TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Jan Joonas Parve 179591IADB

Tagarakendusteenuste analüüs ning Google Firebase rakenduse avatud lähtekoodiga lahendusele migreerimine

Bakalaureusetöö

Juhendaja: Brita Moorus
B.Sc.

Tallinn 2023

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: Jan Joonas Parve

18.03.2023

Annotatsioon

Käesoleva lõputöö raames leiti parim võimalik lahendus rakendusele, mis koges tagarakendusteenuste kasutusega tulenevaid probleeme. Lahenduse hindamiseks migreeriti rakendus alternatiivsele lahendusele, mille eesmärgiks oli hinnata taolise protsessi keerukust ja mahtu. Töö raames analüüsiti ning võrreldi erinevaid tagarakendusteenuseid.

Töö tulemusena tekkis ülevaade, mida arvestada tulevikus tagarakenduse tehnoloogiate valikul ja informatsioon, millele põhineda analoogsetes situatsioonides olevatel projektidel. Tööst saab järeldada, et tagarakendusteenustel on kindel kasutusargument ka minimaalse elujõulise rakenduse faasist kaugemale mõeldes, kui teha arendusprotsessi alguses õigeid otsuseid tehnoloogia valikul.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 28 peatükki, 8 joonist, 5 tabelit.

Abstract

Analysis of Backend-as-a-service Solutions and Migration of a Google Firebase Application to an Open-source Solution

During the research of this thesis, the best possible solution was found for an application that was suffering from problems related to the usage of a backend-as-a-service (BaaS) platform. To analyze the solution, the application was migrated to an alternative solution to rate the complexity and workload of such process. In this thesis different BaaS products were compared and analyzed.

As a result of the study, an overview of what to consider in future during the process of selecting the backend technology of a project and information was gathered on which to base decisions in similar situations in projects. From the thesis, it is possible to conclude that BaaS products have a compelling argument of usage even when thinking further from the application's minimum-viable-product phase if correct decisions are made from the start.

The thesis is in Estonian and contains 26 pages of text, 28 chapters, 8 figures, 5 tables.

Lühendite ja mõistete sõnastik

API	*Application programming interface,* reeglid ja vahendid rakendusprogrammi suhtluseks teise süsteemiga
AWS	*Amazon Web Services,* Amazoni pilveteenuste platvorm
BaaS	*Backend-as-a-service*, tagarakendusteenus. Levinud tagarakenduse funktsionaalsused, mida pakutakse teenusena
CDN	*Content delivery network,* sisulevivõrk
CLI	*Command line interface,* käsurealiides
CSV	*Comma-separated values,* tekstifailitüüp, mis sisaldab komaga eraldatud andmeid
FCM	*Firebase Cloud Messaging,* Firebase'i platvormide ülene sõnumite ja teatiste saatmise lahendus
HTTP	*HyperText Transfer Protocol,* hüperteksti edastuse protokoll
JSON	*JavaScript object notation*, lihtne andmevahetusvorming, mis põhineb JavaScripti alamhulgal
JWT	*JSON Web Token,* internetistandard JSON andmete loomiseks koos digitaalallkirja või krüpteerimisega
MVP	*Minimum viable product,* versioon tootest, millel on minimaalne hulk funktsionaalsust esimestele kasutajatele
npm	Javascripti pakihaldur
SDK	*Software development kit,* arendustarkvara, mis võimaldab arendada konkreetsele platvormile rakendusi
SMS	*Short message service*, lühisõnumiteenus
SQL	*Structured query language,* deklaratiivne interaktsiooni- ja programmikeel relatsiooniliste andmebaaside kasutamiseks
SSL	*Secure Sockets Layer,* krüpteerimisprotokoll
URL	*Uniform Resource Locator,* internetiaadress
UUID	*Universally unique identifier,* globaalselt ühene identifikaator

Sisukord

1 Sissejuhatus 10
1.1 Probleemi püstitus 10
1.2 Lõputöö eesmärk 11
2 Metoodika 13
2.1 Nõuded probleemi lahendusele 13
3 Tagarakendusteenused 14
3.1 Erinevate tagarakendusteenuste võrdlus 15
3.1.1 Google Firebase 15
3.1.2 Supabase 16
3.1.3 Appwrite 18
3.1.4 AWS Amplify 18
3.1.5 Pocketbase 19
3.1.6 Parse 20
3.2 Erinevate teenuste analüüs 20
4 Praktiline osa 22
4.1 Rakendus Greta 22
4.2 Lahenduse leidmine probleemsele rakendusele 22
4.3 Migratsioon paremale lahendusele 23
4.3.1 Supabase projekti ülesseadmine 23
4.3.2 Kasutajate migreerimine 24
4.3.3 Andmebaasi struktuuri loomine 25
4.3.4 Andmete migreerimine 28
4.3.5 Pilvefunktsioonide üle toomine 31
4.3.6 Klientrakenduse kohandamine 32
4.3.7 Kasutajate failide üle toomine 34
4.4 Tulemused 34
Kokkuvõte 36
5 Kasutatud kirjandus 37

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 39
Lisa 2 – Sõitude („rides“) kollektsiooni dokument 40
Lisa 3 – Firebase’i pilvefunktsioon uue maksekaardi lisamiseks..... 42
Lisa 4 – Supabase’i pilvefunktsioon uue maksekaardi lisamiseks 43

Jooniste loetelu

Joonis 1. Turu-uuringute raportite ennustused BaaS valdkonna kasvu kohta [2] [3]. ... 12
Joonis 2. Javascripti funktsioon, mis kutsutakse iga Firestore „chats“ dokumendi kohta. 29
Joonis 3. Kuvatõmmis ühest Firestore „chats“ dokumendist. 29
Joonis 4. Modifitseeritud „chats“ JSON objekt. 30
Joonis 5. „Chats“ objektist eraldatud „chats_participant“ JSON objekt. 30
Joonis 6. SQL päring andmete sisestamiseks 31
Joonis 7. Maksekaardi lisamise komponendi loogika Firebase'i pilvefunktsiooni abil.. 33
Joonis 8. Maksekaardi lisamise komponendi loogika Supabase'i pilvefunktsiooni abil. 34

Tabelite loetelu

Tabel 1. Avatud lähtekoodiga BaaS projektide repositooriumite statistika. 21
Tabel 2. Supabase'i hinnastamusmudelite võrdlus 24
Tabel 3. Firestore’i kollektsioonide ja vastavate PostgreSQL tabelite nimetused ning semantika. 25
Tabel 4. „Ride“ tabeli andmebaasi struktuur. 27
Tabel 5. „Passenger“ tabeli struktuur ja semantika. 28

1 Sissejuhatus

Tagarakendusteenused on pilveteenused, mis pakuvad arendajatele tagarakendustes levinud komponente nagu näiteks autentimine ja andmebaasid. Need tooted on kujunenud serveripoolse tarkvara arenduses arvestatavaks alternatiiviks klassikalistele lahendustele. Pilvepõhised lahendused on võrdlemisi uus nähtus ning koguvad kiirelt populaarsust, kuid sellest tulenevalt on neid ka vähem uuritud ning pikemaajalisi kogemusi nendega on arendajatel märgatavalt vähem kui kümneid aastaid eksisteerinud tavaliste tagarakendustega.

Töö idee on tulnud autori kogemustest tagarakendusteenustega iduettevõttes Greta, kus otsustati rakendus ehitada kasutades Google'i pakutavat Firebase teenust. Tagarakendusteenuse kasutuselevõtt tähendas iduettevõttele oluliselt kiiremini minimaalse elujõulise tooteni (MVP) jõudmist. Hilisemates tootearendusstaadiumites hakkas välja tulema aga mitmeid miinuseid, millega alguses arvestada ei osatud. Seetõttu tekkisid küsimused, kas MVP tsüklist kaugemale vaadates oli tegu õige otsusega ja kas leidub võimalusi kuidas tagarakendusteenuste eeliseid säilitades kaotada või leevendada negatiivseid külgi, mis on tingitud nende kasutamisest.

Selle töö praktilises osas plaanib autor viia läbi antud rakenduse migreerimise analoogsele vabavaralisele lahendusele. Peale praktilise osa tegemist saab autor antud töös kirjeldada ja analüüsida taolise protsessi keerukust, mahtu ja seda kõike arvestades ka migreerimise otstarbekust. Lõpuks annab see võimaluse võrrelda kaht funktsionaalselt sama kuid erineval lahendusel põhinevat rakendust.

1.1 Probleemi püstitus

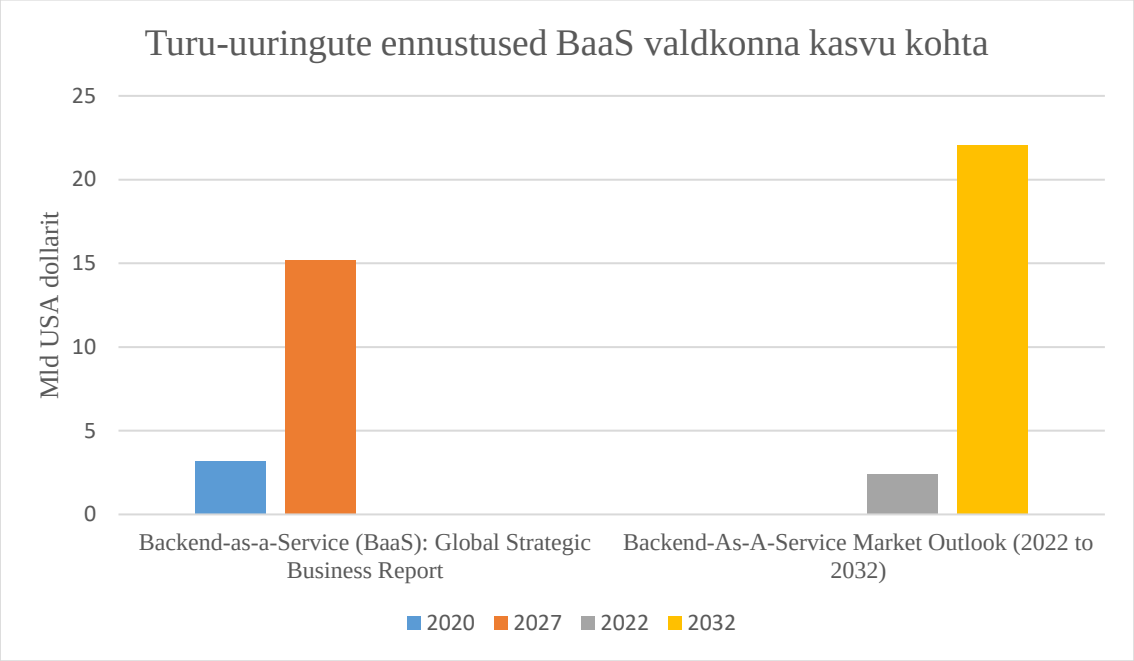
Tagarakendusteenused (BaaS) on pilveteenuste platvormid, mis pakuvad arendajatele võimaluse ühendada oma veebi- ja mobiilirakendused pilveteenustega läbi arendustarkvarade (SDK) ja API-liideste. See kaotab vajaduse spetsiaalse tagarakenduse arenduseks arvestades, et tagarakenduste komponendid tihti korduvad oma funktsionaalsuselt. Levinuimaks näiteks on autentimissüsteemid, mis on vajalikud

