võimaldab koodi täielikult jälgida, et näha, kas on midagi, mida saaks parandada. Seega on GitLab funktsionaalsuse osas ees Githubist. GitLab kasvas 2016. aastal märkimisväärselt ja enam kui 100 000 organisatsiooni koos miljonite kasutajatega kasutasid GitLabi. 2018. aasta mais kolis GNOME üle 400 projekti ja 900 kaasautoriga GitLabi [53].

5.3.1.3 Bitbucket

Bitbucket on osa Atlassiani tarkvarakomplektist, nii et seda saab integreerida teiste Atlassiani teenustega, nagu HipChat, Jira või Bamboo. Seda saab juurutada nii kohalikus serveris, ettevõtte andmekeskuses kui ka pilves. Bitbucket võimaldab luua tasuta ühendust kuni viie kasutajaga. Seega saab enne ostu otsuse tegemist platvormi tasuta proovida. Bitbucket toetab arendusprojekte, mis kasutavad kas Mercuriali või Giti versioonihaldussüsteeme. 2019. aasta augustis avaldas aga Bitbucket plaanid lõpetada Mercuriali hoidlate toetamise alates 1. juunist 2020. See plaan hõlmab kõigi Mercuriali jälgede eemaldamist Bitbucketi API-st ja kõigi Mercuriali, nii avalike kui ka privaatsete, hoidlate kustutamist jäädavalt [54].

5.3.1.4 Mercurial

Mercurial on hajutatud versioonihaldus tööriist tarkvaraarendajatele. Seda toetatakse Microsoft Windowsis ja Unixi-laadsetes süsteemides, näiteks FreeBSD, macOS ja Linux. Mercuriali peamised disaini eesmärgid hõlmavad suurt jõudlust ja eskaleeritavust, detsentraliseerimist, täielikult hajutatud ühisarendust, nii lihtteksti kui ka binaarfailide vastupidavat käsitsemist ning kõrvalharude hargnemis- ja liitmisvõimalusi, jäädes samas kontseptuaalselt lihtsaks. See sisaldab integreeritud veebiliidest [55].

5.3.1.5 Subversion

Peamine ja ehk kõige tuntum erinevus on selle põhiarhitektuuris. Git versioonihaldus on hajutatud, samal ajal kui Subversion on tsentraliseeritud. Tsentraliseeritud versioonihaldusarhitektuuri nagu Subversion puhul võib öelda, et see on efektiivne meeskondade ja arendajate vahelise sünkroniseerimise tagamisel. Kui töötada Subversioniga, peab sünkroniseerima erinevused ainult selle vahel, mis on lokaalselt olemas ja kõige uuemana serveris. See on palju kiirem võrreldes Gitiga. Giti hajutatud olemuse tõttu peab alla laadima kõik muudatused, isegi kui need muudatused lisavad gigabaite kasutuid faile, mis on tegelikkuses selleks ajaks harust juba kustutatud. Giti hajutatud olemus tagab aga, et koodi saab peahoidlast sõltumatult arendada ja versioonihaldushoidlasse kõrvalharuna üles panna. Subversioni harud luuakse peahoidla sees kataloogidena. See tähendab konflikte, puuduvaid faile ja segatud muudatusi, kui mitu arendajat korraga muudatusi lisavad. Ometi on see osade ettevõtete lemmik valikuks; nad peavad seda väärtuslike andmetega usaldusväärseks [56].

5.3.1.6 CVS ehk C*oncurrent Versions System*

CVS on üks vanimaid versioonihaldussüsteeme ja tuntud tööriist nii kommertslike kui ka avatud lähtekoodiga arendajate seas. See võimaldab kontrollida koodi, millega plaanitakse töötama hakata, ja seejärel võimaldab muudatusi sisse viia. Sellel on võime käsitleda mitme haruga projekte, et meeskonnad saaksid oma koodimuudatused ühendada ja projekti ainulaadseid funktsioone lisada. Kuna CVS on juba pikka aega eksisteerinud, on see kõige küpsem versioonihaldus tarkvara [54].

5.3.1.7 IBM Rational ClearCase

ClearCase sarnaselt Subversionile on tsentraliseeritud, samas kui Git on hajutatud. ClearCase'is peab rakendama metaandmeid iga faili jaoks, ükshaaval. Kui on 100 000

või enam faili, siis nende metaandmete käsitsi rakendamine on tülikas. Peab looma kirje iga faili jaoks. See oli ehitatud tuhandete failide ajastul, mitte ajastul, kus on 100 000 kuni miljon faili koodihoidla kohta. Gitil on parem jõudlus lõppkasutajat silmas pidades. Kuid Giti arhitektuur tekitab probleeme ettevõtte tasandil. Ettevõtted vajavad ühte nii öelda tõeallikat. ClearCase'i hoidlad on versioonitud objektide alused (VOB). Ehkki meeskonnal võib olla mitu suurt VOB-i, peetakse neid kõiki monoliitse koodialuse osaks. ClearCase'i kõrvalharude tekitamine toimub failide kaupa. Haru haruga liites, käib ühendamine faili kaupa [57].

5.3.1.8 Versioonihaldustarkvara valik

Versioonihaldustarkvarana on käesoleva projekti raames kasutusele võetud Githubi platvorm, selle hajutatud ja detsentraliseeritud olemuse tõttu ning suure kasutajaskonna ja seega suure kogukonna toe tõttu. Lõputöö autoril on õpingutest tulenevalt küll kogemus erinevate Giti versioonihoidlatega, kuid Github on see, mida autor isiklikke arendusi läbi viies igapäevaselt kasutab.

5.3.2 IDE-d ehk integreeritud arenduskeskkonnad

Järgnevalt on võrdluseks välja toodud tuntumad Java ja React Native'i integreeritud arenduskeskkonnad. Seejärel on põhjendatud IDE-de valikut nii serverirakenduse kui ka mobiilirakenduse loomiseks.

5.3.2.1 Java integreeritud arenduskeskkonnad

Kolm kõige suuremat ja populaarsemat IDE-t, mida serveripoolse Java arendamiseks kasutatakse, on IntelliJ IDEA, Eclipse ja NetBeans. Need pole ainsad valikud, kuid selle bakalaureusetöö raames rohkem Java IDE-sid ei käsitleta.

- InrelliJ IDEA – Sellel on kaks versiooni, nimelt tasuta avatud lähtekoodiga kogukonna väljaanne ja tasuline Ultimate väljaanne. Kogukonna väljaanne on mõeldud JVM-i ja Androidi arendamiseks. Ultimate on aga mõeldud ettevõtte tasemel tarkvara arendamiseks. IntelliJ kavandas IDEA välja arendaja loomingulist voogu ehk "tsoonis olemist" silmas pidades. Kõik, mida soovitakse kirjutamise ajal teha, on kiirklahviga juurde pääsetav, sealhulgas sümbolimääratluste kuvamine hüpikaknas. IDEA on laiendanud programmeerimise abi Springile, Java EE-le, Grailsile, Play'le, Androidile,

GWT-ile, Vaadinile, Thymeleafile, Androidile, Reactile, AngularJS-ile ja muudele raamistikele. Kõik need pole Java raamistikud. Lisaks Javale saab IDEA aru paljudest teistest keeltest, sealhulgas Groovy-ist, Kotlinist, Scalast, JavaScriptist, TypeScriptist ja SQL-ist [58].

