OULUN AMMATTIKORKEAKOULU

Lauri Miettinen

ANSAINTAMALLIT ANDROID-SOVELLUKSISSA JA KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA

ANSAINTAMALLIT ANDROID-SOVELLUKSISSA JA KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA

Lauri Miettinen
Opinnäytetyö
Syksy 2017
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehityksen suuntautumisvaihtoehto
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Lauri Miettinen
Opinnäytetyön nimi: Ansaintamallit Android-sovelluksissa ja klassisen auton elinkaaren seuranta älykkäällä sopimuksella
Työn ohjaaja: Veijo Väisänen
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 67

Oulun ammattikorkeakoulun tietotekniikan tutkinto-ohjelmassa on ollut mahdollista suorittaa opinnäytetyö 1–3 osassa. Tämä opinnäytetyö on suoritettu kahdessa osassa. Ensimmäinen osa on laajuudeltaan 5 opintopistettä. Toinen osa on laajuudeltaan 10 opintopistettä. Ensimmäinen osa valmistui vuoden 2016 keväällä ja toinen vuoden 2017 syksyllä.

Työn ensimmäinen osa oli selvitys Android-sovellusten ansaintamalleista. Työssä tehtiin kirjallinen selvitys eri ansaintamalleista, ansaintamallien toteuttamisesta Googlen palveluiden ja ohjelmointirajapintojen avulla, sekä selvitys Android-sovellusten markkinoista. Työn tekeminen oli yksinkertaista ja hyvin vaivatonta. Työstä saatu hyöty itselle, sekä tekniikan alalle yleensä, tuntui jäävän vähäiseksi.

Työn toisen osan tilaaja oli Hilla-ohjelma. Työssä toteutettiin sovellus, joka hyödyntää Ethereum-alustaa. Kirjallisessa selvityksessä kerrottiin työssä käytetystä tekniikasta ja tehtiin työn käytännön toteutuksesta kirjallinen selostus. Käytännön työssä onnistuttiin toteuttamaan suurin osa suunnitelluista käyttötapauksista. Työ oli haastava toteuttaa. Se vaati paljon vieraiden ohjelmistojen oppimista. Toinen osa antoi kokemusta työskentelytavoista, joita ohjelmistokehittäjältä vaaditaan työelämässä.

Asiasanat: ohjelmistokehitys, koosteopinnäyte, Android, ansaintamallit, lohkoketju, BitCoin, Ethereum

ABSTRACT

Oulu University of Applied Sciences
Degree programme in information technology, software development

---

Author: Lauri Miettinen
Title of thesis: Revenue models in Android-applications and developing a smart contract for tracking the life cycle of a classic car
Supervisor: Veijo Väisänen
Term and year when the thesis was submitted: fall 2017 Pages: 67

Oulu University of Applied Sciences has experimented with a model that lets students make their thesis in multiple smaller parts. The experiment has been going on since 2014. This thesis has been made in two parts. The first part is the equivalent of 5 credits of work. The second part is the equivalent of 10 credits of work. The first part was completed in spring 2016, the second part in fall 2017.

The first part of this thesis is a report on revenue models that can be implemented into Android-application. A written report was made of the different types of revenue models, how to implement them using Google's services and a report on the Android application market. This subject turned out to be quite simple. Knowledge of Google's monetization services did not end up feeling like truly useful knowledge for a graduating software engineer.

The second part of the thesis was to develop an application and write a report on its implementation for a Finnish research project called the Hilla-program. The application was made using the Ethereum-platform. Its goal was to demonstrate in a concrete way how the block chain technology could be used in the future. In the software application most of the planned use cases were successfully implemented. The project was challenging and rewarding. The task required thorough study and understanding of multiple libraries, platforms and frameworks.

Keywords: software development, Android, revenue models, block chain, BitCoin, Ethereum

ALKULAUSE

Kiitän opettajiani – Oulun ammattikorkeakoulussa, sekä muualla – kärsivällisestä avusta ja ohjauksesta. Toivon olevani hyödyksi muille saamallani osaamisella ja ymmärryksellä.