praktiliselt igal rakendusel, mis tähendab, et mõistlik on selliseid süsteeme taaskasutada. Sellega võidab arendusprotsess ajas ja keerukuses, andes järele võimalikes erilahendustes ja skaleeritavuses, sest pakutava teenuse kasutamisel sõltutakse teenusepakkuja pakutavast ja otsustest. See tähendab, et näiteks hiljem väljakujunevaid ebatavalisemaid ärinõudeid on raskem või üldse võimatu täita. Skaleerimisel võib see ka rahaliselt kulukaks muutuda, sest mahtude suurenemisel on arendusmeeskonnal vähem võimalusi protsesside efektiivsemaks muutmisel või teenusepakkujate vahetamisel. Lisaks on ka näiteid ajaloost [1], kus teenusepakkuja otsustab oma teenuse sulgeda, mis võib teenuse tarbija selle tulemusel väga suure koormuse alla panna, et kiirelt leida alternatiive oma rakenduse töö tagamiseks. Selliste ohtudega on raske arendusprotsessi alguses arvestada ja probleemid võivad ilmneda arendusprotsessis hiljem, mil koodibaas on suurem ja loogikat rohkem. Sellest tulenevalt on kõvasti raskem raskustega kohaneda või näiteks kasutatavaid tehnoloogiaid/teenuseid muuta. Lisaks tuleb arvestada sellega, et kommertslikke lahenduste pakkujate eesmärk on oma teenust ja teisi selle ettevõtte tooteid müüa, mis tähendab, et nende huvides ei pruugi olla näiteks alternatiividele migreerimise toetamine või lihtsustamine, teiste konkureerivate pakkujate lahendustega integreerimine ning majutamise võimaldamist mujal kui teenusepakkuja enda pilveteenustes. Lisaks sellise teenuse kasutamisel rakenduse teenusepakkujaga täielikule sidumisele kaotab see ka võimaluse arendustiimidel pakkuda oma rakendust lisaks SaaS mudelile kohapealse paigaldusega (ingl k *on-premise deployment*). Kuigi SaaS rakenduse pakkumise mudel on kasvamas suure kiirusega [2], on uuringute põhjal 92% ettevõtete sõnul nõudlus kohapealse paigaldusega rakendustele endiselt kasvamas [3].

1.2 Lõputöö eesmärk

Rakenduse tehnoloogia valikul on tähtis valitud tehnoloogiatest omada võimalikult põhjalikke teadmisi. Suhteliselt uue tehnoloogiana on alust arvata, et tagarakendusteenustest on teadmised pigem arendajate seas väiksemad kui oleks soovituslik. Selle töö raames kavatseb autor kaardistada hetkel populaarseimate tagarakendusteenuste omadusi, nende kasutuselevõtu ohte ja võimalusi. Lisaks plaanib autor võrrelda kommertslikke ja avatud lähtekoodiga lahendusi ja hinnata ühelt teisele migreerimise tasuvust juhul kui selle töö analüüsi tulemusena saab järeldada, et üks leevendaks teise nõrkuseid. Migratsiooni teostamine võimaldab luua detailse migratsiooniplaani millest saab juhinduda teistes sarnastes projektides.

Joonis 1 illustreerib turu-uuringute raportite ennustusi valdkonna kasvu osas järgmise 10 aasta jooksul. Neist saab järeldada, et tagarakendusteenused on tulevikus aina enam arvestatav tehnoloogia ja sellesse arendus- ja ajaressursi investeerimine on väärt tegevus. Teisalt on valdkonna suhteliselt noorest east tulenevalt varasemate kogemuste ja kättesaadava informatsiooni vähesuse tõttu seda lihtsam teha vigu nende kasutamisel. Selle töö eesmärk on koguda ja esitleda informatsiooni, mis võiks olla tulevikus arendusmeeskondadele väärtuslik allikas tehnoloogiate valikul.



Joonis 1. Turu-uuringute raportite ennustused BaaS valdkonna kasvu kohta [4] [5].

2 Metoodika

Käesolevas peatükis kirjeldab autor lõputöös planeeritavat metoodikat töö eesmärgi saavutamiseks.

2.1 Nõuded probleemi lahendusele

Lõputöös püsitatud probleemi lahenduse leidmisel tuleb antud probleemi analüüsida. Osa sellest on reaalsuses kasutusel oleva rakenduse näitel nõuete koostamine, mis aitab hoida lahenduse skoopi ja reaalset rakenduslikkust.

Nõudeid on võimalik koostada konsulteerides tootega seotud isikutega. Antud isikuteks sobivad nii tooteomanikud, tarkvaraarendajad (soovitatavalt vanemarendajad või arhitektid) kui ka toote tellijad (ingl k *stakeholder*) ja kliendid. Lisaks on kasulikuks allikaks varasemad projekti materjalid ja tehniline dokumentatsioon, mis aitab projekti sisust aru saada. Selle tulemusena saab koostada dokumentatsiooni, millele toetudes saab realiseerida planeeritud probleemi lahenduse.

3 Tagarakendusteenused

Tagarakendusteenustel on mitmeid eeliseid, miks arendustiimid nende kasuks otsustavad. Järgnevalt on toodud välja peamised eelised, mida üldiselt tagarakendusteenustele omistada saab:

- Kiirem arendus – valmis komponentide kasutamine säästab arendustiimile väärtuslikku aega, mida saab selle asemel kulutada antud rakendusele spetsiifiliste funktsioonide arendamisele või lühendada arendustsükli aega tervikuna.

- Majutatud tagarakendusteenuste pakkujad võimaldavad skaleerimist – ressursse jaotatakse vastavalt nõudlusele, mis tagab rakenduse kättesaadavuse ja toimivuse ilma suurema lisatööta. Lisaks makstakse vaid selle jõudluse eest, mida reaalselt kasutatakse, mis võib tähendada paremat kuluefektiivsust kui seda on näiteks oma serverite majandamine.

- Valmis komponentide kasutamine nagu autentimine ja autoriseerimine võib olla turvalisem kui isetehtud lahendustel, mis vähendab andmelekete ja küberrünnakute tõenäosust ja ohtlikkust.

Tagarakendusteenuste kasutusele võtmisel tuleb arvestada ka mitmete puudustega:

- Sõltuvus teenusepakkujast – rakendus hakkab paratamatult sõltuma valitud teenusepakkujast ja selle pakutavatest teenustest. Valitud teenusepakkuja otsused võivad otseselt mõjutada rakenduse elukäiku ja erinevate teenuste ja pakkujate vahel migreerimine ei pruugi lihtne olla.

- Limiteeritud kohandatavus – arendajatel on vähem kontrolli tagarakenduse funktsionaalsuse üle, mis võib kujuneda suureks probleemiks kui ilmneb spetsiifilisemaid äri- või arendusvajadusi.

- Kulukus – skaleerimisel võib teenusepakkuja hinnastamispoliitika kujuneda kulukamaks kui isemajutatud lahenduse puhul. Lisaks pole kaitset hinnastamise muutuste eest.

3.1 Erinevate tagarakendusteenuste võrdlus

Selles peatükis tutvustab ning võrdleb autor erinevaid levinumaid tagarakendusteenuseid, et leida parim lahendus näitena toodud rakenduse probleemide lahendamiseks. Sügavamaks uurimiseks valitud projektid on valitud populaarsuse järgi juhindudes avatud lähtekoodidega projektide puhul nende repositooriumite meeldimiste ja kaastööliste arvust võttes samal ajal arvesse projekti vanust. Lisaks on võetud võrdlusesse Google Firebase, mis on Google'i otsinguterminite statistika põhjal suure vahega kõige populaarsem taoline teenus ning on lisaks käesolevas töös uurimise all oleva rakenduse kasutatud teenus. AWS Amplify on võetud uurimise alla, et oleks ka teine suletud lähtekoodiga projekt valikus.

3.1.1 Google Firebase

Google loodud Firebase teenus on üks populaarsemaid tagarakendusteenuste platvorme, mis põhineb tugeval Google Cloud pilveinfrastruktuuril. Pakutakse laia valikut võimalusi, mis aitavad arendajatel arendada mobiili- ning veebirakendusi, neid testida, lansseerida, monitoorida ja analüüsida.

Peamised pakutavad teenused [6]:

- Autentimine: Google Cloud Identity Platvormil põhinev täielik autentimissüsteem, mis pakub võimalusi kasutajate registreerimiseks, haldamiseks ja autentimiseks ka paljude kolmanda osapoolte teenuste abil. Võimalik on ka kasutada turvalisuse lisamiseks mitmetasandilist autentimist (ingl k *multifactor-authentication)* SMS-ide abil.

- Pilvefunktsioonid: võimalus implementeerida serveripoolset loogikat, mis jooksutatakse turvalises Node.js keskkonnas Google'i pilveteenustes. See kaotab vajaduse arendustiimil hallata enda servereid ning skaleerimine on automaatne. Neid funktsioone saab integreerida kergelt Firebase'i teiste teenustega nagu näiteks Firestore andmehoidla võimaldades käivitada funktsioonid toimunud muudatustele Firestore'i andmetes ja HTTP päringute vastuseks klientrakendusest.

- Firestore: NoSQL andmebaas, mis võimaldab hoida, sünkroniseerida ja pärida andmeid hõlpsasti kasutades firmaomast (ingl k *proprietary*) pärimisliidestust.

Erinevalt klassikalisest HTTP päring-vastus mudelist, sünkroniseeritakse andmeid automaatselt, mis tähendab, et seotud klientrakendustes toimub praktiliselt reaalajas andmete uuendamine. Firestore andmebaasil puudub klassikaline andmebaasiskeem nagu see on relatsioonilistes andmebaasides, mis annab rohkem vabadust objektide hoiustamisel. Olemas on võimalused luua andmete pärimiseks, sorteerimiseks ja filtreerimiseks suhteliselt keerulisi päringuid.