- Eclipse – Vaba ja avatud lähtekoodiga ning kirjutatud enamasti Java keeles, ehkki selle pistikprogrammide arhitektuur võimaldab Eclipse'i laiendada ka teistes keeltes. Eclipse sai alguse 2001. aastal IBM-i projektina, mille eesmärk oli asendada Smalltalkil põhinev IBM Visual Age IDE-de perekond teisaldatava Java-põhise IDE-ga. Java teisaldatav loomus aitab Eclipse'il olla platvormide ülene: Eclipse töötab Linuxis, Mac OS X-is, Solarises ja Windowsis. Eclipse võlgneb oma jõudluse, või selle puudumise, JVM-ile. Eclipse'il on aeglase töötamise maine. Isegi tänapäeval võib see aeglane tunda, eriti siis, kui IDE värskendab ennast taustal paljude installitud pistikprogrammidega. Pistikprogrammi ökosüsteem on aga üks Eclipse'i tugevustest, samuti on see aeg-ajalt pettumuse allikaks. Eclipse'i pistikprogrammide turg sisaldab praegu üle 1600 lahenduse ja kogukonna loodud pistikprogrammid võivad reklaamitud viisil kas töötada või mitte. Sellegipoolest sisaldavad Eclipse'i pistikprogrammid tuge enam kui sajale programmeerimiskeelele ja peaaegu kahesajale rakenduse arendusraamistikule. Samuti toetatakse enamikke Java-servereid [58].

- NetBeans – See on avatud lähtekoodiga IDE, mis võimaldab Java arendajatel moodulikomplektide abil erinevaid rakendusi luua. NetBeans on saadaval mitmesuguste operatsioonisüsteemide jaoks nagu Windows, Linux, macOS, Solaris. NetBeansi abil on kohandatud tarkvararakenduste loomine väga lihtne, kuna see tõstab Java koodi esile nii süntaktiliselt kui ka semantiliselt. Samuti on palju tööriistu, mis aitavad vigadeta koodi kirjutada. Kuigi NetBeans on peamiselt Java IDE, on sellel laiendid töötamiseks teistes programmeerimiskeeltes, nagu näiteks C, C++, PHP, HTML5, JavaScript jne [59].

5.3.2.2 React Native integreeritud arenduskeskkonnad

Järgnevalt on välja toodud neli populaarseimat IDE-t React Native rakenduse arenduseks, millega lõputöö autor on ise suuremal või vähemal määral kokku puutunud ning mida erinevates arendajatele suunatud internetifoorumites kõige rohkem mainitakse. Need pole

ainsad võimalused React Native arendustööks, kuid selle bakalaureusetöö raames rohkem React Native IDE-sid ei käsitleta.

- Atom – Tasuta ja avatud lähtekoodiga teksti- ja lähtekoodiredaktor macOS-i, Linuxi ja Microsoft Windowsi jaoks. Atom toetab pistikprogramme, mis on kirjutatud Node.js-is ja sellele on sisse ehitatud Giti versioonihaldus. Selle on välja töötanud GitHub. Atom on kaasaegne ja vastutulelik 21. sajandi jaoks loodud konfigureeritav tekstiredaktor. Arendajad kasutavad Atomi laialdaselt kõigi suuremate tehnoloogiate jaoks. Sellel on suur ja aktiivne kogukond, mistõttu on kõige jaoks alati saadaval lisa pistikprogramme [60].
- Visual Studio Code – Microsofti välja töötatud lähtekoodiredaktor Windowsi, Linuxi ja OS X jaoks. See on tasuta ja avatud lähtekoodiga ning sisaldab silumise ja süntaksi esiletõstmise funktsioone, intelligentset koodi lõpetamist, sisse ehitatud Giti versioonihaldust, koodilõike ja koodi redigeerimise funktsioone. Visual Studio Code on üsnagi täielik valmislahendus, kuid vajaduse korral on selle funktsionaalsuse laiendamiseks alati pistikprogramme saadaval.
- Sublime Text – Kõrgetasemeline tekstiredaktor koodi, märgistuse ja proosa jaoks, nagu seda väidab Sublime'i koduleht [61]. See pakub funktsionaalsuse laiendamiseks palju kogukonna arendatud pistikprogramme. Sublime Text on olnud läbi aegade arendajate lemmik redaktoriks [62]. See sisaldab laias valikus funktsioone, nagu süntaksi esiletõstmine, automaatne taane, failitüübi tuvastamine, külgriba, makrod, pistikprogrammid ja paketid, mis muudavad koodibaasiga töötamise lihtsaks.
- WebStorm – WebStorm on ehitatud avatud lähtekoodiga IntelliJ platvormi peale, mida JetBrains on arendanud ja täiustanud juba üle 15. aasta. See pakub tihedat integreerimist VSC-iga, kohaliku ajaloo funktsiooni, sellel on elav pistikprogrammide ökosüsteem, see on täielikult konfigureeritav ja pakub palju muud. WebStorm pakub Reacti ja JSX-i kõrgelt arenenud tuge ning ka programmeerimistuge React Native'i rakenduste ehitamiseks. Lisaks IDE-s sisalduvate mitmesuguste sisseehitatud ülevaatus-tööriistade eelistele on arendajatel võimalus kasutada ka selliseid tööriistu nagu JSLint, JSHint, ESLint ja JSCS [62].

5.3.2.3 IDE-de valik

Kuna lõputöö kirjutamise ajal on autoril ülikooli kaudu juurdepääs kõikidele JetBrainsi tööriistade versioonidele, nii ettevõtte tasemel IntelliJ IDEA väljaandele Ultimate kui ka WebStormile, siis on valitud serveripoolse rakenduse ehitamiseks IntelliJ IDEA ja kliendipoolse rakenduse ehitamiseks WebStormi arenduskeskkond.

IntelliJ Ultimate pakub kõige ulatuslikemaid tööriistu ja jõudluse taset võrreldes kõigi kolme eelmainitud IDE-ga. IntelliJ IDEA toetab prominentseid Java versioone, rakendusservereid, kõige populaarsemaid veebiraamistikke, nagu näiteks Spring MVC ja andmebaasi tööriistu. Lisaks on autoril ülikoolist tulenevalt kõnealuse tarkvaraga kogemust ja tunneb end kõige mugavamalt just selle tarkvaraga.

Sama käib WebStorm arenduskeskkonna kohta; see on eelmainitutest kõige intelligentsem tarkvara, mida saab React Native *frontend* arendusel kasutada.

6 Infosüsteemi arendus

Infosüsteemi arendus on jaotatud kahte suuremasse peatükki: serveripoolse rakenduse arendus ja kliendipoolse rakenduse arendus.

