TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Taniel Tari 179479IADB

Android teegi loomine Rootsi ühistranspordi standardi jaoks

bakalaureusetöö

Juhendaja: Jaanus Pöial
PhD

Tallinn 2024

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: Taniel Tari

09.11.2023

Annotatsioon

BoB (*Biljett- och Betalstandard*) on Rootsi ettevõte Samtrafiken loodud standard ühistranspordi süsteemide ühtlustamiseks. Standard kirjeldab veebiteenuste skeemid, piletipakkide struktuuri ning piletipakkide allkirjastamise jaoks vajalikud tingimused. Kuigi tegemist on Rootsi ühistranspordi standardiga, siis on see kasutusel ka mujal riikides nagu näiteks Eesti ja Soome. Standard ise on hästi dokumenteeritud, kuid avalikke koodinäiteid standardi kasutamisest pole tehtud. Selle lõputöö eesmärk on luua Android platvormile mõeldud teek, mis aitab allkirjastada piletipakke ning seda kuvada ruutkoodina validaatoritele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 7 peatükki, 12 joonist, 0 tabelit.

Abstract

Implementing Android Library for Swedish Public Transport Standard

BoB (*Biljett- och Betalstandard*) is a Swedish transport standard made by Samtrafiken to unify public transport systems. Standard defines schemas for web services, ticket bundle structure and requirements to sign ticket bundles. Even though the standard is made for Swedish public transport, then it is also used in other countries like Estonia and Finland for example. Standard is well documented, but there are no public code examples how to implement the standard. This thesis purpose is to create a library for Android platform that helps to sign ticket bundles and generate ticket bundle barcodes for the validator.

The thesis is in Estonian and contains 22 pages of text, 7 chapters, 12 figures, 0 tables.

Lühendite ja mõistete sõnastik

Android	Tarkvarakomplekt, mis koosneb operatsioonisüsteemist, vahetarkvarast ja baasrakendustest
Aztec Code	Ruutkood, mida kasutatakse optiliselt andmete edastamiseks
CBOR	*Concise Binary Object Representation*, andmevahetusvorming
CI	*Continious Integration*, pidev integratsioon
ECDSA	*Elliptic Curve Digital Signature Algorithm*, asümmeetriline krüpteerimisalgoritm
Gradle	Ehitustööriist tarkvara arenduseks, mis on peamiselt kasutusel Java põhistes projektides
HMAC	*Hash-based Message Authentication Code*, sõnumi autentimiskood
IDE	*Integrated development environment*, programmeerimiseks mõeldud tarkvararakendus, mis sisaldab lisaks tekstiredaktorile ka muid tööriistu
Javadoc	Java keele jaoks loodud dokumentatsioonigeneraator
JSON	*JavaScript Object Notation*, tekstipõhine andmevahetusvorming
JVM	*Java Virtual Machine*, Java baitkoodi interpretaator
JWE	*JSON Web Encryption*, standard krüpteeritud andmete vahendamiseks ja allkirjastamiseks
JWS	*JSON Web Signature*, standard vabalt valitud andmete allkirjastamiseks
JWT	*JSON Web Token*, standardiseeritud tõend väidete vahendamiseks
KDK	*Key derivation key*, võtmetuletusvõti
Markdown	Lihtteksti vormindamise süntaks
OpenAPI	Masinloetav liidese määratluskeele spetsifikatsioon veebiteenuste jaoks
REST	*Representational State Transfer*, tarkvaraarhitektuur, mis seab veebiteenustele reeglid suhtlemiseks
XML	*Extensible Markup Language*, üldotstarbeline märgistuskeel

Sisukord

Jooniste loetelu

1 Sissejuhatus

Ühistransport on igapäevaelu valdkond, mille tegevus on tänu digitaliseerimisele paranenud nii juurdepääsetavuselt kui ka efektiivsuselt. Tänu infotehnoloogia arengule on võimalik jälgida sõidukite asukohti reaalajas, osta pileteid mugavalt oma telefonist või saada teavitusi olukorrast liikluses. Tänapäeval on ühistransport väga oluline osa ühiskonnast, sest see tagab viisi liigelda inimestele, kellel pole seda võimalik teha tervislikel või finantsilistel põhjustel muude vahenditega. Lisaks on mitmed riigid võtnud eesmärgiks säästa loodust tulevastele põlvedele ning selleks on ühistransport üks parimaid viise vähendada kogu maailma kütusekulu, tagada puhtam õhk suurlinnades ja vähendada liiklusummikute teket. Eelnimetatud põhjused on valitsuste jaoks üha rohkem olulisemad ühiskonna heaolu kui ka kulude kokkuhoidmise pärast. Seetõttu on tekkinud soosivad tingimused suuremaks ühistranspordi kasutamiseks ja on kasvanud valitsuste ja ühistranspordi ettevõtete investeeringud infotehnoloogilistesse arengutesse, et lihtsustada ja parandada ühistranspordi kasutamist veelgi.

1.1 Taust

Sageli on ühes piirkonnas tegutsemas rohkem kui üks ühistranspordi teenusepakkuja ning seega on oluline, et teenusepakkujate süsteemid jagaksid omavahel andmeid ja teeksid seda mingil kokkulepitud viisil. Seetõttu algatas Rootsi ettevõte Samtrafiken i Sverige AB initsiatiivi, et luua ühtne viis andmete jagamiseks mitmete teenusepakkujate süsteemide vahel. Initsiatiivist arenes välja standard nimega *Biljett- och Betalstandard* ehk BoB. Standard on avalik, põhjalikult kirjeldatud ning kõigil huvilistel on võimalik aktiivselt lüüa kaasa standardi arengus. Lisaks standardile tegeleb Samtrafiken ka standardit kasutatavate organisatsiooni identifikaatorite väljastamise ja avalikult kuvamisega. See on oluline, et kõik osapooled teaksid, millise ettevõtte piletitega on tegemist.

1.2 Standardite olulisus

Standardid on suure olulisusega, sest nende järgimine tagab stabiilsuse ja koostöövõime tarkvarasüsteemide vahel. Peamine kasu standardite kasutamisel tuleneb sellest, et süsteemid saavad andmeid vahetada neile juba tuntud ja aru saadaval viisil. Ka majanduslikult on kasulik standardite järgimine, et vähendada uute väliste süsteemide liitumisel täiendavaid arendusi. Ettevõtetel, kes järgivad levinud standardeid on lihtsam konkureerida ja palgata uusi töölisi kui neil ettevõtetel, kes seda ei tee.

Üldjuhul on standardite loomisel kaasa löönud palju oma ala professionaale, kes on teinud vastavale teemale sobilikud otsused. Standardite järgimine tekitab usalduse nii lõppklientides kui ka ettevõtete vahel.

1.3 Probleem

Peamine probleem on dokumentatsiooni maht. Vastavad osad on küll põhjalikult kirjeldatud, kuid selle läbi lugemine on ajakulukas ja vajab palju analüüsi. Tänapäeval on enamik tarkvaraprojektidele seatud ajakavad, mis eeldavad kiiret arendust ja seega pole arendajatel aega standardite põhjalikuks läbi töötlemiseks. Kuidas oleks võimalik arendajatel kiiremini standard kasutusele võtta?

Teine probleem on turvalisuse tagamine. Oluline on mitte ohustada kasutaja andmete lekkimist nii moraalsetel kui ka rahalistel põhjustel. Kahjuks paljud tarkvaralahendused keskenduvad peamiselt uute funktsioonide lisamisele ja andmete turvalisele käitlemisele ei keskenduta. Kuidas tagada standardi korrektne ja kasutaja andmeid delikaatselt käitlev rakendamine?