- Majutus: Firebase'i *Hosting* teenus võimaldab veebilehti ning rakendusi majutada Google pilveinfrastruktuuris pakkudes lisaks ilma konfiguratsioonita globaalset sisuedastusvõrgu (CDN) teenust, SSL sertifikaati ja palju muud, mis on kasuks rakenduse haldamisel ja kasutajateni viimisel.

- Firebase Cloud Messaging: teenus sõnumite ning teatiste (ingl k *notifications*) saatmiseks iOS, Android ja veebiplatvormidel. Erinevad platvormid kasutavad teatiste saatmiseks oma liidestusi ning ühtne liidestussüsteem nende saatmiseks üle mitme platvormi on väga väärtuslik arendusprotsessi lihtsustamiseks.

- Cloud Storage: teenus andmete ja failide hoidmiseks Google'i pilvemajutuses.

- Firebase Machine Learning: masinõppeliidestus, mis võimaldab treenida ja implementeerida masinõppe mudeleid oma rakenduses. Teenus hoolitseb mudeli majutamise ja rakendusse serveerimise eest.

Firebase'i kasutuselevõtmisel tuleb arvestada, et spetsiifilisema funktsionaalsuste loomine on keeruline või võimatu ning kuigi Google pakub tasuta hinnastamismudelit, siis skaleerimisel tasuta mahte ületades tõusevad kulud kiirelt ning arendustiimil on vähe kontrolli kulude üle. Lisaks täielik sõltuvus teenusepakkujast ning puuduv kontroll infrastruktuuri üle on suur risk rakendusele.

3.1.2 Supabase

Supabase on avatud lähtekoodiga tagarakendusteenuste platvorm, mis pakub mitmeid vajalikke teenuseid rakenduste arenduseks. Supabase projekt on loodud 2020. aastal, mis tähendab, et see on küllaltki noor, kuid on see-eest kiirelt saavutanud tugeva kogukonna arendajaid, kes aitavad kaasa toote kiirele arengule. Seda ilmestab hästi toote GitHubi repositooriumi leht, kus on näha, et toodet on aidanud arendada üle 700 arendaja ning

projekti on meeldivaks märkinud pea 50000 GitHub kasutajat [7]. Kuigi Supabase pakub läbi kolmanda osapoole oma teenuste majutamist, siis on võimalus teenuse majutamine täielikult või osaliselt ka enda kanda võtta kasutades selleks Dockerit. Lisaks on Supabase'il korralik veebipõhine kasutajaliides, kus saab Supabase'i teenuseid, andmeid ja kasutust hallata ja monitoorida.

Peamised teenused [8]:

- PostgreSQL andmebaas: Supabase põhineb relatsioonilisel andmebaasil PostgreSQL. See on suur eelis arendajatele, kes on harjunud klassikaliste SQL andmebaasidega, sest varasemad teadmised SQL andmebaaside arendusest tulevad kasuks ja võimaldavad neid rakendada, sest Supabase'i loodud liidestuse all on täiemahuline PostgreSQL baas, millele saab ka ise ühenduse luua ja seeläbi hallata endale tuttava andmebaasikliendi abil. Sellegipoolest on võimalik andmebaasi ja andmeid hallata ka läbi Supabase'i brauseripõhise töölaua kasutajaliidese. Lisaks PostgreSQL andmebaasi funktsioonidele on Supabase'i platvormile lisatud ka lisafunktsioone nagu näiteks reaalajas andmete jälgimine sarnaselt Firebase'i Firestore'ile. Veel pakutakse andmete varukoopiate tegemist, CSV ja Excel failidest andmete importimist ning automaatselt luuakse tabelitele dokumenteeritud arendusliidestus (API).

- Autentimine: Supabase pakub rohkete võimalustega autentimisplatvormi. Projekti ja andmebaasi loomisel luuakse automaatselt kasutajate skeema andmebaasi ning on ka modifitseeritav. Kasutajate autentimiseks on lisaks tavalisele kasutajanimi-parool meetodile ka telefoni ning mitmete kolmandate osapoolte autentimissüsteemide kasutamise võimalus. Andmebaasi andmete ligipääsetavust saab seadistada vastavalt vajadusele kompleksete turvapoliisidega.

- Edge Functions: Supabase'i pilvefunktsioonide teenus, mis on loodud serveripoolse funktsionaalsuse loomiseks. Funktsioonid käitatakse lõppkasutajale geograafiliselt võimalikult lähedal, et tagada võimalikult väike latentsusaeg (ingl k *latency*). Funktsioone saab kirjutada Typescriptis ning käitatakse kasutades Deno.js käitusmootorit.

- Supabase Storage: failide majutamisteenus, mille turvalisust on võimalik hallata erinevate ligipääsupoliisidega. Haldamiseks on olemas täieväärtuslik veebibrauseri kasutajaliides.

Supabase'il puuduvad mõned vähemolulised teenused, mida pakub näiteks Google Firebase. Puudu on näiteks teatiste saatmise teenus ja analüütika ning monitoorimisvõimalused jäävad alla Google Firebase'i pakutavale, kus on suurt rõhku pandud ülevaatlikule brauseripõhisele kasutajaliidesele ning tehnilisem informatsioon on saadaval Firebase'i platvormi toetavas Google Cloud keskkonnas. Samas, arvestades kui täiemahulise teenuse on Supabase'i tiim vaid paari aastaga ehitanud, siis ettevõtte 2022. aasta lõpus lõpetatud B-seeria investeerimisring 80 miljoni dollariga [9] ning tugev kogukonnatugi näitavad, et tegu on platvormiga, millele saab oma arenduses toetuda ja millelt tulevikus veelgi rohkem võimalusi oodata.

3.1.3 Appwrite

Appwrite on avatud lähtekoodiga pilveteenuste platvorm, mis pakub laia valikut teenuseid ja tööriistasid mobiili- ja veebirakenduste arenduseks. Antud projekt keskendub tugevale API-liideste funktsionaalsusele, mis teeb selle heaks valikuks projektides, kus on eesmärk ehitada API-liideseid või integreerida mitmete teiste liidestega. Appwrite'i ühest suurimaks tugevuseks on võimalus kirjutada pilvefunktsioone mitmetes keeltes: Node.js, PHP, Python, Ruby, Deno, Dart, Swift, .NET, Kotlin, Java ning C++ [10]. Appwrite on tasuta, teenust peab arendaja ise majutama kasutades näiteks Dockerit, aga töö kirjutamise hetkel on arendusstaadiumis ka Appwrite enda pilvemajutusteenus, mis lihtsustaks oluliselt teenuse ülesse seadmist. Serverile ligipääsemiseks pakutakse eri protokolle: GraphQL andmete pärimise protokoll, REST API-liides ja reaalajas sündmustele reageerimise liidestus. Appwrite serveri jaoks on loodud mitmeid arendustarkvarakomplekte eri keeltele, mis on kindlasti kasulik olukordades, kus migreeritakse sellele teenusele osaliselt või on limiteeritud infrastruktuuri otsused [11].

3.1.4 AWS Amplify

Amazoni pilveinfrastruktuuril (AWS) põhinev Amplify platvorm on täielik lahendus arendajatele loomaks, lansseerimaks ja majutamaks taga- ning esirakendusi [12]. Sarnaselt Firebase'ile, mis toetub Google Cloud laiale pilveteenuste valikule, võidab Amplify tugevast Amazoni pilveteenuste platvormist, millele see arendajatele

lihtsustatud ligipääsu annab. Samaaegselt tähendab see teenusepakkujast täielikku sõltuvust, seda nii rakenduse infrastruktuurist, hinnastamisest kui ka teenusepakkuja otsustest ja pakutavast funktsionaalsusest.

Peamised teenused:

- Autentimisteenus: AWS Cognito autentimisplatvormil põhinev Amplify Auth teenus pakub täielikku valikut funktsioonidest, mida võiks vaja minna süsteemi arendusel. Olemas on kolmandate osapoolte läbi autentimine, mitmetasandiline autentimine ja põhjalik ligipääsude haldamisvõimalus koos kasutajagruppide ja seadistavate poliisidega [13].

- Andmebaas: AWS DataStore põhineb AWS DynamoDB NoSQL andmebaasil. Tegu on sarnase võti-väärtustega dokumentidel põhineva platvormiga nagu Firebase'i pakutav Firestore. Tugevaks eeliseks on pakutav klientseadmel andmete hoiustamine, mis annab võimaluse rakendusel saada ligipääsu andmetele ka võrgust lahus töötamise režiimis ning andmed sünkroniseeritakse pilvega kui internetiühendus taastub [14].

- Failide hoiustamine: AWS Amplify Storage pakub lihtsat ligipääsu Amazon S3 hoiustamissüsteemile.

- Kasutajaliidese komponentide teek: Amazon on loonud kiiremaks esirakenduse arenduseks ka valmis komponendid, mida saab kasutada kasutajaliidese arenduses ning esirakenduse liidestamisel tagarakendusega.

3.1.5 Pocketbase

Pocketbase on avatud lähtekoodiga, isemajutatud tagarakendusteenus, mille suurimaks müügiargumendiks on võimalus kogu tagarakendus seada üles ühes failis. See demonstreerib loodud platvormi lihtsust ja kiiret ülesseadmisaega. Teenus pakub SQLite andmebaasi koos JavaScripti ja Dart kliendirakenduse liidestusteekidega, autentimisteenust koos kolmandate osapoolte autentimisteenuste integratsiooniga ning failide majutamise liidestust, mille saab konfigueerida lokaalse failisüsteemi või endale sobiva S3 majutusteenuse peale. Olemas on mitmed sündmuste päästikud (ingl k *trigger)* (näiteks andmebaasis andmete muutused), millele saab kirjutada JavaScripti või Go keeles funktsionaalsust. Platvormi kasutajaliides nagu ka platvorm ise on minimalistlik

ja ilmselt jääb suuremate projektide jaoks liiga lihtsaks, kuid väiksemateks (hobi)projektideks on antud platvormil kindlasti müügiargument. Pocketbase, mis on alustatud alles 2022. aasta juulis ja on täielikult arendatud kogukonna poolt, on alles üsna varajases arendusfaasis, kuid on juba kasvatanud arvestatava kogukonna, mida on näha ka vaadates projekti GitHubi lehte, kus on projekti meeldivaks märkinud 2023. aasta 22. aprilli seisuga 22400 kasutajat ja projekti on aidanud arendada 33 arendajat [15].