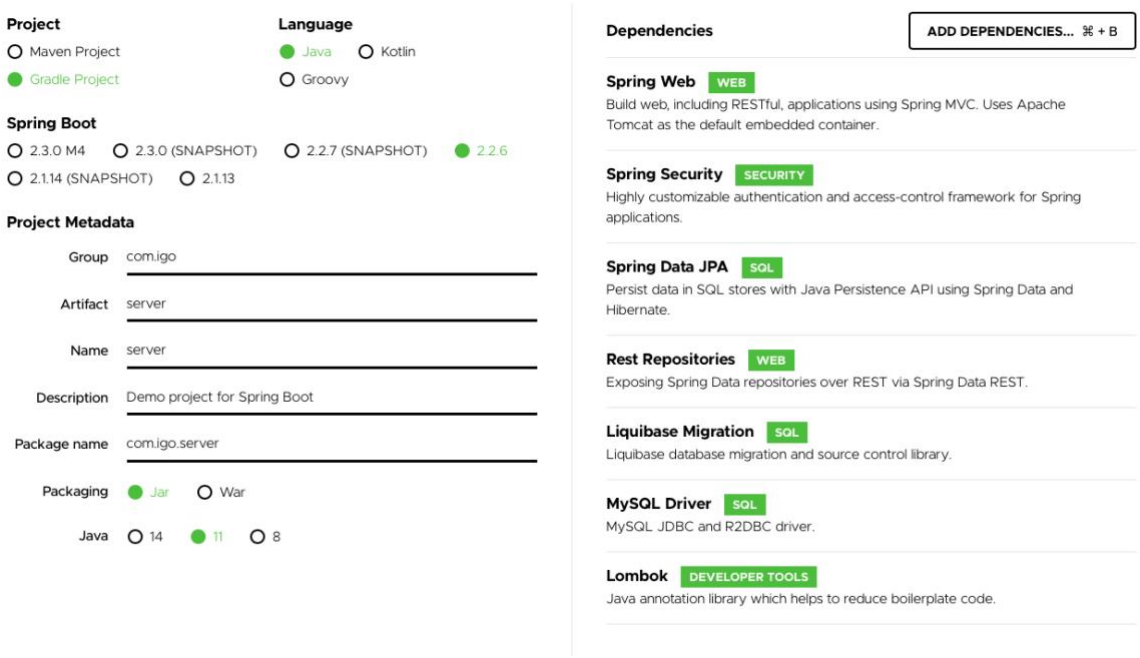
6.1 *Backend* arendus

Serveripoolse rakenduse ülesandeks on HTTP päringutele vastamine, andmete töötlemine ja äriloogika eest vastutamine. Järgnevates peatükkides on lähemalt kirjeldatud kasutatud tehnoloogiaid ja lahenduse erinevaid aspekte.

6.1.1 Spring Boot

Infosüsteemi *backend* on ehitatud Spring Boot raamistiku peale. See vähendab palju arendusaega ja suurendab tootlikkust. Selle abil saab vältida *boilerplate* koodi, annotatsioonide ja XML konfiguratsiooni kirjutamist. Kõiki Springi rakendusi on soovitatav alustada Spring Initializr abiga. Initializr pakub kiiret viisi kõigi rakenduse jaoks vajalike sõltuvuste leidmiseks ja teeb arendaja eest ära palju seadistusi. Spring Initializr on kokkuvõttes veebirakendus, mis genereerib Spring Boot projekti ülesehituse. See ei genereeri ühtegi rakenduse sisest äriloogilist koodi, kuid annab ette põhi projekti struktuuri ja kas Maveni või Gradle'i spetsifikatsiooni rakenduse ehitamiseks [63].

Joonisel 5 on välja toodud serveripoolse rakenduse ehitamiseks ja haldamiseks valitud tööriistad, teegid ja raamistikud.

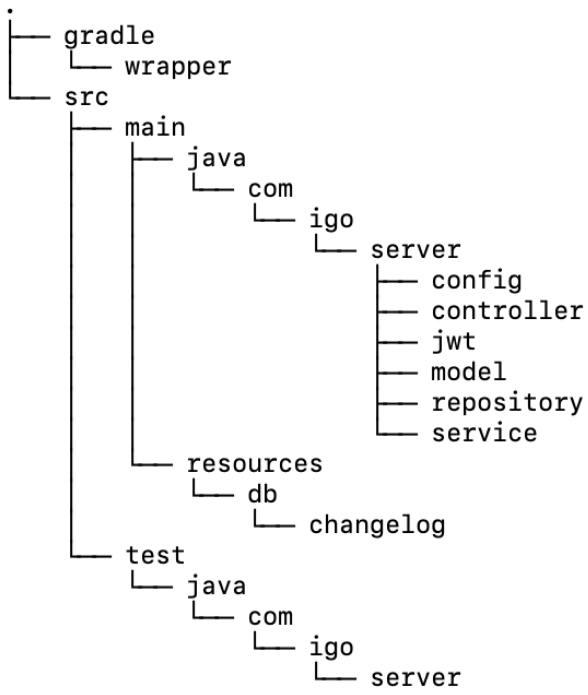


Joonis 5. Spring Initializr kuvatõmmis näitamaks valitud tööriistu.

6.1.2 Mitmetasandiline arhitektuur

See muster on enamiku Java EE rakenduste *de facto* standard ja seetõttu tunneb seda enamus arhitekte, disainereid ja arendajaid [64]. Mitmetasandilise arhitektuuri kohaselt peaks rakendus olema ehitatud kihiti. See sobib ettevõtte tasemel kliendi-serveri rakenduste toetamiseks, pakkudes lahendusi eskaleerimise, turvalisuse, tõrketaluvuse, koodi korduvkasutamise ja hooldamise jaoks [65]. See aitab arendajatel luua paindlikke ja korduvkasutatavaid rakendusi. Mitmetasandilise arhitektuuri peamine kujunduspõhimõte on huvide eraldatus. Põhimõtteliselt on tegemist kohandatava struktuuriga ehk moodulsüsteemiga. Moodulsüsteemi eeliseks on väiksemate komponentide kergem hooldamine. Programmi saab jagada funktsionaalsete aspektide alusel, kus igal kihil on arhitektuurimustris rakenduses konkreetne roll ja vastutus. Kuna kihid on iseseisvad, siis pole neil teistest kihtidest sõltuvusi või sõltuvused on minimeeritud, näiteks *Dependency Injection* abiga. Seega on selle lähenemisviisi korral erinevates kihtides või komponentides muudatuste tegemine ilma kogu süsteemi mõjutamata lihtsustatud. Mitmetasandilise arhitektuuri tõttu saab ka testimise ja silumise jooksul aega säästa.

Bakalaureusetöö käigus ehitatud serveripoolne rakendus järgib eelpool kirjeldatud mitmetasandilist arhitektuuri eelnevalt välja toodud põhjuste tõttu. Joonisel 6 on näha serveripoolse rakenduse failide puud, demonstreerimaks, kuidas on rakendus erinevatesse kaustadesse jaotatud vastavalt funktsionaalsusele.



Joonis 6. Serveripoolse rakenduse failide puu.