Kolmas probleem on lähtekoodi näidete puudumine. Kuigi see probleem on tihedalt seotud eelnimetatud probleemidega, siis näidete olemasolu tagaks parema ülevaate uutele arendajatele arhitektuurist ja kuidas on üldse võimalik nõutud andmetöötluseid sooritada. Kuidas võimaldada arendajatele parem ülevaade standardi kasutamisest?

1.4 Eesmärk

Selle lõputöö eesmärk on lihtsustada ja kiirendada standardi kasutamisele võtmise arendusprotsessi mobiilirakendustes ning teha seda selliselt, et loodav lahendus käitleks delikaatseid andmeid turvaliselt. Loodav lahendus peaks koosnema eraldiseisvatest osadest, mis võimaldaks kergesti teha lahendusele lisa arendusi.

Lõpptulem peab olema ka avalikult kättesaadav teistele arendajatele täiendamiseks, muutmiseks ja piiranguteta kasutamiseks. Lähtekood peaks olema lihtsasti loetav ja kasutatav, et innustada standardi kasutusele võtmist ka teiste ühistranspordi ettevõtete süsteemides.

Lahendus peab järgima standardis nimetatud nõudeid ning olema piisavalt hästi loodud, et ka vanemates kasutajate seadmetes toimuks piletipakkide allkirjastamine ja kuvamine võimalikult kiiresti ja akut säästvalt.

2 Standardi analüüs

Standard koosneb kolmest osast. Esimene osa kirjeldab ära REST arhitektuuri põhised veebiteenused kasutades selleks OpenAPI spetsifikatsiooni [1] . Teises osas on dokumenteeritud kõik piletipakkidega seotud tegevused ja struktuurid. Kolmandas osas on määratletud skeemid, mis aitavad paremini aru saada üldisest arhitektuurist ja selle toimimisest. Standard ise ei sea ette piiranguid arhitektuuriliselt ega tehnoloogiliselt, kuidas piletisüsteem peab olema loodud. Standard määrab ainult liidese, mille abil on võimalik loodud süsteemiga suhelda.

2.1 Veebiteenused

REST on arhitektuuristiil veebiteenuste loomiseks, mis võimaldab teistel teenustel manipuleerida ja lugeda ressursse kasutades selleks HTTP päringuid vastava meetodi abil [2] . Tänapäeval on sellisel arhitektuuril põhinevad veebiteenused üha levinumad nende lihtsuse ja kiiruse tõttu. Üldjuhul on kasutusel järgnevad HTTP meetodid:

- GET – andmete lugemiseks.
- POST – uute andmete lisamiseks.
- PUT – andmete täielikuks uuendamiseks.
- PATCH – andmete osaliseks uuendamiseks.
- DELETE – andmete kustutamiseks.

Kuigi antud arhitektuuristiil võimaldab kasutada ükskõik millist andmevahetusvormingut, siis harilikult kasutatakse andmete edastamiseks inimloetavat andmevahetusvormingut JSON. Lisaks on oluline teada, et REST päringud on olekuta ehk iga uue päringuga peab teenuse tarbija end uuesti isikustama kasutades selleks JWT tõendit, sessiooni identifikaatorit või muid alternatiivseid lahendusi.

Andmeid sellisest teenusest on võimalik kergesti tarbida nii veebirakendustest, mobiilirakendustest, nutistust ehk asjade internetist või ka teistest veebiteenustest. Seetõttu põhinevad ka standardi veebiteenuste spetsifikatsioonid just sellel arhitektuuri viisil.

OpenAPI on spetsifikatsioon, mis on loodud veebiteenuste rakendusliideste määratlemiseks. Kuna REST veebiteenused on tihti peale avalikud, siis on spetsifikatsiooni loomine teenuse tarbijatele oluline, et paremini aru saada teenuse pakutavatest funktsionaalsustest. Lisaks on võimalik antud spetsifikatsiooni kasutada inimloetava dokumentatsiooni koostamiseks või lähtekoodi automaatseks genereerimiseks. Näiteks on võimalik OpenAPI spetsifikatsiooni failist tekitada Java andmemudelid kasutades selleks vastavat Gradle ehitustööriista pistikprogrammi [3] . Kokku on standardis 10 veebiteenust.

- Autentimise veebiteenus – kasutaja identiteedi kinnitamiseks ja vastava tõendi loomiseks.
- Broneerimise veebiteenus – vajaduspõhise transpordi tagamiseks.
- Seadme veebiteenus – kasutaja seadmele võtme loomiseks.
- Inspektsiooni veebiteenus – reisijate sõiduõiguse kontrollimiseks.
- Osalejate andmete veebiteenus – standardit jälgivate piletisüsteemi pakkujate andmete hankimiseks.
- Toote veebiteenus – reisijatele mõeldud toote andmete kuvamiseks.
- Pileti veebiteenus – piletitega seotud tegevuste kasutamiseks.
- Tõendi veebiteenus – väljastatud tõendite informatsiooni hankimiseks või tõendite kasutusõiguse eemaldamiseks.
- Reisija veebiteenus – reisijatega seonduvate andmete hankimiseks või uuendamiseks.
- Valideerimise veebiteenus – reisijatele ostetud piletite valideerimiseks.

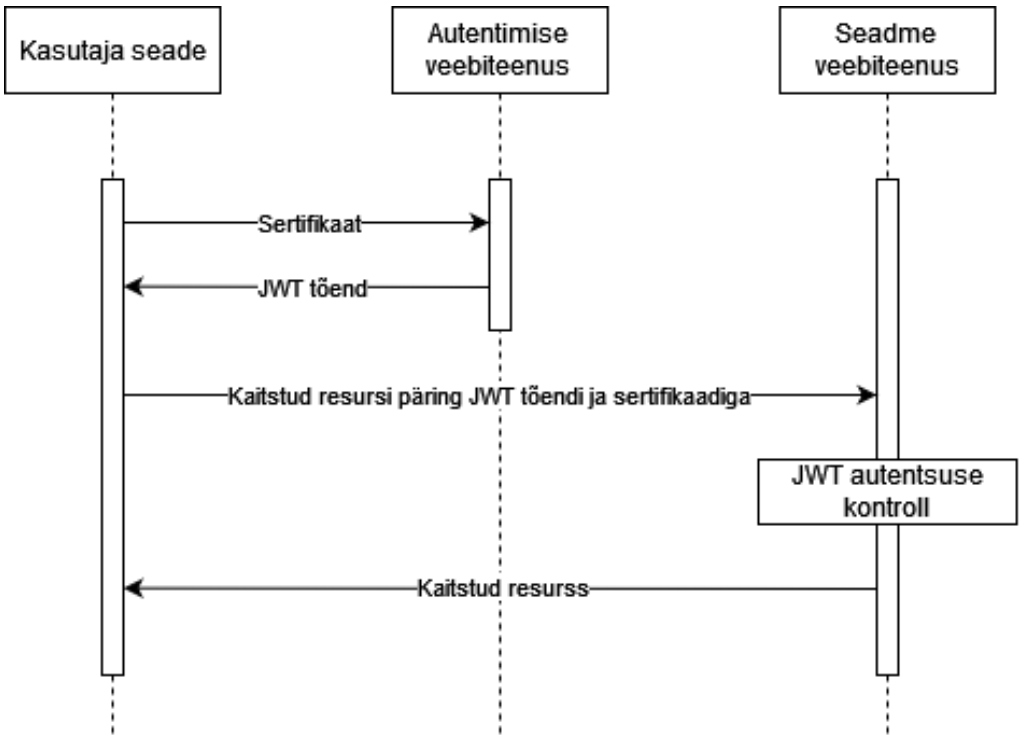
Täielikult ühilduva süsteemi jaoks on oluline luua kõik standardis kirjeldatud REST veebiteenused OpenAPI spetsifikatsioonide järgi, kuid kuna teenuseid on palju ja alati pole igal ühistranspordisüsteemil esialgseks ühilduse loomiseks kõiki vaja, siis on neid võimalik lisada hiljem vajaduspõhiselt. Selle lõputöö teegi jaoks on oluline kahe veebiteenuse olemasolu – seadme ja autentimise veebiteenus.