3.1.6 Parse

Parse on avatud lähtekoodiga tagarakendusteenus. Parse oli varasemalt populaarne kinnise lähtekoodiga projekt, kuni 2016 aastal selle omanik Facebook teatas projekti lõpetamisest. Selle tulemusena muutis Facebook projekti avatud lähtekoodiga projektiks [16], mis kogus endale kiirelt tugeva kogukonna kaastöölisi, kes on projekti edasi arendanud [17]. Projekt on suunatud peamiselt mobiilirakenduste tagarakenduse funktsionaalsuse pakkumiseks, kuid sobib ka näiteks veebirakenduste jaoks. Projekt ühildub PostgreSQL ja MongoDB andmebaasidega, aga erinevalt paljudest teistest projektidest tuleb see arendajal ise üles seada ning seda hallata. Pakutakse autentimist, andmete- ja failidehoiustamist, mobiili teatiste saatmist ning brauseripõhist töölauavaadet. Projektil on suur hulk klienditeeke liidestumaks kolmanda osapoolte teenustega. Majutamine on arendaja enda kanda, selleks sobib igasugune infrastruktuur, kus töötab Node.js käitusmootor [18]. Parse on ainuke vaadeldav vabavaraline BaaS platvorm, millel on teatiste saatmise funktsionaalsus, kuid ka see vajab Firebase'i projekti. Siiski on see lihtsam, kui integreerida käsitsi Firebase'i teatiste saatmise teek.

3.2 Erinevate teenuste analüüs

Projekti populaarsus on teenuste valikul tähtis parameeter, sest see indikeerib projekti jätkusuutlikust ning oodatavat kiirust uue funktsionaalsuse arendamisel. Avatud lähtekoodiga projektide populaarsust arendajate seas saab hinnata võrreldes nende GitHub koodihoidlate lehtedel väljatoodud statistikat.

Tabel 1 on väljatoodud peatükis 3.1 uuritud projektide kohta käivad peamised indikaatorid projekti populaarsuse hindamiseks. Kaastööliste arv (GitHub lehel ingl k „*contributors*“) on GitHubi lehel arv kasutajaid, kes on panustanud projekti arendusse. Meeldimiste arv viitab kasutajate arvule, kes on projektile andnud positiivse hinnangu (GitHubi lehel ingl k „*stars*“). Harutuste arv (GitHubi lehel ingl k „*forks*“) viitab

kordadele, mil antud projektist on tehtud koopia, et projektist midagi põhiprojektist erinevat arendada või eksperimenteerida mõjutamata põhiprojekti kulgu. Projekti alguskuupäev on ametliku kuupäeva puudumisel võetud koodihoidla esimese koodimuudatuse pealt.

Tabel 1. Avatud lähtekoodiga BaaS projektide repositooriumite statistika.

Nimi	**Projekti alustamise kuupäev**	**Kaastööliste arv**	**Meeldimiste arv**	**Harutuste arv**
Parse	2013 (lähtekood avalikustatud 2016. aasta mais)	303	20200	4700
Supabase	November 2019	727	49700	3700
Appwrite	August 2019	281	31000	2700
Pocketbase	Juuli 2022	33	22900	915

Numbreid võrreldes on selgelt populaarseim teenus arendajate seas töö kirjutamise hetkel Supabase. Suur kaastööliste ning meeldimiste arv suhteliselt noore projekti kohta annab kindlust, et projekt jätkab vähemalt sama kiirelt arengut ning seda ei ohusta väljasuremine. Märkimisväärne on ka Pocketbase'i lühikese aja jooksul kogutud meeldimiste arv, millest juhindudes võib loota, et ka kaastööliste arv kasvab jõudsalt ja võib tulevikus arvestatavaks valikuks osutuda. Hetkel on Pocketbase siiski liiga algusjärgus oma arendusprotsessis, et tehnoloogia valikul seda kaaluda kaalukamates projektides kui seda on hobiprojektid. Supabase'i puhul tuleb arvestada, et see põhineb Deno.js käitusmootoril. See tähendab, et Node'i sisseehitatud funktsioonid on puudu Deno't kasutades. Pilvefunktsioonide kirjutamisel tähendab see ka oluliselt väiksemat teekide valikut kui märkimisväärselt arenenumal Node.js'i pakihalduri „npm" registris. Selle probleemi leevendamiseks on Deno'l välja tulnud hiljuti võimalus kasutada enamus Node'i sisseehitatud standard-teeki ning „npm" pakke [19]. Siiski tuleb arvestada, et tegu on veel arenduses funktsionaalsusega ning kõikide teekide töötamine pole garanteeritud. Projektides, kus on suur sõltuvus Node.js teekidest, on seetõttu soovituslik kaaluda tagarakendusteenust, mis töötab Node.js käitusmootoril nagu Parse või Appwrite. Appwrite on hea valik tiimidele, mille kompetents on teises keeles kui JavaScript, olles ühilduv kõigi peamiste tagarakenduste keeltega.

4 Praktiline osa

Selles peatükis analüüsib autor rakendust mis kogeb probleeme tulenevalt kommertsliku tagarakendusteenuse kasutusest ning kirjeldab läbi viidud migratsiooni paremale lahendusele hindamaks sellise protsessi tasuvust ning keerukust.

4.1 Rakendus Greta

Käesolevas töös on näidiseks võetud iduettevõtte Just Ride Technologies linnadevaheliste sõitude jagamise rakendus Greta. Rakendus on loodud kasutades React Native raamistikku, mis võimaldab ühe koodibaasiga arendada rakendust nii iOS kui ka Android operatsioonisüsteemiga telefonidele kasutades veebiarenduses levinud raamistikku React.js. Rakenduse tagarakenduse loogika on täies mahus Google Firebase'i platvormil implementeeritud, selline otsus tehti arendajate varasema kokkupuute tõttu sellega ning kuna tegu on populaarseima omataolise teenusega. Selle kasutuselevõtt võimaldas arendusmeeskonnal kiirelt välja tulla funktsionaalse rakendusega investeerides tagarakenduse arendusse kuluva aja selle asemel esirakenduse funktsionaalsuse ja kasutajaliidese arendusse. Projektis ilmnesid aga arendusprotsessi edenedes probleeme, mis olid tingitud Firebase'i limitatsioonidest, mida ei osatud projekti alguses näha. Kasutajate hoiustamise objektide struktuur oli muutmatu, mis tähendas vajadust duplitseerida kasutajate kohta käivaid andmeid eraldi.

4.2 Lahenduse leidmine probleemsele rakendusele

Rakenduse Firebase projekti analüüsides selgub, et rakendus kasutab autentimisteenust, Firestore andmebaasi, pilvefunktsioone ning teatiste saatmise teenust. Autentimist pakutakse läbi Facebooki ja Apple ID teenuste. Rakenduses on tähtsal kohal reaalajas andmete liikumine, kuna rakenduses näeb teiste kasutajate sõitusid ning olemas on (grupi)vestlusfunktsioon. Kuna rakendus pole aktiivses kasutuses, siis ei pea tagama rakenduse tööd, st pole vajadust üleminekuperioodile, kus uuendamata kasutajate rakenduste jaoks on vajalik hoida töös Firebase tagarakendus ja välja mõelda andmete sünkroniseerimissüsteem uue ja vana süsteemi vahel. Uus lahendus peaks olema võimalusega iseseisvalt majutada vastav teenus. Avatud lähtekoodiga projekt tagab suurema kontrolli rakenduse infrastruktuuri üle, kaotab sõltuvuse teenusepakkuja

otsustest ning annab suurepärase võimaluse lähtekoodtarkvara arengule kaasa aidata. Peatükis 3 analüüsitud teenustest on võimaluste poolest tugevaim pakkumine AWS Amplify teenusel, kuid selle välistavad sarnaselt kaasnevad probleemid nagu *vendor lock in* ja sõltuvus teenusepakkujast, mis praegusel Firebase teenusel põhineval lahendusel eksisteerivad. Seetõttu jäävad valikusse Supabase, Appwrite, Parse ning Pocketbase. Pocketbase projekt pole veel täiemahulise teenuseni jõudnud ning projekti tiimi sõnul pole enne versioon 1-ni jõudmist garanteeritud tagasiühilduvus [15], mis võib tähendada ebatarvilikku lisatööd projekti arenedes. Supabase ja Appwrite pakutavat eristab Supabase'i põhjalikum töölaua kasutajaliides, PostgreSQL andmebaas (mis annab võimaluse tulevikus lihtsamini andmed tagarakendusteenusest täiesti eraldi, enda hallatud andmebaasi viia), tugevam kogukonnatugi ja aktiivsem arendus. Supabase'i valides tuleb arvestada Deno.js käitusmootorile üleminekut, mis antud projektis on tehtav, sest sõltuvusi kolmandate osapoolte teekidest on vähe. Seda kõike arvestades otsustati projekti tagarakendus migreerida täies mahus (kus võimalik) Supabase'i teenusele. Ainuke järeleandmine sellise lahenduse valides tuleb teha teatiste saatmises, mida hetkel Supabase'i platvorm ei paku ning teatiste saatmiseks on mõistlik jääda toetuma Google Firebase'i pakutavale FCM teenusele.

4.3 Migratsioon paremale lahendusele

Järgnevalt on kirjeldatud migratsioon näidisrakenduse loogika Google Firebase'i pealt Supabase'ile. See hõlmab Supabase-i projekti üles seadmist ja seadistamist, andmebaasi struktuuri loomist, andmete ja kasutajate migreerimist, pilvefunktsioonide üle toomist Supabase'i ja klientrakenduses Firebase'i raamistikul põhineva loogika välja vahetamist Supabase'i raamistiku vastu.

4.3.1 Supabase projekti ülesseadmine

Supabase'i lehel tuleb uue projekti loomisel seada rakenduse nimi, andmebaasi parool, regioon ning hinnastamismudel.

Regiooniks on mõistlik valida rakenduse Greta kasutajatele lähim saadaolev server, milleks on Supabase'i pakutavatest Kesk-Euroopa regioon, mis asub Frankfurdis. Hinnastamismudeliks saab valida tasuta ning tasulise versiooni vahel, mis maksab 25 dollarit kuus. See erineb Firebase'i hinnamudelist, mis põhineb jooksval hinnastamisel (ingl k *pay-as-you-go*), mis tähendab, et maksta tuleb üle tasuta limiidi minevatest

transaktsioonidest ja hoiustamismahtudest ühikute kaupa. Tabel 2 on välja toodud mõlema mudeli tähtsamad limiidid [20].

Tabel 2. Supabase'i hinnastamusmudelite võrdlus.