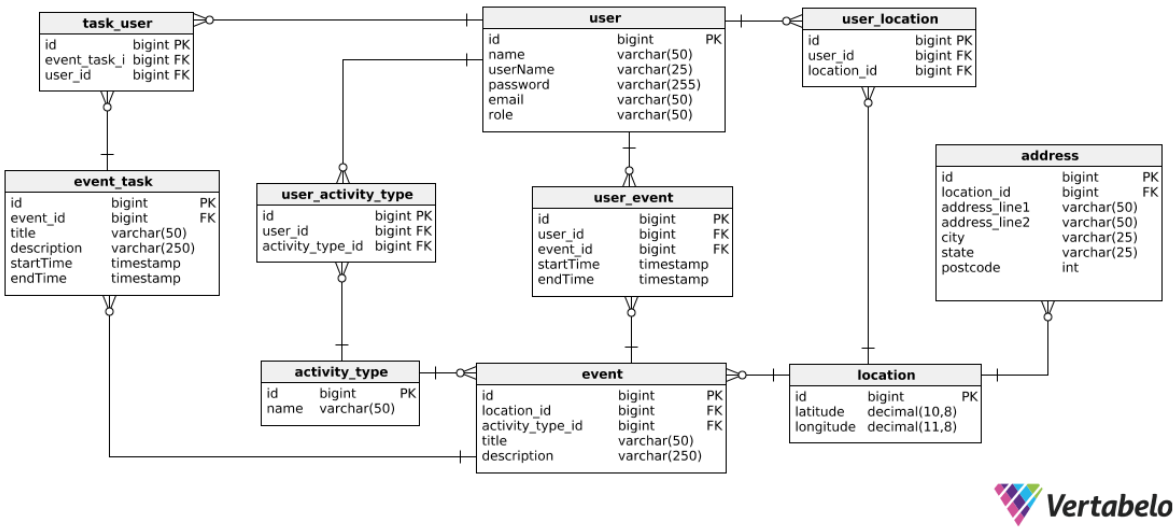
6.1.3 Hoidla muster

Mitmetasandilise arhitektuuri puhul on äriloogika ja andmete kihi vahel tihti ka andmetele juurdepääsu kiht DAL (*Data Access Layer*). Selle asemel, et teha päring andmete kättesaamiseks otse andmebaasi mitmest tabelist korraga, võib näiteks kutsuda rakenduse DAL-ist ühte meetodit, mis neid andmebaasi päringuid kokku võtab. Andmepöörduskihi omamine tagab tsentraliseeritud asukoha kõigi andmebaasi päringute jaoks ja hõlbustab rakenduse teisaldamist teistesse andmebaasisüsteemidesse, kui selleks peaks vajadus tekkima [66]. Seda kihti lahendatakse erinevate arhitektuurimustrite kaudu. Lõputöö autoril on kogemust kahe DAL mustriga – DAO ja hoidla mustriga. Kõnealuse projekti käigus ehitatud infosüsteemi serveripoolse rakenduse puhul on lähtutud hoidla mustrist.

Martin Fowler kirjeldab raamatus „Patterns of Enterprise Application Architecture“ järgmiselt: „Hoidla täidab vahendaja ülesandeid domeenimudelite kihtide ja andmete kaardistamise vahel, toimides sarnaselt mälus asuvate domeeniobjektide komplektiga. Kliendiobjektid loovad päringud deklaratiivselt ja saadavad need vastuste saamiseks hoidlatesse. Kontseptuaalselt kapseldab hoidla andmebaasi salvestatud objektide komplekti ja nendega tehtavaid toiminguid. Hoidlad toetavad ka eesmärki eraldada domeenimudelite ja andmete jaotamise või kaardistamise vahelist sõltuvust selgelt ja ühes suunas.“ [67]

6.1.4 Andmebaas ja selle teostus

ERD ehk *Entity Relationship Diagram* on kõige laiemalt levinud metoodika andmemudelite koostamiseks ja kirjelduse esitamiseks [68]. Järgnevalt on joonisel 7 välja toodud andmebaasi olemi-suhte diagramm. Loodud olemi-suhte diagrammi alusel genereeriti ehitatava andmebaasi loomise käsud.

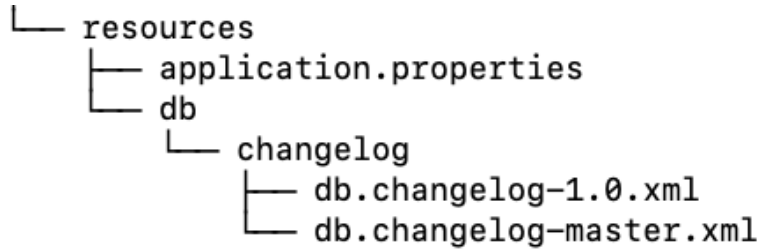


Joonis 7. Loodava andmebaasi olemi-suhte diagramm.

Serveripoolse rakenduse puhul on kasutatud objekt-relatsioonilist kaardistamist, sidumaks domeeni mudelid relatsioonilise andmebaasiga. Selleks on rakenduses kasutatud Hibernate’i, mis on objekt-relatsioonilise kaardistamise tööriist. Hibernate on üks küpsemaid JPA rakendustest, mille taga seisab ja mida toetab tohutu kogukond. JPA kirjeldab, kuidas peaks haldama rakenduses erinevate andmebaasi tabelite vahelisi suhteid. JPA tegeleb annotatsioonidega. Kui Java objekt määratletakse *@Entity*

annotatsiooni kaudu olemiks, siis ühendatakse või lisatakse kõik selle väljad olemi tabeli eri veergudena automaatselt.

Andmebaasi konfiguratsioon, nagu näiteks andmebaasi asukoht, sisse logimise andmed juurdepääsuks, andmebaasi skeemi loomise faili asukoht jms asub *application.properties* failis *resources* kataloogi all. Andmebaasi muudatuste jälgimiseks, haldamiseks ja muudatuste tegemiseks on kasutatud Liquibase'i teeki. Liquibase kasutab muudatuste logi, et loetleda andmebaasi muudatusi selgesõnaliselt järjekorras. Muutuste logi ehk *changelog* toimib muudatuste raamatupidamis žurnaalina ja sisaldab *changeSet*-e ehk muudatuste üksusi, mida Liquibase saab andmebaasi peal täita [69]. Nagu jooniselt 8 näha võib, on andmebaasi skeemi SQL käsud *changelog* kausta all XML failidena.



Joonis 8. Andmebaasi failide puu.

6.1.5 REST API

Serveripoolel on järgitud API ehitamise häid tavasid. URI peaks sisaldama ainult ressursse (nimisõnu), mitte toiminguid ega tegusõnu. /companies on hea URI näide, mis ei sisalda ühtegi toimingut. Ressurss peaks API puhul alati mitmuses olema. Kui tahetakse ressurssi lisada, muuta või kustutada, siis saadetakse see päringuga objektina kaasa. Kui tahetakse juurdepääsu ressursi ühele eksemplarile, siis lisatakse URI-le ressursi ID. Server teab mida kõnealuse ressursiga peale hakata tänu HTTP-meetoditele GET, POST, DELETE, PUT.