2.2 Autentimise veebiteenus

Autentimise veebiteenuse eesmärk on luua JWT tõendeid, mis tagavad teiste veebiteenuste tarbimiseks usaldusväärse viisi saada informatsiooni kasutaja identiteedist [4] . Usaldusväärsuse tagab tõendile allkiri, mille abil saab kontrollida, et tõendit pole proovitud kolmandate osapoolte poolt muuta ja tõend on loodud ainult süsteemi poolt tunnustatud teenuse poolt. JWT tõendeid on olemas kahte tüüpi. JWS on rohkem levinud JWT tõendiliik, kus andmed pole krüpteeritud ja on loetavad kõigile, kes tõendile ligi pääsevad. JWE on vähem levinud JWT tõendiliik, kus andmed on krüpteeritud ning ainult loetavad vastava võtme omanikele. Selles veebiteenuses on kasutusel rohkem levinud JWS tüüpi JWT tõendid.

Kui tavaliselt kasutatakse kasutaja tuvastamiseks emaili ja parooli, siis selle veebiteenuse puhul on tuvastamiseks kasutusel hoopis sertifikaat. Sertifikaat on digitaalne dokument, mis sisaldab lisaks informatsiooni osapoole usalduse loomiseks ka teavet selle kehtivuse kohta. Sertifikaat sisaldab tavaliselt avalikku võtit ja võtme omaniku andmeid. Omaniku andmete hulka kuuluvad tavaliselt isiku või organisatsiooni nimi, e-post ja aadress. Sertifikaadi loomine pole osa standardist ning seega vastutab iga süsteem ise selle loomise eest. Tavaliselt on süsteemikasutaja seotud ühe sertifikaadiga. Võimalik on kasutada ka seadmepõhist sertifikaati ehk ühel kasutajal võib olla mitu sertifikaati olenevalt tema seadmete hulgast.

Seadmepõhine sertifikaat tagab suurema turvalisuse, sest sertifikaati hoiustatakse alati ainult tarbija seadme mälus. Tagajärjena on aga seadme kadumise või hävinemise puhul võimatu taastada sertifikaati kui pole tehtud tagavara koopiat ning seega on ostetud piletid kasutajale ligipääsetamatud. Täpsema kasutaja tuvastamise loogika kasutades sertifikaati leiab jooniselt 1.

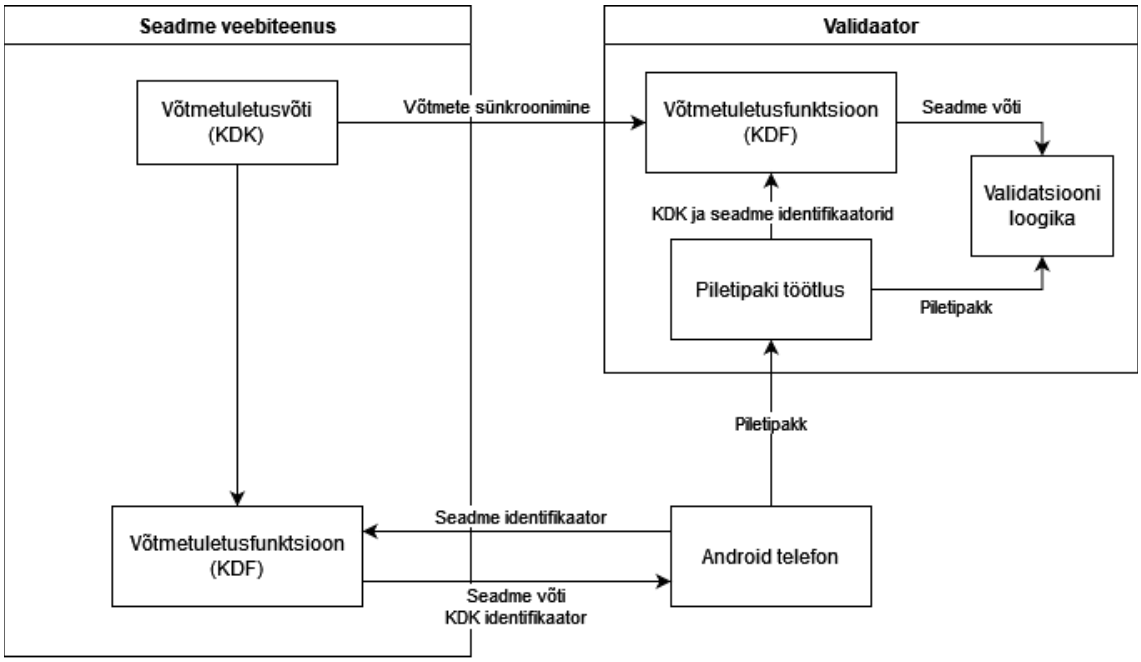


Joonis 1. Autentimise loogika skeem

2.3 Seadme veebiteenus

Seadme veebiteenuse eesmärk on luua võtmeid seadmetele, mis soovivad piletipakke allkirjastada. Põhimõttelt on piletipakkide allkirjastamise protsess sarnane JWT allkirjastamise protsessile, kuid andmevahetusvorminguks on JSON-i asemel CBOR, mille tulemuseks on kiirem andmete pakkimine ja väiksem edastatav infohulk.

Seadme veebiteenus kasutab vastava seadme identifikaatorit ja võtmetuletusvõtit ehk KDK sisenditena võtmetuletusfunktsioonis [5] . Kuna protsess põhineb krüptograafilisel räsifunktsioonil, siis tulemuseks saadud võtit pole võimalik kasutada lihtsasti algsete sisendite tuletamiseks. Täpsema ülevaate saamiseks protsessist on võimalik tutvuda joonisega 2.



Joonis 2. Andmete liikumine seadme veebiteenuse, validaatori ja seadme vahel

Lisaks seadme võtmete loomisele peab see teenus võimaldama ka ühistranspordi vahendis olevatel validaatoritel pärida kasutusel olevaid võtmetuletusvõtmeid, et validaatorid saaksid veenduda seadme võtmete autentsuses ka internetiühenduseta. Selleks on vajalik luua täpselt sama võtmetuletusfunktsioon ka validaatori tarkvara jaoks. Kuna üldjuhul kasutavad validaatorid kohandatud Android platvormi, siis on võimalik kergesti võtta kasutusele sama võtmetuletusfunktsiooni lähtekood nii kasutaja seadmes kui ka validaatoris.

K veebiteenuse loodavad võtmed ei ole asümmetrilised, siis tähendab see seda, et sama võtit kasutatakse nii piletipaki allkirja loomiseks kui ka allkirja valideerimiseks. Kuigi seadme allkirja eesmärk on tagada ainult, et piletipakk on loodud kindlas seadmes, siis on ikkagi oluline ka selles protsessis vältida võtmete lekkimist kolmandate osapoolte haldusesse.