Teenus	**Tasuta**	**Tasuline (25 $ kuus)**
Andmebaasi maht	Kuni 500 megabaiti	Kuni 8 gigabaiti
Maht failide hoiustamiseks	Kuni 1 gigabait	Kuni 100 gigabaiti
Ribalaius	2 gigabaiti	50 gigabaiti
Aktiivseid kasutajaid kuus	Kuni 50 000	Kuni 100 000
Pilvefunktsioonide kutsumised	Kuni 500 000 kuus	Kuni 2 000 000 kuus
Logide retentsioon	1 päev	7 päeva
Tugi	Kogukonna tugi	Ametlik, emaili-põhine tugi

Analüüsides projekti Firebase’is kasutatud mahtusid, siis projekti *Google Cloud App Engine Quotas* [21] lehel on näha, et hetkel on andmebaasi suurus 0.06 GB ehk 60 megabaiti. Kasutajaid on rakendusel hetkel 211. Pilvefunktsioonide kasutuse statistikat näeb Firebase’i projekti ülevaate lehel, kus on näha viimase 7 päeva statistika. Antud projektil hetkel aktiivne kasutus puudub, on vaid arenduse käigus tehtud kasutus, mis on kirjutamise hetkel 250. Tehes eelduse, et see on ühe kasutaja tavapärane kasutuse maht, siis võttes praeguse kasutajate mahu ja korrutades see läbi keskmise nädalase kasutusega, saame hinnanguliseks funktsioonide kutsumiste arvuks 52 750. Päeva kohta teeb see ligikaudu 7535 ja kuu kohta 226 071 kutsumist. See jääb esialgu üle kahekordse varuga alla tasuta mudeli seatud limiidile 500 000 kutsumist kuus. Kogukonna tugi on Supabase’i repositooriumit ja foorumit vaadates aktiivne ning kõiki mahtusid võrreldes saab järeldada, et tasuta versioon on antud hetkel sobiv.

4.3.2 Kasutajate migreerimine

Kasutajate migreerimiseks pakub Supabase spetsiaalseid tööriistu, millega teistelt teenustelt kasutajaid üle tuua. Firebase’ist Supabase’i migreerimiseks on eraldi tööriist [22] koos detailse õpetusega [23]. Migreerimiseprotsess koosneb kahest osast: kasutajate eksportimine Firebase-ist üldkasutatavale kujule, antud näite puhul JSON failitüüpi ning JSON-kujul andmete Supabase’i PostgreSQL andmebaasi importimine. Sellised

võimalused üldkasutatavas formaadis andmete kättesaamiseks ja sisestamiseks tähendavad, et BaaS rakenduste vahel liikumine on oluliselt lihtsustatud, seda ka spetsiaalsete tööriistade puudumisel. Selle jaoks, et kasutajate jaoks oleks migratsioon märkamatu, saab märkida Supabase'i autentimisseadetes emaili kinnitamise nõude välja, et Supabase ei hakkaks kasutajatelt seda nõudma, kuna antud süsteemi jaoks on tegu uute kasutajatega. Migreerimise käigus on ka kogu info teiste autentimise pakkujate nagu näiteks Facebook ja Apple kasutajate kohta üle toodud ja Supabase integreerib selle info hästi oma süsteemi. Väliste autentimisteenuste kasutamiseks on vaja enne migreerimist Supabase-is need konfigueerida, ka selleks on Supabase'i dokumentatsioonis vajalik informatsioon välja toodud. Kuna konfiguratsioonid on Firebase-is juba üles seatud, on see protsess lihtne: piisab Firebase'i seadetest vajalike võtmete ja URL-ide ümber kopeerimisest Supabase'i seadistustesse. Väliste pakkujate konfiguratsioonides tuleb vaid ära muuta ümbersuunamise URL-id, mis on teenuseti erinevad.

4.3.3 Andmebaasi struktuuri loomine

Supabase põhineb Postgres'i relatsioonilisel andmebaasil, mis on suur erinevus Firebase'i pakutavast *NoSQL* Firestore andmehoidlast. Andmeid hoitakse dokumentides, mis on jaotatud kollektsioonidesse ning dokumendid on vaba struktuuriga ja koosnevad võti-väärtus paaridest (ingl k *key-value pair*) [24]. SQL andmebaasid nagu Postgres koosnevad kindla struktuuri ja andmetüüpidega veergude ja ridadega tabelitest [25].

Mitterelatsiooniliselt andmehoidlalt üle minek relatsioonilisele SQL tüüpi andmebaasile tähendab andmebaasistruktuuri loomise SQL skripti loomist, kus tuleb defineerida Firestore'is kasutatud kollektsioonid tabelitena, eraldada alamobjektid eraldi tabeliteks, seada andmetüübid väljadele ning luua relatsioonid tabelite vahel kasutades primaar- ning võõrvõtmeid. Tabelis 2 on väljatoodud loodava andmebaasi tabelite nimetused, mis on viidud SQL andmebaasi tabelite nimede konventsioonile [26] ning nende semantika.

Tabel 3. Firestore'i kollektsioonide ja vastavate PostgreSQL tabelite nimetused ning semantika.

Firestore kollektsioon	**PostgreSQL tabel**	**Semantika**
bookings	booking	Kasutajate broneeringud sõitudele
chats	chat	Kasutajate vestluste *parent* tabel

(eksisteeris kasutajate võõrvõtmete listina chats dokumentides)	chat_participant	Mitu-mitmele seos vestluste ja vestluse osalejate vahel
locations	location	Sõitude algus- ja lõpp-punktide asukoha info Google Maps'i teenusest
messages	message	Kasutajate sõnumid
(eksisteeris rides dokumentides kasutajate võõrvõtmete listina)	passenger	Mitu-mitmele seos sõitude ja sõitjate vahel
payment-logs	payment_log	Sõitude eest tehtud maksete ja tagasimaksete tabel
rideAlerts	ride_alert	Kasutajate otsitavate sõitude hoiustamise tabel (juhul kui soovitud sõitu ei leita)
rides	ride	Juhtide lisatud sõidud
users	auth.users	Kasutajad („user" tabel eksisteerib juba Supabase'i poolt loodud *auth* andmebaasiskeemis)
users-public	user_public	Avalik kasutajate informatsioon (profiil)

Firestore andmehoidlal pole kollektsioonidel kindlat struktuuri, mis tähendab, et tabelite struktuuri loomisel tuleb võtta näidiseks üks dokument vastavast kollektsioonist ning tuletada vajalikud väljad ning andmetüübid dokumendi andmetest. Lisas 2 on joonisel toodud kollektsiooni „rides" üks dokument JSON kujul. Kui Firestore'i peamine idee on kollektsioonides hoida ühe vaate jaoks kõik andmed, siis SQL tabelites on mõistlik hoida eri objektid eraldi. See tähendab, et „rides" dokumentidest tuleb luua lisaks kirjed „location" tabelisse nii algus- kui lõpp-punkti jaoks ning sõitjate listi („passengerIds") asemel luua „passenger" vahetabel, mis hoiab endas mitu-mitmele seost sõitjate ja sõitude vahel. Lisaks saab kaotada mitmed väljad nagu näiteks „seatsBooked" ja „seatsConfirmed", sest need andmed saab kätte andmebaasist, pärides „booking" tabelist antud kirje võtmega kirjeid vastavas olekus. Kuigi üheks eelistatuimaks primaarvõtme tüüpideks on UUID või number, siis migreerides Firebase-ist, kus võtmeteks on 20 tähemärgi pikkused tähtnumbrilised (ingl k *alphanumeric*) võtmed, on mõistlik *varchar* või *text* andmetüüpi kasutada, mis lihtsustab relatsioonide hoidmist migratsiooniprotsessi

käigus. Primaarvõtmetel on vajalik seada vaikimisi väärtus ning selleks saab kasutada UUID genereerimise funktsiooni uuid_generate_v4(), mille saab ümber konverteerida *varchariks*. Tabel 4 ja Tabel 5 on tulemuseks saadud tabelite struktuur. Tabeli struktuur on selgelt lihtsam ja lamedam ning seotud objektid on vastavalt vajadusele kättesaadavad lihtsate „JOIN“ klauslitega võõrvõtmete põhjal.

Tabel 4. „Ride“ tabeli andmebaasi struktuur.

Veerg	**Andmetüüp**	**Semantika**
id	varchar(36) (vaikimisi genereeritud UUID varchar tüüpi konverteeritud)	Primaarvõti
seats	smallint	Sõidul pakutavate kohtade arv
price	smallint	Ühe koha hind eurodes
mid_ride_pickup	boolean	Näitab, kas sõidu alguspunkt ning reisijate peale võtmise punktid on erinevad
is_finished	boolean	Näitab, kas sõit on lõpetatud
is_create_handled	boolean	Näitab, kas sõidu loomine on tehniliselt lõpetatud
travel_distance_m	integer	Sõidu distants meetrites
travel_duration_s	integer	Sõidu ajaline pikkus sekundites
travel_duration_traffic_s	integer	Sõidu ligikaudne ajaline pikkus sekundites tihedama liikluse puhul
leave_date	timestamp with time zone	Sõidu alustamise aeg
created_at	timestamp with time zone	Sõidu loomise aeg
finished_at	timestamp with time zone	Juhi märgitud reaalne sõidu lõpetamise aeg
driver_id	UUID	Võõrvõti juhi kasutajale (tabel auth.users)
group_chat_id	varchar(36)	Võõrvõti sõidu vestlusele (tabel chat)
from_place_id	varchar(36)	Võõrvõti sõidu alguspunktile (tabel location)
to_place_id	varchar(36)	Võõrvõti sõidu lõpp-punktile (tabel location)

Tabel 5. „Passenger“ tabeli struktuur ja semantika.

Veerg	**Andmetüüp**	**Semantika**
user_id	UUID	Võõrvõti, mis viitab sõitjale (auth.users)
ride_id	varchar(36)	Võõrvõti, mis viitab sõidule (ride)

4.3.4 Andmete migreerimine

Andmete migreerimiseks on Supabase loonud kaks tööriista [22]:

- Firestore2json: tööriist, mis võimaldab Firestore’ist andmed kätte saada JSON kujul. Lisaks on võimalus kirjutada JavaScript meetodeid, mis võimaldavad muuta seda, kuidas andmed JSON-isse kirjutatakse. See on ülimalt kasulik, et objekt lamedamaks muuta, see tähendab kaotada alamobjektid või luua alamobjektidest eraldi objektid, et sisestada need eraldi seotud tabelisse.

- Json2supabase: tööriist, mis võtab JSON kujul objektide massiivi ja sisestab selle etteantud Postgres adnmebaasi. Tööriista miinus, mis tuleneb sellest, et JSON failitüüp ei oma väärtuste tüüpide kohta eraldi infot, on see, et andmeid osatakse sisestada vaid teatud andmetüüpidena. See tähendab, et sõned sisestatakse *text*-tüübina, numbrid *numeric*-tüübina ja lisaks pole informatsiooni, kas väärtus võib olla *null*.