Kuna mobiilirakenduse kasutajad värskendavad oma rakendusi erinevatel sagedustel, siis on API versiooni määramine olulisem siin, kui muudes, paremini kontrollitavates keskkondades. Kuna rakenduse mitu erinevat versiooni võib töötada korraga, peab server konsolideerima ja töötlema erinevaid taotlusi, mis tulevad nii uutelt kui ka vanadelt kasutajatelt. API peab hakkama saama nii vanade kui ka uute taotlustega. Seda on tehtud

suunates päringuid versiooninumbriga. Mõned väidavad, et see versiooninumber peaks minema URI peale ja mõned väidavad, et see tuleks paigutada päringu päisesse [70]. Käesolevas töös on eelistatud see panna URI peale, et see oleks hõlpsamalt avastatav. Seega tuleb iga päringu ette lisada /api/v1 osutamaks API konkreetsele versioonile.

Järgnevalt on joonisel 9 näitena välja toodud mõned serverirakenduses täide viidud API *endpoint*-id. *Endpoint* kujutab endast mingit HTTP kaudu ligipääsetavat URL-i, mis vastab ressursiga. Välja pole toodud *endpoint*-ide vastuse koode, kuid kõige sagedamini on kasutusel 200 *OK*, 201 *Created*, 204 *No Content*, 304 *Not Modified*, 400 *Bad Request*, 401 *Unauthorized*, 403 *Forbidden*, 404 *Not Found*, 500 *Internal Server Error*, 503 *Service Unavailable*.

```
POST    /user/registration
POST    /user/login
POST    /user/logout
GET     /users
GET     /users/{userId}
GET     /users/{userId}/events
GET     /events
POST    /events
PATCH   /events
DELETE  /events
GET     /events/{eventId}
POST    /events/{eventId}/tasks
GET     /events/{eventId}/tasks
DELETE  /events/{eventId}/tasks
```

Joonis 9. Serverirakenduses loodud API päringud.

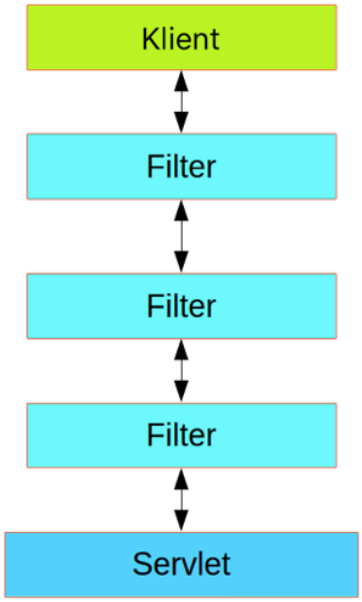
6.1.6 Veebirakenduse turvalisus

Turvepiirangute kehtestamiseks on kasutatud Spring Security raamistikku. Spring Security on raamistik, mis pakub autentimist, autoriseerimist ja kaitset levinud rünnakute eest. Nii on ka lõputöös kasutatud Spring Security'it turvaliseks sisse logimiseks ja autentimise REST API teede jaoks.

Kliendi poolelt saadetakse serveripoole kasutajanimi ja parool. Seda turvalisel meetodil – autentimise krüptitud päisega. Kasutajanime ja parooli saaks saata ka URI kaudu muutujatena, kuid see pole kohe kindlasti turvaline viis, sest nii on kõigil

kõrvalistel osapooltel võimalik neid andmeid näha. Kui serveri pool kliendilt sisse logimise andmed krüptitud päise kaudu kätte saab, siis pannaksegi Spring Security'i ja selle filtrid kasutusse. Spring Security saab sisse logimise taotluse ja dekrüptib selle autentimise päise automaatselt. Spring Security edastab need detailid *userDetails* teenuse abil. Selles klassis kontrollitakse kasutajat vastavalt kasutajanimele. Põhiline turvalisus on sellega tagatud.

JSON Web Token on standard, mida kasutatakse rakenduse pääsulubade ehk märgendite loomiseks. Neid märgendeid jagatakse HTTP-päringutega kliendi ja serveri vahel. Süsteemi sisse logimisel, saadetakse sisse logimise taotlus kliendi poolelt serveripoole. Pärast serveripoole jõudmist laseb Spring Security selle oma turvafiltritest läbi. Filter kontrollib HTTP päiseid ja sisu ning autoriseerib selle või suunab ebaõnnestunud URI peale. Spring Security põhineb Servlet-filtritel. All olev joonis 10 näitab töötlejate tüüpilist kihilisust ühe HTTP-päringu korral. Filtrid moodustavad ahela, nii et need on järjestatud, ja üks filter saab ülejäänud ketile n.ö. veto peale panna, kui ta soovib ise taotlust käsitleda. Filter võib ka muuta päringut ja/või vastust, mida kasutatakse allavoolu filtrites ja servletis [71].



Joonis 10. Spring Security Servlet-filtrid.

Pärast autentimist luuakse märgend kooskõlas JSON veebi märgendiga. Selle loomiseks on kasutatud salajast võtit ja kasutajanime. Siis saadetakse see HTTP-vastusena tagasi kliendile. Pärast neid toiminguid on kliendi poolel turvaline märgend olemas, seega saab

klient selle märgendiga päringuid saata. Näiteks saadab klient päringuna API URI ja märgendi väärtuse koos autoriseerimise HTTP päisega nagu võib jooniselt 11 näha. Siis kontrollib serveripoolne JWT filter seda autentimispäise väärtust ja kui see on kehtiv, tagastab filter eduka vastuse. JWT peaks kontrollima, kas märgend on kehtiv või mitte. Lõputöös on JWT konfiguratsioon, nagu JWT aegumise aeg ja võti, millega märgendit krüptida, koos autentimise skeemi jms leitav *application.properties* failis *resources* kataloogi alt.

```
async setHeaders() {
    const user = await UserService.currentUser;
    this.headers = {
      'Content-Type':'application/json',
      'authorization':'Bearer ' + (user ? user.token: '')
    };
}

updateUser(user) {
    return axios.put(API_URL + "user-update",
                     JSON.stringify(user), {headers:
this.headers});
}
```

Joonis 11. Päringu näide kliendi poole pealt.

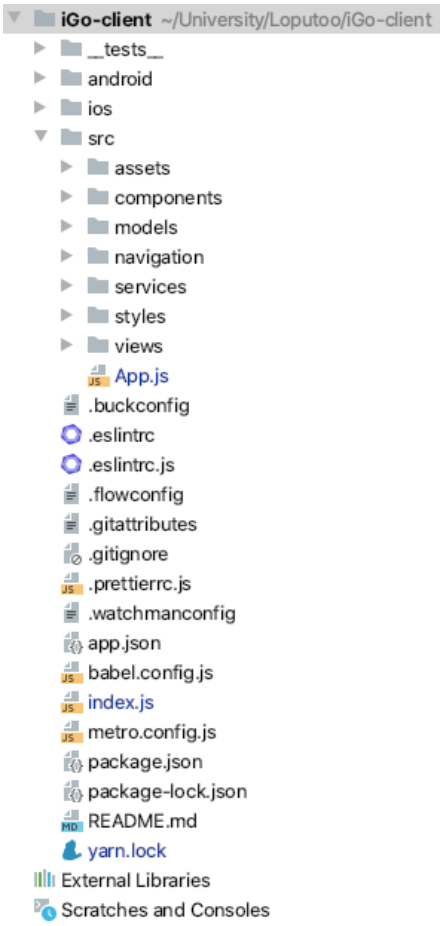
6.2 Mobiilirakenduse arendus

Mobiilirakendus on loodud kasutades Facebooki React Native'i dokumentatsiooni. Kuigi arenduskeskkonnaks on valitud WebStorm, on dokumentatsiooni järgides alla tõmmatud ka vastavate platvormide *native* arenduskeskkonnad – Android Studio ja Xcode. Need arenduskeskkonnad aitavad seadistada vajalikud tööriistad, et luua React Native rakendusest Android- või iOS-rakendus. Samuti pakuvad mõlemad keskkonnad vastava platvormi erinevate mobiiltelefonide simulaatoreid.