3 Piletipakk

Piletipakk on sarnane ümbrikule, mille sisse on võimalik panna üks või rohkem piletit. Piletipakk ei määra, millisel kujul peab olema kindel pilet, sest tihti peale on piletipakkides erinevate ühistranspordi asutuste piletid, mis on erineva struktuuriga [6] . Piletipaki eesmärk on pakendada piletid selliselt, et kõik seadmed mõistaksid, kuidas leida vajaliku ühistranspordi vedaja pileteid ja valideerida nende autentsust. Kasutajale tähendab piletipakk paremat mugavust, sest mitme erineva ühistranspordi pakkuja sõidukites saab sõiduõigust tõendada ühe piletipaki esitamisega.

3.1 Piletipaki struktuur

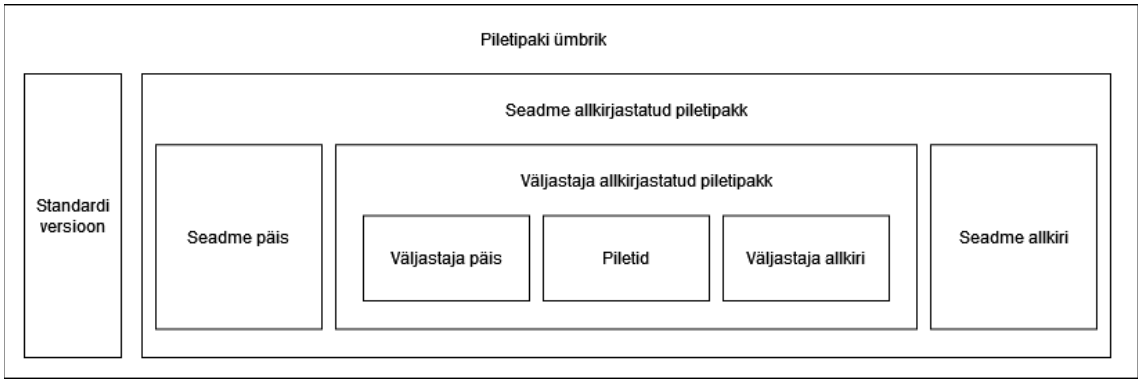
Piletipakk koosneb erinevatest kihtidest, kus iga kiht on kodeeritud CBOR vormingus. Kõige välimine kiht sisaldab standardi üldversiooni numbrit ja piletipaki andmeid. Üldversioon on selle lõputöö kirjutamise hetkel alati 1. Piletipaki andmed on vastavalt kontekstile, kas piletipaki väljastaja allkirjastatud piletipakk või konkreetse seadme allkirjastatud piletipakk. Mõlemad nii väljastaja allkirjastatud piletipakk kui ka seadme allkirjastatud piletipakid koosnevad kolmest osast. Kogu piletipaki struktuuri paremini mõistmiseks on joonisele 3 lisatud eelnimetatud osad ja nende paigutus piletipakis.

Päis sisaldab võti-väärtus paare, mis annavad taustainformatsiooni piletipaki enda kohta. Näiteks peab väljastaja päis alati sisaldama algoritmi, väljastaja identifikaatorit, võtme identifikaatorit ja standardi alamversiooni. Lisaks võib väljastaja päis sisaldada informatsiooni, millal piletipakk on loodud ja mis hetkest piletipakk enam ei kehti. Sarnaselt väljastaja päisele on ka seadme päises kohustuslikud ja valikulised võti-väärtus paarid. Seadme päis peab sisaldama algoritmi, seadme identifikaatorit ja võtme identifikaatorit ning võib sisaldada mobiilirakenduse identifikaatorit, piletipaki loomise aega ja kasutaja seadme asukohta.

Väljastaja algoritmi väärtus on alati kas ES256 või RS256 ning seadme algoritmi väärtus on alati HS256 või HS256-128. Kõik algoritmid põhinevad krüptograafilistel algoritmidel, kus räsifunktsiooniks on SHA256. Väljastaja algoritmid põhinevad asümmeetrilistel algoritmidel ECDSA ja RSA ning seadme algoritmid põhinevad sümmeetrilisel algoritmil HMAC, kus HS256-128 päise väärtuse puhul on tulemus lühendatud esimese 128 biti peale.

Teine piletipaki sisu osa on piletite enda informatsiooni jaoks. Piletid on jaotatud ühistranspordi vedaja identifikaatorite järgi, kuid iga ühistranspordi vedaja piletite sisu saab vastava vedaja süsteem ise määratleda. Keeruliseks teeb sisu tõlkimine see, et kui väljastaja allkirjastatud piletipakk sisaldab alati vastavaid pileteid, siis seadme allkirjastatud piletipakk sisaldab tegelikult väljastaja piletipakki koos päise ja allkirjaga. Seadme piletipakk seega on järjekordne kiht väljastaja piletipakile.

Piletipaki viimane ehk kolmas osa pole inimloetav ja on mõeldud piletipaki allkirja jaoks. Allkiri võimaldab veenduda, et piletipaki sisu ei ole muudetud ja piletipaki väljastus on tehtud usaldatud teenuse või seadme poolt.



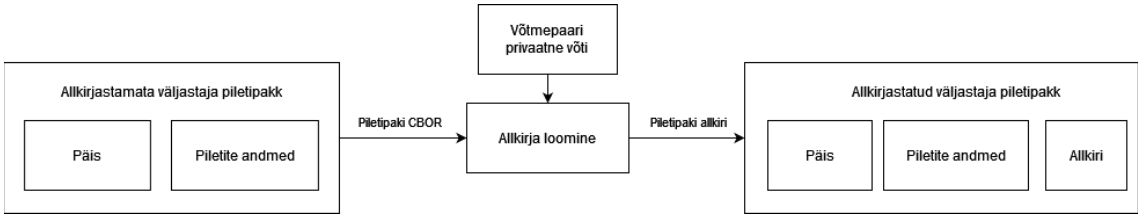
Joonis 3. Seadme allkirjastatud piletipaki struktuur

3.2 Allkirjastamine

Piletipakkide kõige olulisem osa on allkiri. See võimaldab veenduda, et piletipakk on edastatud ainult usaldatud osapoole poolt ning selle sisu pole proovitud edastamise hetkest alates muuta. Lõplikud piletipakid sisaldavad nii väljastaja allkirja kui ka seadme allkirja, mis tagab parema turvalisuse.

Allkirjastamise protsess ise algab esialgu piletivahendaja süsteemist. Kasutaja ostab piletivahendaja süsteemist enda reisi jaoks vajaliku piletipaki, mille tagajärjel ostetakse kõik vajalikud piletid reisi jaoks ühest või mitmest piletisüsteemist. Peale piletite ostmist luuakse esialgne piletipaki sisu, kus on piletipaki päis ja piletid ise. Esialgne piletipakk kodeeritakse CBOR vormingusse ja tulemuseks saadud andmed on sisendiks allkirjastamise funktsioonile. Väljastaja piletipaki allkirjastamiseks kasutatakse võtmepaari privaatset võtit. Võtmepaari eelis on see, et kontrolliks on vaja jagada ainult avaliku võtit ning privaatne võti jääb alati looja süsteemi. Funktsiooni väljund liidetakse piletipakile ehk saadakse piletipaki kolmas osa. Tulemuseks on väljastaja allkirjastatud piletipakk, mis edastatakse kliendi seadmele ümbrikus.