Iga Firestore’i kollektsiooni kohta kirjutati JavaScripti meetod, mis kutsutakse välja iga kollektsiooni dokumendi kohta. Meetodis saab ümber nimetada väljad SQL andmebaaside nimetamiskonventsioonidele vastavaks, luua Date objekt sisendis olevatest kuupäevade sekunditest ning luua eraldi objektid sisendis olevatest massiividest. Joonis 2 on näitena toodud „chats“ kollektsioonile loodud meetod, mis muuhulgas loob „chat_participant“ JSON massiivi sisendis olnud „passengerIds“ listist.

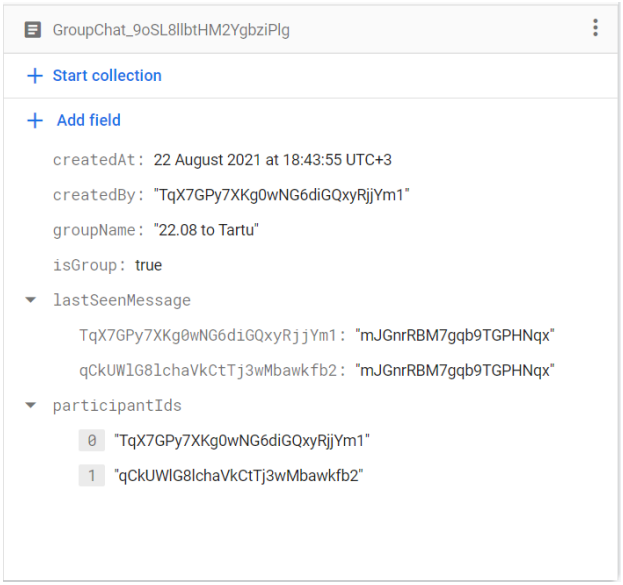
```
module.exports = (collectionName, doc, recordCounters, writeRecord) => {
    for (let i = 0; i < doc.participantIds.length; i++) {
        const chatParticipant = {
            user_id: doc.participantIds[i],
            chat_id: doc.firestore_id,
            last_seen_message_id: doc.lastSeenMessage[doc.participantIds[i]],
            created_at: new Date(doc.createdAt._seconds * 1000),
        }
        writeRecord('chat_participant', chatParticipant, recordCounters)
    }
    doc.is_group = doc.isGroup;
    doc.group_name = doc.groupName;
    doc.created_at = new Date(doc.createdAt._seconds * 1000);
    doc.created_by = doc.createdBy;

    delete doc.isGroup;
    delete doc.groupName;
    delete doc.createdAt;
    delete doc.createdBy;
    delete doc.participantIds;
    delete doc.lastSeenMessage;

    return doc;
}
```

Joonis 2. Javascripti funktsioon, mis kutsutakse iga Firestore „chats“ dokumendi kohta.

Joonis 4 ja Joonis 5 on näidisena välja toodud modifitseeritud „chats“ objekt ja „chat_participants“ JSON objekt, mis eraldati „chats“ objektist ning mille algne kuju Firestore’i andmehoidlas on välja toodud joonisel 3.



Joonis 3. Kuvatõmmis ühest Firestore „chats“ dokumendist.

```
{
  "firestore_id": "GroupChat_9oSL8llbtHM2YgbziPlg",
  "is_group": true,
  "group_name": "22.08  to Tartu",
  "created_at": "2021-08-22T15:43:55.000Z",
  "created_by": "TqX7GPy7XKg0wNG6diGQxyRjjYm1"
}
```


Joonis 4. Modifitseeritud „chats“ JSON objekt.


```
{
  "user_id": "TqX7GPy7XKg0wNG6diGQxyRjjYm1",
  "chat_id": "GroupChat_9oSL8llbtHM2YgbziPlg",
  "last_seen_message_id": "mJGnrRBM7gqb9TGPHNqx",
  "created_at": "2021-08-22T15:43:55.000Z"
}
```


Joonis 5. „Chats“ objektist eraldatud „chats_participant“ JSON objekt.

Peale kõikide kollektsioonide eksportimist sobivatesse JSON massiividesse saab kasutada Supabase’i poolt loodud tööriista Json2supabase, et need objektid Supabase’i andmebaasi sisestada. Tööriista kasutamisel ilmnesid mõned probleemid, millele tuli leida töö käigus lahendus. Esiteks tekitas tööriist rohkem kui 100 objekti korraga sisestamisel duplikaate. Sellele leidus tööriista GitHub koodirepositooriumi probleemide lehel ajutine lahendus kogukonna poolt, mis tähendas INSERT klauslite tükkideks jagamist. Selline ajutine parandus töötas hästi ning kõik objektid sai andmebaasi sisestatud. Seejärel tuli lahendada probleem andmetüüpidega. Tööriist ei võimalda olemasolevatesse tabelitesse andmeid sisestada, vaid loob ise uue tabeli. Kuna JSON failitüübis pole andmete tüübi kohta niivõrd palju informatsiooni kui PostgreSQL andmebaasis veergudel võimalik on, siis loob tööriist tabeli vaid *text* ja *numeric* tüüpidega, mis antud tööriist loodavate tabelite sõne (ka kuupäevade) ning numbriväljadele tüübiks seab, siis on lahenduseks lasta tööriistal luua oma tabel ning sealt omakorda sisestada andmed varem loodud tabelitesse, vajadusel konverteerides andmete tüübid vajalikele tüüpidele. Lisaks võimaldab see leida sisestamisel kasutajate primaarvõtme väärtus võõrvõtmetele, mis viitavad kasutajatele Firestore’is kasutusel olnud identifikaatori kaudu.

Joonis 6 on SQL päring, mis valib andmed ajutisest, Json2supabase tööriista poolt loodud tabelist ja sisestab need peatükis 4.3 loodud „booking“ tabelisse. Sarnaselt toimiti ka ülejäänud tabelite andmetega.

```
INSERT INTO booking (id, seats, state, created_at, driver_id, booker_id,
ride_id, transfer_id, charge_id,
           report_message, report_status, rate_message, rate_point,
payment_method)
SELECT id,
    seats,
    state,
    created_at::timestamptz, (SELECT u.id
                 FROM auth.users u
                 WHERE u.raw_user_meta_data - > 'fbuser' ->> 'uid' =
"driverId"),
    (SELECT u.id FROM auth.users u WHERE u.raw_user_meta_data - > 'fbuser' -
>> 'uid' = "bookerId"),
    "rideId",
    "transferId",
    "chargeId",
    "reportMessage",
    "reportBookingStatus",
    "rateMessage",
    "ratePoint",
    "paymentMethod"
FROM bookings;
```

Joonis 6. SQL päring andmete sisestamiseks.

4.3.5 Pilvefunktsioonide üle toomine

Peale andmete migreerimist implementeeriti serveripoolne äriloogika Supabase'i pilvefunktsioonidena. Mõned pilvefunktsioonid vanas Firebase projektis saab jätta klientrakendusse, sest nende käigus vaid sisestati või küsiti andmeid, mida on lihtne implementeerida ka Supabase'i klienditeegi abil. Keerulisem on olukord meetoditega, mis muuhulgas saadavad ka teatiseid kasutaja seadmele. Sellist funktsionaalsust Supabase (ja enamus teised tagarakendusteenused) ei paku, mis tähendab, et Firebase projektist tasub alles jätta *Firebase Cloud Messaging* (FCM) teenus, mis võimaldab mugavalt teatiseid saata.

Supabase kasutab pilvefunktsioonide jooksutamiseks uuemat, aga vähemlevinud käitusmootorit Deno.js erinevalt Firebase'ist, mis kasutab levinud käitusmootorit Node.js. Mõlemad baseeruvad JavaScript keelel ning FCM teenusel on ka Deno.js jaoks olemas teek, mis tähendab, et funktsioonide üle toomine on enamjaolt üpriski lihtne. Maksetevahendusteenuse Stripe'i teegil puudub Deno jaoks loodud pakk, kuid värskelt loodud Deno.js ühilduvus „npm" pakihalduri pakkidega [19] lahendab ka selle probleemi. Lisaks teekidele, tuleb välja vahetada ka kõik kasutused Node.js sisseehitatud standardteegi moodulitest. Deno.js analoogi puudumisel saab taaskord kasutada Deno.js meeskonna poolt loodud Node.js ühilduvuskihti, mis lihtsustab oluliselt üleminekut Node.js platvormilt.

Supabase’i pilvefunktsioonide projekti üles seadmine seisnes järgmises:

1. Paigaldada Supabase’i käsurea (CLI) tööriist arendaja arvutis
2. Logida sisse tööriista kasutades käsku „supabase login“
3. Initsialiseerida uus Supabase’i projekt kasutades käsku „supabase init“
4. Ühendada lokaalne projekt peatükis 4.3.1 loodud Supabase’i projektiga käsuga „supabase link --project-ref ’projekti referentskood’“

Uue funktsiooni loomiseks on käsklus „supabase functions new hello-world“, mis loob uue funktsiooni põhja koos lihtsa näitega. Loogika tuleb seejärel kopeerida ümber Firebase’i projektist ning vahetada välja Firebase’i teek Supabase’i vastu. See tähendab, et päringu kontekst, andmete pärimine ning kolmandate osapoolte teekide loogika tuleb vastavalt välja vahetada Supabase’i teegi liidestuse vastu. Lisas 3 on näide ühest migreeritavast funktsioonist kasutajale uue maksekaardi lisamiseks, mis kasutab Stripe’i, maksete vahendamise teenuse teeki. Funktsioonis kasutatakse Firebase’i teeki, et pärida kasutaja objekt päringu kontekstis kaasa antud kasutaja identifikaatori järgi, mis on vajalik Stripe’i keskkonnas kasutajate ühendamiseks Greta rakenduse kasutajatega. Loodavas Supabase’i pilvefunktsioonis tuleb päringu kontekst luua teisiti, luues klient objekt päringu „Authorization“ päises kaasa antud JWT veebitunnuse (ingl k *web token)* baasil. Lisasse 4 on lisatud loodud Deno.js funktsioon, mis kasutab Supabase’i klienditeeki sama funktsionaalsuse saavutamiseks. Peale funktsiooni loomist tuleb funktsioon paigaldada Supabase’i pilveprojekti, kasutades käsurea käsklust „supabase functions deploy hello-world“. Seejärel on funktsioon kättesaadav Supabase’i töölaua kasutajaliideses, kus saab funktsiooni statistikat, logisid ja muud vajalikku informatsiooni vaadata. Lisaks on seal loodud käsklus, millega antud funktsiooni saab käsurealt kutsuda funktsiooni testimiseks. Taoline protsess viidi läbi ka ülejäänud pilvefunktsioonidega.