Uue React Native projekti genereerimiseks kasutati React Native'i sisseehitatud käsurealiidest.

6.2.1 Rakenduse struktuur

Projekti struktuur on üles ehitatud pidades silmas aatomidisaini põhimõtteid ja tuleviku laiendamise võimalusi. Projekti struktuur on määratletud ka selleks, et hõlbustada selle hooldamist. Jooniselt 12 on näha failide jaotust kaustadesse. Esiteks on loodud algkaust *src*, mis sisaldab kõiki peamisi projektifaile.



Joonis 12. Kliendirakenduse projekti struktuur.

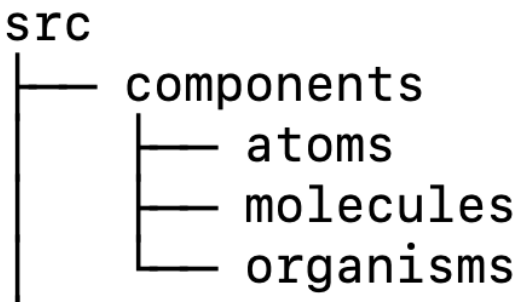
6.2.1.1 Aatomidisain

Kasutajaliidesed koosnevad väiksematest komponentidest. See tähendab, et kogu liidese võib jagada põhilisteks ehitusplokkideks ja sealt edasi töötada. See on aatomidisaini peamine idee. Aatomidisain on disainisüsteemide loomise metoodika. Aatomidisainis on viis erinevat taset [72]:

1. Aatomid
2. Molekulid

3. Organismid
4. Mallid
5. Leheküljed

Käesolevas lõputöös on leheküljed ja suuremad komponendid üles ehitatud kasutades väiksemaid komponente. Sellest lähtuvalt on komponentide kausta all vastavad kaustad aatomitele, molekulidele ja organismidele nagu võib failide puust joonisel 13 näha. Igas komponentide kataloogis on fail index.js, mis ekspordib kõnealuse kategooria.



Joonis 13. Komponentide kausta struktuur.

Aatomid on väikseimad komponendid, nagu nupud, pealkirjad, sisendid või sündmuse värvipaletid, animatsioonid ja fondid. Nagu looduses olevad aatomid, on nad üsna abstraktsed ja pole üksi sageli kasulikud [72].

Molekulid on aatomite rühmad, mis on omavahel seotud ja on ühendi väikseimad põhiüksused. Need molekulid on disainisüsteemide alustalaks. Näiteks vormi silt, sisend või nupp pole iseenesest väga kasulik, aga ühendades nad kokku vormiks, saavad nad koos juba midagi ära teha [72].

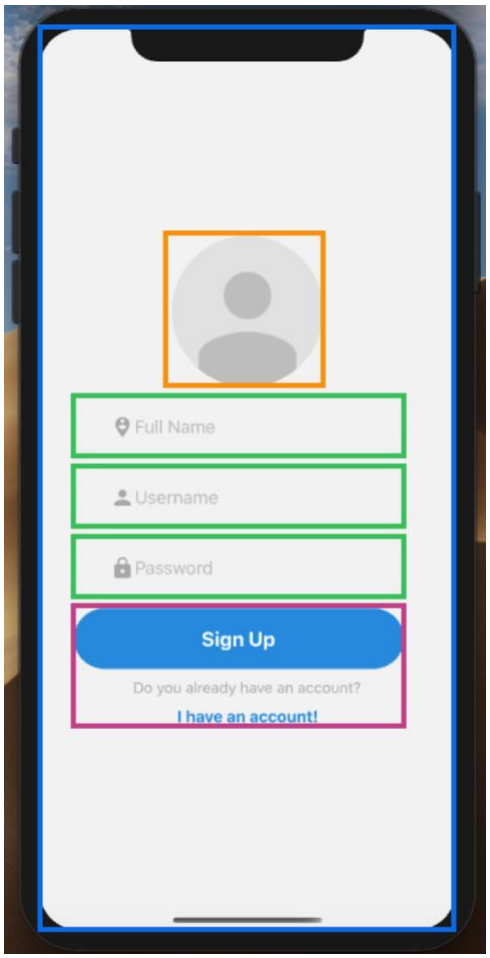
Organismid on molekulide rühmad, mis on omavahel ühendatud, et moodustada liidese suhteliselt keeruline ja selge osa. Molekulidelt organismidele ülesehitamine soodustab eraldiseisvate, ülekantavate ja korduvkasutatavate komponentide loomist [72].

6.2.2 Komponendipõhine arhitektuur

Komponendid on mis tahes Reacti rakenduse ehitusplokid ja tüüpilisel Reacti rakendusel on neid palju. Lihtsustatult öeldes on komponent JavaScripti klass või funktsioon, mis valikuliselt aktsepteerib sisendeid, see tähendab atribuute (*props*) ja tagastab React elemendi, mis kirjeldab, kuidas kasutajaliidese osa peaks välja nägema [73].

Komponendid jagunevad funktsioonide ja klasside komponentideks. Funktsiooni komponent on olekuta ja klassi komponent on olekuga. Oleku omamise all mõeldakse seda, et komponendis on muutujaid, mille seisu kõnealune komponent jälgib.

Ehitatud kliendirakenduse juures on kasutatud sedasama komponendipõhist arhitektuuri ja iga lehekülg on üles ehitatud järgides seda lähenemist. Joonisel 14 võib näitena näha rakenduse kasutaja registreerumise lehte ja kuidas see on jaotatud neljaks komponendiks.



Joonis 14. Kasutajaliides komponentideks jaotatult.

Kuna klassi komponentide liigne ja asjatu kasutamine võib negatiivselt mõjutada jõudlust, samuti koodi loetavust, hooldatavust ja kontrollitavust, siis on käesolevas lõputöös valminud mobiilirakenduse ehitamisel eelistatud võimaluse korral kasutada just funktsionaalseid komponente nende ennustatava käitumise ja lakoonilisuse tõttu. Funktsiooni komponendid on puhtalt esituslikud ja neid tähistab lihtsalt funktsioon; see tähendab, et nende väljund on alati sama, kui sisendiks on antud samad atribuudid.