Allkirjastamise teine samm toimub konkreetses seadmes kasutades selleks kasutaja seadme identifikaatori alusel saadud jagatud võtit. Esmalt luuakse esialgne piletipakk kasutades selleks seadme päist ja ümbrikust saadud piletipakki. Saadud tulemus kodeeritakse järjekordselt CBOR vormingusse ja on sisendiks allkirjastamis funktsioonile. Allkirjastamise funktsiooni väljund liidetakse piletipakile ja lõplik piletipakk pannakse ümbrikusse nagu seda on näidatud joonisel 4.



Joonis 4. Allkirjastamise protsess

3.3 Edastamise võimalused

Standardis on kirjeldatud ainult üks edastamise viis kasutades selleks optilist esitust. Optilisel kujul piletipaki edastamiskes peab nii seade kui ka validaator kasutama Aztec ruutkoodi [7] . Aztec ruutkood on maatrikskoodi tüüp, mis on väga sarnane maailmas laialt levinud QR ruutkoodile, kuid erineb mõne võrra oma paigutuse ja võimekuste poolest. Nimelt on Aztec ruutkoodid võimelised mahutama suuri andmemahte tagades vea kindluse kasutades selleks korrektuuralgoritmi. Aztec koodi tunneb ära koodi keskel asuva kujutise järgi, mis meenutab asteekide püramiidi ülevalt alla vaadates.

Piletipakk peab enne ruutkoodiks muutmist olema pakendatud ja tihendatud zlib vormingus, et vähendada piletipaki suurust. Kuna zlib vorming on maailmas laialt levinud, siis on üldjuhul võimalik kasutada antud vormingut pea kõikides seadmetes. Aztec ruutkoodi kasutamisel on oluline kasutada vähemalt 23% väärtusega veakoodi parandust, et tagada võimalikult töökindel ruutkoodi lugemine. Lisaks on soovituslik, et ruutkoodi diagonaalne pikkus oleks minimaalselt 35 millimeetrit ja maksimaaselt 75 millimeetrit, et ruutkoodi kuvav seade ei peaks olema validaatorile liiga ligidal või liiga kaugel.

Lisaks optilisele piletipaki edastamisele on võimalik kasutada ka muid kontaktivabu andmeedastusvõimalusi nagu näiteks NFC või Bluetooth, kuid nende kasutus pole veel nii laialt levinud ning enamik validaatoreid ei oma veel võimekust riistvaraliselt andmeid vahendada eelnimetatud kontaktivabadel viisidel. NFC kasutamine pole küll standardis kirjeldatud, kuid on lõputöö kirjutamise hetkeks juba mõne standardit järgiva piletisüsteemi poolt edukalt kasutusele võetud.

4 Android teek

Analüüsist on selgunud, et peamised raskused standardi kasutusele võtmisega on seotud piletipakkidega ja nende allkirjastamisega. Kuna lõputöö autor kasutab igapäevaselt Android seadmeid ja on ka varem teinud Androidi seadmete jaoks arendusi, siis otsustas autor selle lõputöö raames arendada abistava teegi Androidi platvormile [8] , [9] . Lisaks on võimalik kasutada loodavat teeki inspiratsiooniks ka iOS platvormi teegi loomise jaoks.

4.1 Prototüüp piletipakkide lugemiseks

Enne teegi arenduse alustamist oli vaja autoril leida või luua rakendus, mis lihtsustaks piletipakkide arendamist. Selleks lõi lõputöö autor olemasolevasse veebiteenusesse uue meetodi, mis võimaldab piletipakke inim loetavaks muuta ning valideerida nende struktuuri korrektsust.

Prototüübi loomise käigus uuris lõputöö looja juba enda firma teiste arendajate tehtud validaatori tarkvara lähtekoodi. Kuna validaatorid loevad piletipakke optiliselt oli võimalik juba loodud lähtekoodi mõnevõrra kasutada alusena prototüübi loomiseks. Esmalt oli vaja eemaldada lähtekoodist ruutkoodi lugemine ja töötlus. Teiseks oli vaja kuvada piletipakkide struktuur lihtsasti aru saadaval viisil, selleks oli võimalik kasutada firma ühe teise teenuse lähtekoodi, mis võimaldas JWT-i andmeid kuvada tekstina. Kuna teenuses kasutusel olev JSON-i protsessor oli võimalik vahetada välja kerge vaevaga CBOR-i protsessori vastu, siis kiirendas see oluliselt prototüübi loomist. Tulemuseks sai loodud lõputöö raames veebiteenuse meetod, mis võimaldab piletipakki edastada kodeerimisskeemi Base64url tekstina ja saada vastu inimloetav piletipaki struktuur koos andmetega JSON vormingus. Peale prototüübi loomist oli autoril plaan eraldada lisatud meetod veebiteenusest käsureaprogrammiks, kuid kuna tegemist oli kiirelt loodud prototüübiga polnud mõistlik sellele aega kulutada selle lõputöö raames.

Prototüübi loomise käigus avatas autor, et erinevad objektid piletipaki sees on kõik kodeeritud eraldi CBOR vormingu jadadena. Kuna standardis pole piletipaki skeemil seda näidatud, siis tuli standardis kirjeldatud teksti mitmed korrad põhjalikult lugeda, et veenduda, kas tegevus on nõutud või mitte.

Loodud prototüüp oli suureks abiks teegi loomisel, sest see aitas leida tehtud tarkvaravigu ja kontrollida teegi abil loodud piletipakkide sisu. Lisaks sellele on võimalik tulevikus lisada prototüübile ka võimekus piletipakkide allkirjade korrektsuse kontrollimiseks.

4.2 Rakendusliidese disain

Rakendusliidese disainimiseks oli vaja panna paika täpsed sammud, mis on vaja teha väljastaja allkirjastatud piletipakiga, et saada kätte lõplik piletipakk ruutkoodi jaoks.

Kuna rakendusliidest oli autoril raske ette planeerida ja disainida ilma lähtekoodita otsustas ta kasutada koodi silumise protsessi, kus tükk haaval eraldas ta lähtekoodist erinevate nõuetega seonduvad read [10] . Tulemuseks oli paremini loetav, hallatavam ja laiendatavam teek, mida saab kasutada ka muude ülesannete täitmiseks.

Lõplik rakendusliides on ülesannete põhiselt tükeldatud neljaks osaks. Esimene osa tegeleb veebiteenuste suhtlusega, teine on mõeldud üldiste teenuste jaoks nagu näiteks seadme identifikaatori lugemine või piletipaki tihendamine, kolmas on andmemudelite definitsioonid ja neljas osa aitab piletipakkide edastamisega.

4.3 Teegi loomine

Androidi teegi loomisel on võimalik valida programmeerimiskeeleks Java või Kotlin. Mõlemad programmeerimiskeeled on ühilduvad JVM baitkoodiga. Java on vanem ja rohkelt kasutatav objektorienteeritud keel, mis oli esimene ametlikult tunnustatud programeerimiskeel Android tarkvara loomiseks. Kotlin on JetBrainsi loodud Java alternatiiv, mis on lõputöö loomise hetkel Androidi platvormil ametlikult tunnustatud ja soovitatud keel. Kuigi Kotlin on uuem ja oluliselt parema loetavusega programeerimiskeel, siis otsustas autor siiski Java kasuks, kuna puutub sellega igapäevaselt kokku tööl ning selle keele semantika on talle paremini selge. Lisaks

programmeerimiskeelele on võimalik valida ka kahe projekti ehitustööriista vahel – Maven või Gradle. Mõlemad tööriistad on laialt levinud just Java põhiste projektide ehitamiseks. Maven on vanem ja rohkem levinud ehitustööriist, mis kasutab konfiguratsiooniks XML faili. Gradle on uuem ja kiirem ehitustööriist, mis põhineb Groovy programmeerimisekeeles loodud konfiguratsioonil, kuid toetab lisaks ka Kotlin keelel loodud konfiguratsiooni. Autor otsustas selle lõputöö raames kasutada Gradlet, kuna selle tööriistaga puutub autor kokku igapäevaselt oma töös. Kuigi mõlema otsuse tegemisel on omad positiivsed ja negatiivsed küljed, siis teegi kasutaja jaoks ei ole kumbki otsus niivõrd oluline, sest teegi kasutamisele võtmise protsess on mõlemast otsusest sõltumata täpselt sama.