4.3.6 Klientrakenduse kohandamine

Migratsiooni viimase sammuna kohandati klientrakendus, milleks on React Native’i mobiilirakendus iOS-ile ja Androidile, kasutama Supabase’i klienditeeki ning peatükis 4.5 loodud pilvefunktsioone. React Native (ja selle aluseks olev React) on väga levinud raamistik ning Supabase’i dokumentatsioonis on palju spetsiaalseid juhendeid ning

näiteid, kuidas taolistes rakendustes Supabase'i klienditeegi abil autentimise, andmete ja pilvefunktsioonide loogika implementeerida.

Klientrakenduses tuleb kõik Firebase'i teegi funktsionaalsus välja vahetada analoogse funktsionaalsuse vastu Supabase'i teegist. See protsess on ilmselt kõige ajamahukam ning erinevalt andmete migreerimisest sõltub selle mahukus klientrakenduse keerukusest ja suurusest. Mida rohkem on funktsionaalsust rakenduses, seda rohkem on manuaalset tööd. Kuna nii autentimine, andmete pärimise ja pilvefunktsioonide kutsumise loogika on neil sarnane, siis keerukas see ei ole, aga automaatselt seda lahendada ei saa, sest taoline loogika on rakendusse põimitud igal pool. Protsessi käigus otsiti üle projekti viiteid Firebase'i teegi viidetele, mis siis ükshaaval analoogsete Supabase'i meetoditele konverteeriti. Joonis 7 on klientrakendusest võetud funktsioon, mis kutsub lisas 3 välja toodud Firebase'i pilvefunktsiooni, et kasutaja sisendi põhjal lisada kasutajale uus maksekaart.

```
const doCreateCard = functions.httpsCallable('doCreateCard')

const onPressAddCard = async () => {
  const cancelLoading = appStore.requestLoading()
  try {
   const { data } = await doCreateCard({ ...cardInfo })
   console.log(data)

   if (!data.success) {
    Alert.alert('Error', `${data.error.raw.message}`)
   } else {
    Alert.alert('Success', 'Payment card was added successfully!')
    navigation.goBack()
    refeshData()
   }
  } catch (e) {
   throw e
  } finally {
   cancelLoading()
  }
}
```

Joonis 7. Maksekaardi lisamise komponendi loogika Firebase'i pilvefunktsiooni abil.

Supabase'i funktsiooni kutsumiseks tuleb muuta vaid funktsiooni ennast juhul kui funktsiooni parameetrid ning vastuse struktuur on jäetud samaks funktsioonide migreerimise käigus. Joonis 8 on kooditõmmis muudatuste tulemusena loodud funktsionaalselt identsest funktsioonist, mis kasutab Supabase'i teeki.

```
const onPressAddCard = async () => {
  const cancelLoading = appStore.requestLoading()
  try {
   const { data, error } = await supabase.functions.invoke('do-create-card',
{
    body: cardInfo,
   })
   console.log(data)

   if (!data.success) {
    Alert.alert('Error', `${error.message}`)
   } else {
    Alert.alert('Success', 'Payment card was added successfully!')
    navigation.goBack()
    refeshData()
   }
  } catch (e) {
   throw e
  } finally {
   cancelLoading()
  }
}
```

Joonis 8. Maksekaardi lisamise komponendi loogika Supabase'i pilvefunktsiooni abil.

4.3.7 Kasutajate failide üle toomine

Rakenduse Greta kontekstis puudus vajadus kasutajate või muude failide üle toomiseks Firebase failihoidlast Supabase'i, sest rakenduses ei ole kasutajate genereeritud sisu. Sellise vajaduse olemasolu puhuks on Supabase'i kogukonna poolt loodud käsurea tööriistad [27], mis lihtsustavad seda protsessi. Migratsioon on lihtne ning dokumenteeritud ja see koosneb kahest osast:

1. Failid laetakse alla Firebase'i andmehoidlast kohalikku failisüsteemi

2. Failid laetakse üles kohalikust failisüsteemist Supabase'i andmehoidlasse

4.4 Tulemused

Lõputöö eesmärgiks oli kaardistada tagarakendusteenuste valdkond ning selle informatsiooni põhjal leida ühele probleemsele rakendusele parem lahendus ja sellele üleminek läbi teha, et hinnata sellise protsessi keerukust ja teostuslikkust. Käesolevas peatükis leiti analüüsitud tagarakendusteenustest parim sobiv variant, Supabase, ning teostati üleminek Firebase'ilt sellele. Migratsiooni käigus esines probleeme, alustades tööriistade vigasest tööst kuni Deno.js käitusmootoril „npm" teekide kasutamiseni, kuid

tänu aktiivsele kogukonnatoele, mis Supabase'il on, said sellised probleemid kiirelt lahenduse. Kogu migratsioon võttis aega ühele arendajale arvestuslikult 10 täiskohaga päeva, kuid tuleb arvestada, et nii klientrakenduse kohandamine kui ka pilvefunktsioonide ületoomise ajaline pikkus on suhteline rakenduse keerukusest. Andmete migreerimine sõltub vähem rakenduse keerukusest, see tähendab andmete mahust. Keerukamate projektide puhul, eriti projektides, kus on kasutusel palju „npm" pakihalduri teeke, on soovituslik Parse või Appwrite'i platvormi uurida. Deno.js platvorm, eriti selle „npm" pakkide ühilduvuse funktsionaalsus pole veel sealmaal, et saaks kindlusega kasutada suurema kaaluga projektides. Supabase pakkus siiski väga terviklikku lahendust, mis ei jää oma funktsionaalsuselt kuidagi Firebase'ile alla ning migratsiooni tulemusena loodi uus tagarakenduse lahendus mis on samaväärne Firebase'i lahendusega välja arvatud teatiste saatmise funktsionaalsus. Sellist funktsionaalsust on aga Google Firebase'ist väljaspool raske leida, ka näiteks OneSignal teatiste saatmise platvorm, millele on Supabase ka integratsiooni loonud, vajab Firebase'i projekti üles seadmist, et Androidile teatiseid saata.

Supabase'i töölaua kasutajaliides on väga kasulik tööriist rakenduse ülesseadmisel, arendamisel kui ka haldamisel. Valmis koodijuppide genereerimist vastavalt loodud andmebaasimudelile on erakordselt mugav ja kiirendab oluliselt arendusprotsessi. Samas on Supabase'i dokumentatsioon mõnedes teemades ootamatult sõnaaher või on loodud hoopis video teema selgitamiseks, mis on vähemalt autori isiklikust vaatepunktist märksa vähemeelistatud variant korralikule tekstipõhisele tehnilisele dokumentatsioonile.

Migratsiooni tulemusena on Greta rakendus nüüd ühest teenusepakkujast oluliselt vähem sõltuv. Endiselt on rakendus seotud Supabase'i ning nende kasutatava Microsoft Azure pilveteenuste platvormi hinnastamisega, kuid võimalus igal hetkel edaspidi muuta rakenduse majutamise teenust on arendustiimile suure väärtusega. Migreerimisele kulutatud aeg oleks võidud kasutada uue funktsionaalsuse arendamisele, kuid taoline protsess võtaks aina rohkem aega, mida hiljem arendusprotsessis see ette võtta.

Kokkuvõte

Käesoleva töö raames võttis autor eesmärgiks analüüsida ning võrrelda tagarakendusteenuseid. Analüüsi eesmärgiks oli koguda informatsiooni, mis oleks kasulik arendajatele tulevikus rakenduse tehnoloogia valikul. Lisaks hinnati võimaliku probleemse olukorra lahendatavust läbi migratsiooni alternatiivsele lahendusele.

Töö analüüsi osas võeti uurimise alla 6 levinud tagarakendusteenust: 2 kommertslikku ning 4 avatud lähtekoodiga projekti. Töös analüüsiti ning võrreldi omavahel Firebase, AWS Amplify, Parse, Supabase, Pocketbase ning Appwrite teenuseid. Välja toodi teenuste pakutav funktsionaalsus ning kitsaskohad, millega antud teenuse puhul arendaja arvestama peaks.

Töö praktilises osas põhines autor töö teoreetilises osas teostatud analüüsidele, et leida näitena toodud probleemsele rakendusele lahendus ning seejärel viis läbi rakenduse migratsiooni leitud platvormile. Selle läbi viimine andis autorile informatsiooni, millele põhineda, et hinnata sellise lahenduse tasuvust, keerukust ja mahtu. Autor järeldas, et selline protsess on teostatav mõistliku ajaga ning on soovitatav lahendus paljudes olukordades.

Lõputöö teostamise käigus omandas autor uusi teadmisi Deno.js käitusmootorist ning Supabase’i reaalse kasutamise kohta. Töö tulemusena on loodud allikas, millele saavad arendajad põhineda tulevikus rakenduse tehnoloogia valikul.

5 Kasutatud kirjandus

[1] F. Tepper, „Facebook’s Parse developer platform is shutting down today | TechCrunch,“ 31 Jaanuar 2017. [Võrgumaterjal]. Saadaval: https://techcrunch.com/2017/01/30/facebooks-parse-developer-platform-is-shutting-down-today/. [Kasutatud 19 märts 2023].

[2] L. S. Vailshery, „Saas Market Size Worldwide 2024,“ Statista, aprill 2023. [Võrgumaterjal]. Saadaval: https://www.statista.com/statistics/55243/worldwide-software-as-a-service-revenue/. [Kasutatud 10 mai 2023].

[3] Dimensional Research, „The State of On-Prem: Modern Solitions for a Traditional Problem,“ märts 2021. [Võrgumaterjal]. Saadaval: https://f.hubspotusercontent10.net/hubfs/8554162/The%20State%20of%20On-Prem_Modern%20Solutions%20for%20a%20Traditional%20Problem.pdf. [Kasutatud 2 mai 2023].

[4] „Future Market Insights,“ august 2022. [Võrgumaterjal]. Saadaval: https://www.futuremarketinsights.com/reports/backend-as-a-service-baas-market. [Kasutatud 1 mai 2023].

[5] Global Industry Analysts, Inc, „Research and Markets,“ märts 2023. [Võrgumaterjal]. Saadaval: https://www.researchandmarkets.com/reports/3619381/backend-as-a-service-baas-global-strategic. [Kasutatud 1 mai 2023].

[6] „Firebase Products,“ Google, [Võrgumaterjal]. Saadaval: https://firebase.google.com/products-build. [Kasutatud 20 aprill 2023].

[7] „Supabase GitHub repositoorium,“ [Võrgumaterjal]. Saadaval: https://github.com/supabase/supabase. [Kasutatud 20 aprill 2023].

[8] „Supabase dokumentatsioon,“ [Võrgumaterjal]. Saadaval: https://supabase.com/docs. [Kasutatud 18 aprill 2023].