7 Tulemuste analüüs

Eesmärgiks oli luua töötav lahendus sissejuhatuses kirjeldatud probleemile. Arvestades ajalist piirangut oli projekti skoop liiga suur, saavutamaks oodatud tulemust. Mitte kõik funktsionaalsed ja mittefunktsionaalsed nõuded said projekti käigus täidetud. Lõputöö autoril puudus enne kõnealust projekti mobiiliarenduse kogemus täielikult, seega tuli kliendirakenduse ehitamisel ilmsiks probleeme, millega ei oldud projekti alguses arvestatud. Lõputöö käigus tehti aga ära ulatuslik analüüs ja tulemusena ehitati tugev põhi mobiilirakenduse edasi arendamiseks. Serveripoolne API valmis täielikult ja mobiilirakendus on käitatav nii Androidi kui ka iOS-i platvormi peal.

Tuleviku edasi arendustele mõeldes võiks lisada järgmised funktsionaalsed nõuded:

- Kasutaja saab näha asukohti teatud spordialaga tegelemiseks kaardi pealt.
- Süsteem peaks võimaldama kasutajal digitaalseid pilte üles laadida teatud asukoha kohta kaardil.
- Kasutaja saaks lugeda teiste liikmete arvamusavaldusi spordialaga tegelemise kohta teatud asukohtadel kaardil.
- Kasutaja saab näha teiste liikmete pilte spordialaga tegelemise kohta teatud asukohtadel kaardil.
- Süsteem peaks võimaldama kaarti suurendada ja vähendada.

Lisaks eelmainitule peaks arvestama kasutajate arvu tõusuga tuleviku edasi arenduste juures ja sellega seotud turvaprobleemidele, nagu näiteks halbade kavatsustega kasutajad. Sellele mõeldes peaks rakendusel olema „Anna kasutajast teada“ funktsioon, kus kasutajad saaksid rakenduse kaudu administraatoritele teada anda kahtlastest ja illegaalsetest tegevustest või mitte õigesti käituvatest kasutajatest. Samas võiks ka rakendada soovituste põhist kasutajate registreerumist, et platvormile saaksid juurdepääsu ainult n.ö. soovitatud inimesed.

8 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua spordisõprade kokku toomiseks mobiilirakendus, mis lahendaks kaaslase leidmise probleemi vaba aja tegevuste harrastamiseks ja täidaks eksisteerivate lahenduste puudujääke, nagu näiteks ilmastikutingimuste detailne ülevaade ja töö organiseerimise võimalus sündmuse erinevate liikmete vahel.

Lõputöö käigus valmis ulatuslik analüüs ja uus mobiilirakendus. Mobiilirakenduse arendamisel hoiti API- ja kliendirakenduse arendus eraldi, mõeldes tuleviku laiendamise võimaluste peale. Mõlema rakenduse puhul peeti silmas moodulsüsteemi põhimõtteid ja mitmetasandilise arhitektuuri tõekspidamisi. Eelmainitud on abiks koodi loetavuse, testimise ja hooldamise puhul. Lisaks saab eraldi API abil rakenduse loogikat tulevikus kasutada suvaline arv kasutajaliidese rakendusi. Kliendi ja serveripoolt eraldi hoides tõusis ka mobiilirakenduse reageerimisvõime, mis parandab kasutajakogemust.

Projekt oli suuremahuline ja hõlmas kõiki tarkavaraarenduse elutsükli osasid. Tänu tehtud tööle on mobiilirakendus edasi arenduseks valmis ja teatav funktsionaalsus on saavutatud nii iOS-i kui ka Android platvormi jaoks.

Kasutatud kirjandus

[1] A. Lynch, „Edrawsoft,“ 14 aprill 2020. [Võrgumaterjal]. Available: https://www.edrawsoft.com/what-is-bpmn.html.

[2] „Eventbrite,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Eventbrite.

[3] B. Vigliarolo, „Techrepublic,“ 31 märts 2017. [Võrgumaterjal]. Available: https://www.techrepublic.com/pictures/gallery-10-apps-for-making-the-most-of-your-free-time/2/.

[4] „Fitness Buddy App 5F - Find Fit Friends,“ [Võrgumaterjal]. Available: http://app5f.com.

[5] „statcounter Global Stats,“ [Võrgumaterjal]. Available: https://gs.statcounter.com/os-market-share/mobile/estonia.

[6] J. B. Cunha, „freecodecamp,“ 5 Juuni 2018. [Võrgumaterjal]. Available: https://www.freecodecamp.org/news/a-deeply-detailed-but-never-definitive-guide-to-mobile-development-architecture-6b01ce3b1528/.

[7] A. Monus, „Raygun,“ 19 märts 2019. [Võrgumaterjal]. Available: https://raygun.com/blog/native-app-development/.

[8] P. L. Sam Richard, „web.dev,“ 24 veebruar 2020. [Võrgumaterjal]. Available: https://web.dev/what-are-pwas/.

[9] A. G. Margaret Rouse, „Techtarget,“ märts 2018. [Võrgumaterjal]. Available: https://searchsoftwarequality.techtarget.com/definition/native-application-native-app.

[10] M. Rouse, „Techtarget,“ [Võrgumaterjal]. Available: https://searchsoftwarequality.techtarget.com/definition/native-application-native-app.

[11] M. Firtman, „Techbeacon,“ [Võrgumaterjal]. Available: https://techbeacon.com/app-dev-testing/web-native-mobile-app-frameworks-how-sort-through-choices.

[12] P. Saccomani, „MobiLoud,“ [Võrgumaterjal]. Available: https://www.mobiloud.com/blog/native-web-or-hybrid-apps/.

[13] „Codecademy,“ [Võrgumaterjal]. Available: https://www.codecademy.com/articles/back-end-architecture.

[14] M. Soucoup, Microsoft, 7 Mai 2019. [Võrgumaterjal]. Available: https://mybuild.techcommunity.microsoft.com/sessions/77149.

[15] „DA14,“ 07 veebruar 2018. [Võrgumaterjal]. Available: https://da-14.com/blog/ultimate-guide-api-architecture-rest-soap-or-graphql.

[16] „Clockwise,“ 20 august 2019. [Võrgumaterjal]. Available: https://clockwise.software/blog/choice-between-mobile-backend-as-a-service-and-custom-backend/.

[17] T. Valeva, „Thetool.io,“ 12 juuni 2017. [Võrgumaterjal]. Available: https://thetool.io/2017/mbaas-for-app-developers.

[18] C. Fanchi, „Backendless,“ 27 november 2019. [Võrgumaterjal]. Available: https://backendless.com/what-is-mobile-backend-as-a-service-mbaas/.

[19] D. Prothero, „Twilio Docs,“ [Võrgumaterjal]. Available: https://www.twilio.com/docs/glossary/what-is-serverless-architecture.

[20] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Serverless_computing.

[21] F. Bashir, „Freecodecamp,“ 5 juuli 2019. [Võrgumaterjal]. Available: https://www.freecodecamp.org/news/what-is-serverless-architecture-what-are-its-pros-and-cons/.

[22] [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Ruby_(programming_language).

[23] [Võrgumaterjal]. Available: https://www.tutorialspoint.com/ruby/ruby_overview.htm.

[24] Y. Nader, „Hackr.io,“ 09 aprill 2020. [Võrgumaterjal]. Available: https://hackr.io/blog/python-vs-java.