Seadme piletipaki loomiseks oli autoril esmalt vaja luua seadme päis, mis sisaldab lisaks identifikaatoritele ka informatsiooni seadmest endast. Kuna päis peab sisaldama kindlasti ka seadme identifikaatorit oli see esimene samm, millest alustada. Standardi MTS2 dokumendis on täpselt sõnastatud sammud, mida peab tegema seadme identifikaatorite saamiseks. Identifikaatori aluseks on Androidi operatsioonisüsteemi muutuja ANDROID_ID, mis on juhuslik 64-bitine number ning see luuakse seadme esimese korra avamisel. Kuigi esialgu võib tunduda, et seda numbrit on võimalik kasutada piletipakis identifikaatorina, siis on see turvalisuse tõttu mitte soovituslik. Standardis on soovitatud see muutuja esmalt räsida SHA256 räsifunktsiooniga ja vastavast tulemusest võtta esimesed 64-biti.

Kuna tulemuseks saadud identifikaator on räsitud ja räsitulemus lühendatud on peaaegu võimatu saada teada esialgne muutuja väärtus. Lisaks on tulemus piisavalt unikaalne, et identifikaator ei kattu pea kunagi ühe kasutaja jaoks mitme erineva seadme kasutamisel. Kuna Java standardteek ise ei sisalda räsifunktsioone otsustas lõputöö autor kasutada Google loodud Java teeki Guava, mis on kasutajasõbralik, laialt levinud ja hästi hooldatud teek. Jooniselt 5 on võimalik näha lähtekoodi lõiku, kuidas toimub räsifunktsiooni kasutamine selle teegi abil.

```
Hashing.sha256()
  .hashBytes(BaseEncoding.base16().decode(deviceId))
  .writeBytesTo(deviceIdBytes, 0, deviceIdBytes.length);
```

Joonis 5. Näide Google Guava teegi räsifunktsiooni kasutamisest

Lisaks kasutaja identifikaatorile on oluline ka töödelda esmalt seadmest saadud kuupäeva ja kellaaega standardis määratletud viisil. Selleks oli lõputöö loojal võimalik kasutada Java standardteegi poolt pakutud võimalusi, täpsemalt klasse Instant ja SimpleDateFormatter.

Järgmise sammuna oli vaja luua teeki võimekus piletipakke lugeda ja Java objektideks muuta. Selleks kasutas lõputöö autor juba prototüüpi tehes saadud teadmisi ning prototüübis kasutatavat teeki Jackson. Jackson on üks populaarsemaid Java teeke peamiselt mõeldud JSON-i ja Java objektide vaheliseks teisendamiseks, kuid toetab lisateekidega ka muid andmevahetusvorminguid nagu näiteks selle lõputöö raames vajalikku CBOR-i vormingut. Näite teegi kasutusest leiab jooniselt 6. Kuna piletipakk sisaldab rohkem kui ühe kihina CBOR vormingut oli algne plaan luua rekursiivne funktsioon piletipaki lugemiseks. Plaanitud viisiga oleks pidanud kasutama andmetüüpe, millega oleks kadunud ära tugev tüüpimine ning seetõttu leidis autor, et antud lahendus pole kõige parem. Seega oli vaja iga kiht käsitsi lugeda vastavaks Java objektiks.

```
Container container = cborMapper.readValue(containerBytes, Cont-
ainer.class);
```

Joonis 6. Näide Jackson CBOR teegi kasutamisest

Lahti pakitud väljastaja allkirjastatud piletipakist on võimalik lugeda sisse piletipaki sisu. Saadud sisu on vaja uue piletipaki loomiseks, sest kogu väljastaja loodud piletipakk on sisuks seadme allkirjastatud piletipakile. Seadme piletipaki loomiseks seega oli vaja autoril esmalt luua uus seadme informatsiooni sisaldav piletipaki päis ja lisada väljastaja piletipaki sisuks. Piletipakk kodeeritakse seejärel uuesti CBOR-i vormingusse ning tulemuseks saadud baidirida on sisendiks seadme allkirja loomiseks.

Allkirja loomiseks on vaja ka hankida võti. See tuleb pärida seadme veebiteenusest kasutades selleks juba päisesse lisatud seadme identifikaatorit ja autentimise veebiteenusest saadud JWT tõendit. Selleks lõi autor teeki abistava funktsiooni, mis kasutab Java teeki OkHttp, et vastav võti veebiteenusest kätte saada. Näide OkHttp teegi kasutamisest leiab jooniselt 7. Autor otsustas kasutada OkHttp teeki, sest selle kasutamine on oluliselt lihtsam ja loetavam kui Java standardteegist leitavad

rakendusliidesed. Lisaks oli autoril plaanis luua ka abistav funktsioon JWT tõendi saamiseks autentimise veebiteenusest, kuid kuna see osa võib olla erinevalt lahendatud ühistranspordi piletisüsteemides otsustas ta selle osa jätta teegi kasutaja vastutuseks.

```
try (Response response = client.newCall(request).execute()) {
  /* ... */
  String responseJson = response.body().string();
  /* ... */
  return jsonMapper.readValue(responseJson, DeviceKeyResponse.class);
}
```

Joonis 7. Näide OkHttp teegiga HTTP päringu tegemisest ja vastuse lugemisest

Kuna piletipakke edastatakse peamiselt Aztec ruutkoodina validaatoritele oli mõistlik lisada autoril ka abistav funktsioon teeki, mis aitaks luua Android Bitmap klassi soovitud piletipakile. Antud klassi on seejärel võimalik kasutada piletipaki visuaalseks kuvamiseks kasutajale. Vastavalt standardile peab esmalt piletipakki tihendama zlib vormingusse. Selleks leidis autor Java standardteegist vajalikud klassid. Täpsemalt sai kasutada klasse Deflater ja Inflater. Nende klasside kasutamiseks oli esmalt vaja luua puhver ning luua tsükkel, mis kirjutaks puhvrisse tihendatud baidid. Kuna enamik piletipakke on üldjuhul üsna väikesed, siis tavaliselt mahub kogu tihendatud piletipakk korraga suuremasse puhvrisse ära. Autor kasutas ühe kilobaidise suurusega puhvrit, mis tagab, et tihendamiseks ei kasutada liiga palju muutmälu.