[9] F. Lardinois, „Supabase raises $80M Series B for its open source Firebase alternative,“ TechCrunch, [Võrgumaterjal]. Saadaval: https://techcrunch.com/2022/05/10/supabase-raises-80m-series-b-for-its-open-source-firebase-alternative/. [Kasutatud 20 aprill 2023].

[10] „Functions - Docs - Appwrite,“ [Võrgumaterjal]. Saadaval: https://appwrite.io/docs/functions. [Kasutatud 10 mai 2023].

[11] „Docs - Appwrite,“ [Võrgumaterjal]. Saadaval: https://appwrite.io/docs. [Kasutatud 27 aprill 2023].

[12] „Amplify Libraries - AWS Amplify Docs,“ [Võrgumaterjal]. Saadaval: https://docs.amplify.aws/lib/. [Kasutatud 24 aprill 2023].

[13] „Authentication - Getting started - React Native - AWS Amplify Docs,“ [Võrgumaterjal]. Saadaval: https://docs.amplify.aws/lib/auth/getting-started/q/platform/react-native/. [Kasutatud 24 aprill 2023].

[14] „DataStore - Getting started - React Native - AWS Amplify Docs,“ [Võrgumaterjal]. Saadaval: https://docs.amplify.aws/lib/datastore/getting-started/q/platform/react-native/. [Kasutatud 24 aprill 2023].

[15] „Pocketbase GitHub repositoorium,“ [Võrgumaterjal]. Saadaval: https://github.com/pocketbase/pocketbase. [Kasutatud 21 aprill 2023].

[16] J. MSV, „Facebook's Parse May Be Dead But It Continues To Live Within The FOSS Community,“ Forbes, 29 Jaanuar 2016. [Võrgumaterjal]. Saadaval: https://www.forbes.com/sites/janakirammsv/2016/01/29/facebooks-parse-may-be-dead-but-it-continues-to-live-within-the-foss-community/?sh=5d22ce787d2c. [Kasutatud 1 mai 2023].

[17] „Parse Server GitHub repositoorium,“ [Võrgumaterjal]. Saadaval: https://github.com/parse-community/parse-server. [Kasutatud 1 mai 2023].

[18] „Parse Server Guide | Parse,“ [Võrgumaterjal]. Saadaval: https://docs.parseplatform.org/parse-server/guide/. [Kasutatud 1 mai 2023].

[19] „npm Modules | Manual | Deno,“ [Võrgumaterjal]. Saadaval: https://deno.com/manual@v1.29.2/node. [Kasutatud 5 mai 2023].

[20] „Pricing & Fees | Supabase,“ [Võrgumaterjal]. Saadaval: https://supabase.com/pricing. [Kasutatud 16 aprill 2023].

[21] „Quotas | Google App Engine standard environment docs,“ [Võrgumaterjal]. Saadaval: https://cloud.google.com/appengine/docs/standard/quotas. [Kasutatud 1 mai 2023].

[22] „Firebase-to-supabase tööriista GitHub repositoorium,“ [Võrgumaterjal]. Saadaval: https://github.com/supabase-community/firebase-to-supabase. [Kasutatud 12 aprill 2023].

[23] „Supabase dokumentatsioon - migreerimine Supabaseile,“ [Võrgumaterjal]. Saadaval: https://supabase.com/docs/guides/resources/migrating-to-supabase/firebase-auth. [Kasutatud 4 aprill 2023].

[24] „Cloud Firestore Data model | Firebase,“ [Võrgumaterjal]. Saadaval: https://firebase.google.com/docs/firestore/data-model. [Kasutatud 16 aprill 2023].

[25] „PostgreSQL: About,“ [Võrgumaterjal]. Saadaval: https://www.postgresql.org/about/. [Kasutatud 16 aprill 2023].

[26] E. Drkusic, „Learn SQL: Naming Conventions,“ 28 juuli 2020. [Võrgumaterjal]. Saadaval: https://www.sqlshack.com/learn-sql-naming-conventions/. [Kasutatud 17 aprill 2023].

[27] „Firebase-to-supabase Storage tööriista GitHub repositoorium,“ [Võrgumaterjal]. Saadaval: https://github.com/supabase-community/firebase-to-supabase/tree/main/storage. [Kasutatud 3 mai 2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Jan Joonas Parve

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tagarakendusteenuste analüüs ning Google Firebase rakenduse avatud lähtekoodiga lahendusele migreerimine ", mille juhendaja on Brita Moorus
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

24.04.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Sõitude („rides“) kollektsiooni dokument


```
{
  "driverRef": {
    "_firestore": {
      "projectId": "project-id"
    },
    "_path": {
      "segments": [
        "users",
        "qCkUWlG8lchaVkCtTj3wMbawkfb2"
      ]
    },
    "_converter": {}
  },
  "midRidePickup": false,
  "toPlaceDesc": "Tartu, Estonia",
  "isFinished": false,
  "seats": 3,
  "toPlaceId": "ChIJ9z1d1dg260YREG38GG2zAAQ",
  "createdAt": {
    "_seconds": 1617024726,
    "_nanoseconds": 778000000
  },
  "fromPlaceDesc": "Tallinn, Estonia",
  "fromPlaceDetail": {
    "administrative_area_level_1": "harjucounty",
    "location": {
      "lng": 24.7535746,
      "lat": 59.43696079999999
    },
    "administrative_area_level_1_long_name": "Harju County"
  },
  "driverId": "qCkUWlG8lchaVkCtTj3wMbawkfb2",
  "price": 5,
  "leaveDate": {
    "_seconds": 1617199254,
    "_nanoseconds": 0
  },
  "fromPlaceId": "ChIJvxZW35mUkkYRcGL8GG2zAAQ",
  "toPlaceDetail": {
    "administrative_area_level_1": "tartucounty",
    "administrative_area_level_1_long_name": "Tartu County",
    "location": {
      "lng": 26.7290383,
      "lat": 58.37798299999999
    }
  },
  "groupChatId": "GroupChat_a1ExjKeyE6i78WcmwSdv",
  "fromPlaceRef": {
    "_firestore": {
      "projectId": "project-id"
    },
    "_path": {
      "segments": [
        "locations",
        "ChIJvxZW35mUkkYRcGL8GG2zAAQ"
      ]
    },
```

```
    "_converter": {}
  },
  "isCreateHandled": true,
  "travelInfo": {
    "distance": {
      "text": "183 km",
      "value": 182929
    },
    "duration_in_traffic": {
      "text": "2 hours 15 mins",
      "value": 8071
    },
    "duration": {
      "text": "2 hours 13 mins",
      "value": 7955
    }
  },
  "toPlaceRef": {
    "_firestore": {
      "projectId": "project-id"
    },
    "_path": {
      "segments": [
        "locations",
        "ChIJ9z1d1dg260YREG38GG2zAAQ"
      ]
    },
    "_converter": {}
  },
  "seatsConfirmed": 1,
  "passengerIds": [
    "TqX7GPy7XKg0wNG6diGQxyRjjYm1"
  ],
  "seatsBooked": 0,
  "transferId": "",
  "transferAt": 0,
  "firestore_id": "a1ExjKeyE6i78WcmwSdv"
}
```

Lisa 3 – Firebase'i pilvefunktsioon uue maksekaardi lisamiseks

```
export const doCreateCard = functions.https.onCall(
    async (data: StripeCardStruct, context) => {
        if (!context.auth?.uid) throw new NotAuthenticatedError()
        try {
            const stripeCustomer = await createOrUpdateStripeCustomer({ card:
data, uid: context.auth?.uid })
            return {
                success: true,
                data: stripeCustomer
            }
        } catch (error) {
            return {
                success: false,
                error
            }
        }

    }
)
export const createOrUpdateStripeCustomer = async (
    { card, uid }: { card: StripeCardStruct, uid: string }) => {
    const userDoc = await get(usersCollection, uid);
    const param = {};

    param["card"] = {
        number: card.number,
        exp_month: card.exp_month,
        exp_year: card.exp_year,
        cvc: card.cvc,
        name: card.name
    };
    const newSource = await stripe.tokens.create(param);
    if (userDoc?.data.customerStripeId) {
        // add card to stripe existing user
        await stripe.customers.createSource(userDoc?.data.customerStripeId, {
            source: newSource.id,
            metadata: {
                createdAt: new Date().getTime()
            }
        });
    } else {
        const user = await admin.auth().getUser(uid);
        //create new stripe user
        const newStripeCustomer = await stripe.customers.create({
            email: user.email,
            source: newSource.id
        });
        await update(usersCollection, uid, { customerStripeId:
newStripeCustomer.id })
        return newStripeCustomer;
    }
    return await get(usersCollection, uid);
}
```

Lisa 4 – Supabase'i pilvefunktsioon uue maksekaardi lisamiseks

```
const stripe = Stripe(Deno.env.get("STRIPE_API_KEY"));

export type StripeCardStruct = {
  number: string,
  exp_month: string,
  exp_year: string,
  cvc: string,
  name: string
}

serve(async (req) => {
  try {
   const supabaseClient = createClient(
    Deno.env.get('SUPABASE_URL') ?? '',
    Deno.env.get('SUPABASE_ANON_KEY') ?? '',
    { global: { headers: { Authorization: req.headers.get('Authorization')! }
} },
   )
   const { data } = await supabaseClient.auth.getUser()

   if (!data?.user) {
    return new Response(JSON.stringify({ success: false, error:
error.message }), {
     headers: { 'Content-Type': 'application/json' },
     status: 401,
    })
   }

   const stripeCustomer = await createOrUpdateStripeCustomer(req.json(),
data.user)

   return new Response(
    JSON.stringify(stripeCustomer),
    { headers: { 'Content-Type': 'application/json' } },
   )
  } catch (error) {
   return new Response(JSON.stringify({ success: false, error:
error.message }), {
    headers: { 'Content-Type': 'application/json' },
    status: 500,
   })
  }
})

const createOrUpdateStripeCustomer = async (card: StripeCardStruct, user:
PublicKeyCredentialCreationOptions.user) => {
  const newSource = await stripe.tokens.create(card)
  if (user.customerStripeId) {
   // add card to stripe existing user
   await stripe.customers.createSource(user.customerStripeId, {
    source: newSource.id,
    metadata: {
```

```
      createdAt: new Date().getTime(),
     },
    })
    return await stripe.customers.get(user.customerStripeId);
   } else {
    const newStripeCustomer = await stripe.customers.create({
     email: user.email,
     source: newSource.id,
    })
    supabaseClient.auth.updateUser({ customerStripeId: newStripeCustomer.id })
    return newStripeCustomer
   }
}
```