[25] P. Giampedraglia, „Asapdevelopers,“ 23 november 2019. [Võrgumaterjal]. Available: https://www.asapdevelopers.com/python-backend-language/.

[26] K. Tolani, 11 jaanuar 2019. [Võrgumaterjal]. Available: https://learntocodewith.me/posts/backend-development/#python.

[27] „Yourteaminindia,“ [Võrgumaterjal]. Available: https://www.yourteaminindia.com/blog/java-backend-development/.

[28] „Php.net,“ [Võrgumaterjal]. Available: https://www.php.net/manual/en/intro-whatis.php.

[29] „Guru99,“ [Võrgumaterjal]. Available: https://www.guru99.com/what-is-php-first-php-program.html.

[30] E. Tsygankov, „Toptal,“ [Võrgumaterjal]. Available: https://www.toptal.com/microsoft/eight-reasons-why-microsoft-stack-is-still-a-viable-choice.

[31] M. Marinina, „Codeburst.io,“ 10 oktoober 2017. [Võrgumaterjal]. Available: https://codeburst.io/java-vs-net-factors-to-consider-cde1d22b06a7.

[32] „Docs.microsoft,“ 19 juuni 2018. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server.

[33] J. Rachowicz, „Netguru,“ 23 veebruar 2017. [Võrgumaterjal]. Available: https://www.netguru.com/blog/use-node-js-backend.

[34] „DA-14,“ 30 jaanuar 2019. [Võrgumaterjal]. Available: https://da-14.com/blog/nodejs-vs-java-backend-language-large-software-development.

[35] N. Chrzanowska, „Netguru,“ 23 märts 2017. [Võrgumaterjal]. Available: https://www.netguru.com/blog/pros-cons-use-node.js-backend.

[36] E. Waszkowski, „Hackernoon,“ 17 jaanuar 2020. [Võrgumaterjal]. Available: https://hackernoon.com/why-use-nodejs-as-your-backend-4gpa31qk.

[37] Z. W. Wu, „Medium,“ 3 oktoober 21018. [Võrgumaterjal]. Available: https://medium.com/@zhenwu93/relational-vs-non-relational-databases-8336870da8bc.

[38] „Bitdegree,“ 23 jaanuar 2020. [Võrgumaterjal]. Available: https://www.bitdegree.org/tutorials/types-of-databases/.

[39] I. C. Education, „IBM,“ 06 august 2019. [Võrgumaterjal]. Available: https://www.ibm.com/cloud/learn/relational-databases.

[40] Ian, „Database guide,“ 23 juuni 2016. [Võrgumaterjal]. Available: https://database.guide/what-is-a-column-store-database/.

[41] M. Techlabs, „Medium,“ 26 juuli 2016. [Võrgumaterjal]. Available: https://medium.com/@MarutiTech/how-to-choose-the-perfect-back-end-technology-5674db9c0192.

[42] C. Wodehouse, „Upwork,“ 19 aprill 2017. [Võrgumaterjal]. Available: https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/.

[43] o. a. M. Drake, „Digitalocean,“ 19 märts 2019. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[44] M. Rouse, „Techtarget,“ [Võrgumaterjal]. Available: https://searchdatamanagement.techtarget.com/definition/MariaDB.

[45] J. Stein, „Toptal,“ [Võrgumaterjal]. Available: https://www.toptal.com/mobile/comparing-react-native-to-cordova.

[46] R. Pot, „Medium,“ 6 september 2017. [Võrgumaterjal]. Available: https://medium.com/all-titanium/building-native-apps-with-javascript-titanium-5139c855302e.

[47] „Nmgtechnologies,“ [Võrgumaterjal]. Available: https://nmgtechnologies.com/blog/frameworks-hybrid-app-development.html#react-native.

[48] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/NativeScript.

[49] „Mobileappdaily,“ 23 märts 2020. [Võrgumaterjal]. Available: https://www.mobileappdaily.com/best-hybrid-app-frameworks.

[50] A. meeskond, „Csform,“ 8 november 2018. [Võrgumaterjal]. Available: https://csform.com/best-cross-platform-mobile-frameworks-for-2019/.

[51] „Techtarget,“ [Võrgumaterjal]. Available: https://searchsoftwarequality.techtarget.com/definition/development-environment.

[52] „Wikipedia,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Git.

[53] Momchil, „Hackernoon,“ 21 aprill 2020. [Võrgumaterjal]. Available: https://hackernoon.com/gitlab-vs-github-key-differences-and-which-one-is-better-2020-update-rhaw3y3z.

[54] N. Kadivar, 1 november 2018. [Võrgumaterjal]. Available: https://hackernoon.com/top-10-version-control-systems-4d314cf7adea.

[55] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Mercurial.

[56] I. Brudo, „Codota,“ 09 märts 2020. [Võrgumaterjal]. Available: https://blog.codota.com/svn-vs-git/.

[57] B. Hart, „Perforce,“ 28 oktoober 2019. [Võrgumaterjal]. Available: https://www.perforce.com/blog/vcs/clearcase-vs-git.

[58] M. Heller, „Javaworld,“ 18 detsember 2018. [Võrgumaterjal]. Available: https://www.javaworld.com/article/3114167/choosing-your-java-ide.html.

[59] „Geeksforgeeks,“ [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/what-will-be-the-best-java-ides-in-2020/.

[60] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Atom_(text_editor).

[61] [Võrgumaterjal]. Available: https://www.sublimetext.com.

[62] M. Basrai, „Icicletech,“ 03 detsember 2019. [Võrgumaterjal]. Available: https://www.icicletech.com/blog/top-10-editors-for-react-native.

[63] C. Walls, Spring Boot in Action, Manning Publications, 2015.

[64] M. Richards, Software Architecture Patterns, O'Reilly Media, Inc., 2015.

[65] „Guru99,“ [Võrgumaterjal]. Available: https://www.guru99.com/n-tier-architecture-system-concepts-tips.html.

[66] „Wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Data_access_layer.

[67] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.

[68] P. Raspel, „Itcollege,“ [Võrgumaterjal]. Available: http://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/04/4.htm.

[69] „Liquibase,“ [Võrgumaterjal]. Available: https://www.liquibase.org/get_started/how-lb-works.html.

[70] M. Tea, „Savvyapps,“ 23 juuli 2019. [Võrgumaterjal]. Available: https://savvyapps.com/blog/how-to-build-restful-api-mobile-app.

[71] [Võrgumaterjal]. Available: https://spring.io/guides/topicals/spring-security-architecture.

[72] B. Frost. [Võrgumaterjal]. Available: https://bradfrost.com/blog/post/atomic-web-design/.

[73] J. Kagga, „Medium,“ 14 mai 2018. [Võrgumaterjal]. Available: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb.

[74] „Javatpoint,“ [Võrgumaterjal]. Available: https://www.javatpoint.com/ionic-vs-cordova.

Lisa 1 – Serveripoolse rakenduse ja mobiilirakenduse versioonihaldus

Backend – https://github.com/DigiZiggy/iGo-server

Kliendirakendus – https://github.com/DigiZiggy/iGo-client