Aztec ruutkood on kahe dimensionaalne maatriks, kus iga biti väärtus määrab ära, kas tegemist on visuaalses koodis musta või valge ruuduga. Kuna ruutkoodi genereerimine käsitsi on keeruline, siis leidis lõputöö autor, et on võimalik kasutada Google loodud Java teeki nimega Zxing. Näite teegi kasutamisest lähtekoodist leiab jooniselt 8. Antud teek toetab laialt levinud ribakoode ja ruutkoode seal hulgas QR ning Aztec ruutkoode. Kuna antud teek ei toeta Androidi spetsiifilisi klasse, siis väljundiks saadud biti maatriks oli vaja käsitsi ümber teisendada Androidi Bitmap klassiks.

```
import com.google.zxing.aztec.encoder.Encoder;
/* ... */
BitMatrix bitMatrix = Encoder.encode(data, ecc, 0).getMatrix();
```

Joonis 8. Näide ZXing teegi kasutamisest

4.4 Automaattestid

Loodud teegi kontrollimiseks lõi lõputöö autor esmalt ühiktestid, sest need võimaldavad kontrollida iga komponenti teegis iseseisvalt. Ühiktestide loomiseks oli võimalik kasutada Java teeke JUnit 4, AssertJ ja seadme veebiteenuse simuleerimiseks OkHttp MockWebServerit. Nende testide loomisega kinnitas autor, et teegi üksikud osad töötavad ootuspäraselt tagades sellega teistele teegi osadele kindla põhja. Ühikteste luues avastas ka autor teegist mitmeid vigu. Kasutades testipõhist arendus metoodikat toimus testide loomine paralleelselt ka teegi enda arendusega [11] , [12] . Ühiktestid on kasulikud lisaks korrektsuse kontrolliks arenduse hetkel ka tulevikus uute arenduste või eksisteerivate funktsionaalsuste muutmisel kinnituseks, et teeki ei lisata uusi vigu. Lõputöö autor seadis endale eesmärgiks katta ühiktestidega kõik teegi osad võimalikult suure katvusprotsendiga. Saavutatud katvusprotsendid leiab jooniselt 9.

Element	Class, %	Method, %
com	88% (8/9)	85% (42/49)
ridango	88% (8/9)	85% (42/49)
bob	88% (8/9)	85% (42/49)
mtb	88% (8/9)	85% (42/49)
api	100% (3/3)	100% (17/17)
core	100% (3/3)	100% (10/10)
model	100% (2/2)	75% (15/20)

Joonis 9. Ühiktestide katvusprotsent

Lisaks ühiktestidele kasutas autor ka instrumentatsiooni teste, mille abil oli võimalik kontrollida ruutkoodide korrektset loomist ja loodud koodide suurust. Instrumentatsiooni testid aitavad kontrollida lähtekoodi toimimist reaalsetes kasutustingimustes kuna testid käivitatakse mitte arendaja masinas, vaid Androidi emulaatoril või füüsilises seadmes. Nende testide loomiseks oli vajalik kasutada Java teeke Espresso ja JUnit 4. Eelnimetatud teekide seadistus oli juba autori jaoks ette tehtud uue projekti loomisel.

Autor seadistas ka koodihoidlas GitHub ülesse pideva integratsiooni kasutades selleks CI tööriista GitHub Actions. Selle tööriista abil kontrollitakse automaatselt igakord peale uue muudatuse lisamist koodihoidlasse, kas ühiktestid on edukad ega pole lisatud uusi vigu. Lisaks testimisele kontrollitakse lähtekoodi vormindust Gradle

pisitkprogrammiga Spotless ja ehitatakse valmis ka teegi arhiiv fail, mida on võimalik üles laadida avalikku paketihalduri registrisse.

4.5 Teegi kasutamine ja dokumenteerimine

Teegi kasutamiseks peab arendaja selle lisama enda projekti uue sõltuvusena. Kuna teegi üleslaadimiseks avalikku Maven Central hoidlasse on vaja kinnitust hoidla haldurilt, siis hetkel pole seda teeki veel võimalik kusagilt alla laadida. Õnneks on võimalik lisada sõltuvusi ka kohalikult masinast seega teegi kasutaja peab esmalt kopeerima kogu teegi projekti endale masinasse ja jooksutama Gradle ehitustööriista ülesannet, mis ehitab valmis Android teegi arhiivifaili ja lisab selle kohalikku Maveni hoidlasse. Näited ehitustööriista kasutamisest leiab jooniselt 10.

```
/* Linux või MacOS kasutajatele */
./gradlew publishToMavenLocal
/* Windows kasutajatele */
.\gradlew.bat publishToMavenLocal
```

Joonis 10. Näide teegi ehitamisest ja lisamisest kohalikku hoidlasse

Pärast teegi lisamist kohalikku hoidlasse on võimalik lisada teek ehitustööriistaga Gradle enda Android projekti sõltuvustesse ning kasutada kõiki teegi pakutud võimalusi. Jooniselt 11 on võimalik näha, kuidas lisada teek ehitustööriistaga Gradle projekti sõltuvuste hulka.

```
/* Linux või MacOS kasutajatele */
implementation("com.ridango.bob.mtb:mtb-helper:1.0.0")
```

Joonis 11. Näide teegi sõltuvuse lisamisest Gradle abil

Teegi dokumentatsiooni otsustas autor lisada lähtekoodi hoidlasse Markdown failina, mis on väga laialt levinud failitüüp seletusfailide loomiseks ning on toetatud enamikes tekstiredaktorites.

5 Edasiarenduse võimalused

Selle lõputöö raames valminud teek aitab arendajal kerge vaevaga piletipaki ruutkoode luua, kuid kuna piletipakke on võimalik lisaks optilisele viisile edastada ka NFC abil võiks olla see näiteks üks lihtne viis, kuidas teeki veelgi kasulikumaks teha. Lisaks on standardis lubatud seadme allkirju teha ka kärbitud algoritmi kujul, mida ma hetkel teeki ei lisanud, kuid täieliku standardi katvuse pärast oleks vajalik lisada.

Kindlasti on võimalik lisada ka rohkem ühikteste, mis kataksid ka vähe esinevaid olukordi ning tagaksid parema kindluse uute arenduste lisamiseks. Ideaalis võiks olla ka koodi kaetavus 100% ühiktestidega, sest sellisel juhul on alati kõik tingimuslaused kaetud.

Lisaks Androidi teegile peaks sarnase teegi looma ka iOS platvormile, sest tihti peale on ühistranspordi mobiilirakendused loodud nii Androidi kui iOS platvormile. Sellest tulenevalt on võimalik ka seda teeki kasutada hübriidtehnoloogiates nagu näiteks React Native. Selleks tuleb luua vastavad lisad, mis võimaldavad kasutada Java/Kotlini või Swifti funktsioone teistest keeltest nagu näiteks JavaScript.

Hetkel on kogu teegi dokumentatsioon lisatud lähtekoodi seletusfaili, kuid soovituslik oleks kasutada Javadoc dokumentatsioonigeneraatorit, mis tagab parema IDE koodilõpetuse ja võimaluse luua staatilised HTML veebilehed, mida lihtsasti veebilehitsejas kuvada.

6 Kokkuvõte

Lõputööga valmis avalikult kasutatav ja loetav Androidi teek, mis lihtsustab arendajatel võtta kasutusele Rootsi ühistranspordi standard BoB nii uutes kui olemas olevates ühistranspordi piletisüsteemides. Teek aitab piletipakkide allkirjastamise ja ruutkoodiks muutmisega, kuid on arendatud modulaarselt ning võimaldab arendajal kasutada teegist ainult vajalikke funktsioone nagu näiteks seadme identifikaatori hankimine või andmejada tihendamiseks zlib andmevormingusse. Lisaks arendaja töö lihtsustamisele pakub ka teegi kasutamine kindlust, et teegi pakutud funktsioonide kogu jälgib rangelt standardis määratud nõudeid.

Valminud teek täidab lõputöö alguses välja toodud eesmärke ning loodetavasti innustab ka teisi ühistranspordi süsteemide arendusega tegelevaid ettevõtteid uurima ja võtma kasutusele antud lõputöös käsitlevat ühistranspordi standardit. Kuigi lõputöö autori tööna valminud teek on kasutamiseks valmis, siis on plaan tulevikus jätkata selle teegi arendust, et lisada uusi võimekusi.

Plaanis on ka luua sarnane teek iOS platvormile, et võimaldada piletipakkide kuvamine ka Apple nutitelefonide omanikele.

7 Kasutatud kirjandus

[1] J. Ponelat and L. Rosenstock, Designing APIs with Swagger and OpenAPI. New York: Manning Publications Co. LLC, 2022.

[2] J. J. Geewax, API Design Patterns, 1st ed. New York: Manning Publications Co. LLC, 2021.

[3] Ikkink, Gradle GmbH, and GmbH, Gradle Effective Implementation Guide. Birmingham: Packt Publishing, 2012.

[4] S. Ahmed and Q. Mahmood, "An authentication based scheme for applications using JSON web token," in 2019 22nd International Multitopic Conference (INMIC), 2019, pp. 1–6, doi: 10.1109/INMIC48123.2019.9022766.

[5] F. Ljunggren, "Mobile Ticket Specifications – Device Key Derivation" Document ID: MTS2 version 1.0.2, Apr. 24, 2019. Available: https://bitbucket.org/samtrafiken/bob-mts-2-device-key-derivation/src/master/mts2.pdf

[6] F. Ljunggren, "Mobile Ticket Specifications – Mobile Ticket Format" Document ID: MTS1 version 1.7.1, Nov. 24, 2023. Available: https://bitbucket.org/samtrafiken/bob-mts-1-mobile-ticket-format/src/master/mts1.pdf

[7] Gaoyan Zhang and Haifeng Ke, "An algorithm of distortion correction for Aztec code," in 2010 2nd IEEE International Conference on Information Management and Engineering, 2010, pp. 173–176, doi: 10.1109/ICIME.2010.5477754.

[8] G. Allen, Android for Absolute Beginners, 1st ed. Berkeley, CA: Apress L. P, 2021.

[9] H. Deitel and P. Deitel, Android, 2nd ed. NOIDA: Pearson Education, 2015.

[10] Eric Freeman and Elisabeth Robson, Head First Design Patterns, 2nd Edition. O'Reilly Media, Inc, 2020.

[11] V. Farcic and A. Garcia, Test-Driven Java Development, Second Edition, 2nd edition. Birmingham: Packt Publishing, Limited, 2018.

[12] L. Koskela, Test Driven: Practical TDD and Acceptance TDD for Java Developers. Greenwich, CT: Manning Publications, 2007.

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

Mina, Taniel Tari

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Android teegi loomine Rootsi ühistranspordi standardi jaoks" mille juhendaja on Jaanus Pöial.

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.

06.12.2023

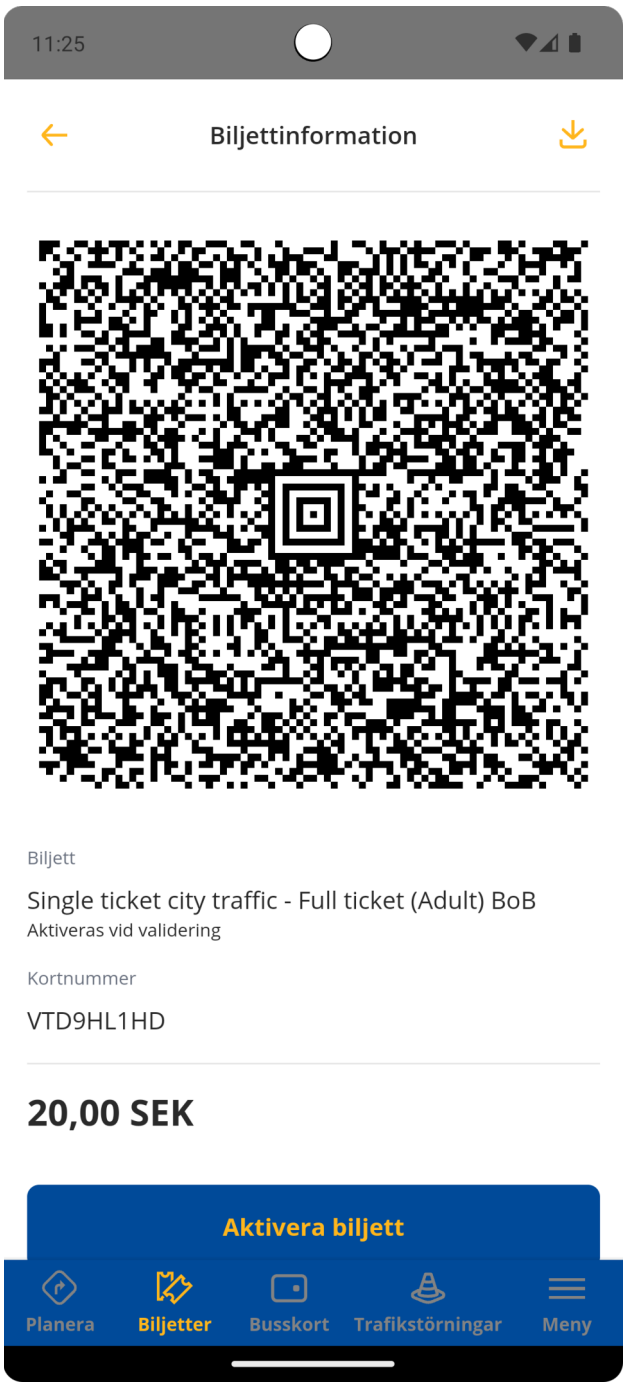
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või 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 – Prototüüp veebiteenus piletipaki lugemiseks

```
(CBOR Object) {
  "v": (CBOR String) "1",
  "p": (CBOR Array) [
    (CBOR Object) {
      "alg": (String) "HS256",
      "did": (String) "IuiwIENM5iA",
      "kid": (String) "bob_kdk_20230821",
      "t": (String) "20231021T111909.76Z"
    },
    (CBOR Array) [
      (CBOR Object) {
        "alg": (String) "ES256",
        "kid": (String) "60:202302_bob_mtb",
        "iid": (String) "60",
        "miv": (String) "4",
        "exp": (String) "20230827T061126Z"
      },
      (CBOR Object) {
        "60": (Array) [
          (Object) {
            "c_vn": (String) "5",
            "c_vm": (String) "1",
            "c_vt": (String) "1",
            "c_md": (Array) [
              (Object) {
                "tid": (String) "3013978180"
              },
              (Object) {
                "p_cid": (String) "VTCM0N7RV"
              }
            ],
            "c_tc": (String) ""
          }
        ]
      },
      (Binary) "fe5ea59a3a71209d27cb0647225594b6531dd3785b1748fe-
b0784c7b1090d3f9e-
f69e9397914ba516e03f6495d0dc19e65c96d0d8520dfd11d6a660bddd76fa7"
    ],
    (Binary) "30f7a6b48205bfaf4e7ce-
d914d4cf08bf6f9a5b19b864ce8a3cccf65c4e1defa"
  ]
}
```

Joonis 12. Näide prototüübi veebiteenuse vastusest

Lisa 3 – Teegi abil allkirjastatud piletipakk ruutkoodina



Lisa 4 – Lähtekood

Tehtud teegi lähtekoodi leiab https://github.com/tanieltari/bob-mtb