Kiitän Hilla-ohjelmaa mielenkiintoisesta aiheesta. Työnne on tärkeä Suomen ja koko maailman kannalta.

Thanks to Eyal Ron for sharing his inspiring outlook on the block chain technology and the Ethereum-platform. May the block chain cult grow ever greater.

1.12.2017

Lauri Miettinen

SISÄLLYS

1 JOHDANTO

Opinnäytetyö toteutettiin koosteopinnäytetyönä. Koosteopinnäytetyö on uusi malli opinnäytetyön tekemiselle Oulun ammattikorkeakoulussa. Koosteopinnäytetyökokeilu aloitettiin tietotekniikan tutkinto-ohjelmassa vuonna 2014.

Koosteopinnäytetyö oli mahdollista tehdä kolmessa tai kahdessa osassa. Tämän työn ensimmäinen osa tehtiin keväällä 2014 ja toinen osa aloitettiin keväällä 2015.

Työn ensimmäinen osa oli omatoiminen selvitys Android-sovellusten kehittämisestä. Työllä ei ollut tilaajaa. Tehtiin selvitys siitä, miten Android-sovelluksia voisi kaupallistaa. Tutustuttiin myös sovellusten kaupallistamisen toteuttamiseen Googlen tarjoamien palveluiden ja ohjelmointirajapintojen avulla. Ensimmäinen osa vastasi laajuudeltaan 5 opintopistettä.

Työn toisen osan tilaaja oli Hilla-ohjelma. Työssä toteutettiin sovellus, joka hyödyntää Ethereum-alustaa. Kirjallisessa selvityksessä kerrottiin työssä käytetystä tekniikasta ja tehtiin työn käytännön toteutuksesta kirjallinen selvitys. Toinen osa vastasi laajuudeltaan 10 opintopistettä.

2 OPINNÄYTETYÖN ENSIMMÄISEN OSAN ESITTELY

Opinnäytetyön ensimmäisessä osassa (liite 1) aiheena oli tutustua Android-sovellusten ansaintamalleihin. Ensimmäinen osa aloitettiin vuonna 2015, opiskelijan toisena opiskeluvuotena.

Aihe valittiin omasta kiinnostuksesta Android-käyttöjärjestelmän sovelluskehitykseen, sekä kiinnostuksesta liiketalouteen ja yrittäjyyteen. Työssä ei tehty käytännön ohjelmistoprojektia, vaan tutustuttiin eri rajapintojen ja kehitysympäristöjen dokumentaatioon.

Työssä tehtiin myös selvitys Android-sovelluksien kaupasta ja mitä niiden myynti on maailmanlaajuisesti. Työtä toteuttaessa, kirjallisuutta etsiessä selvisi pian, kuinka vaivatonta ansaintamallien toteuttaminen on Google-palveluiden avulla. Googlen laatima dokumentaatio on hyvin selkeää, helppolukuista ja kattavaa.

Työn toteutuksen loppupuolella saatiin selville, kuinka mobiilisovelluksilla on vaikea tehdä liiketoimintaa. Täten työn hyödyllisyys, sekä omalle ammatilliselle kehittymiselle että tekniikan alalle yleensä, jäi vähäiseksi. Silti työ antoi uusia näkökulmia ohjelmistotekniikan alalta.

3 OPINNÄYTETYÖN TOISEN OSAN ESITTELY

Vuonna 2016 työn tekijä osallistui Hilla-ohjelman järjestämään Blockchain Hackathon -kilpailuun. Tapahtumassa oli tavoitteena suunnitella ja toteuttaa viikonlopun aikana sovellus, joka hyödyntää lohkoketjutekniikkaa. Työhön sisältyi käyttötapauksien ideoiminen sekä yksinkertaisen sovelluksen tekeminen ja esitteleminen. Työ toteutettiin 3–4 ihmisen ryhmissä.

Ryhmäni toteutus keskittyi siihen, miten lohkoketjutekniikkaa voisi hyödyntää liikennöinnin alla. Ryhmä pääsi kilpailussa jaetulle toiselle sijalle. Hilla-ohjelma kutsui ryhmän hankkeeseen mukaan luomaan sovellusta, joka voisi konkreettisella tavalla esitellä, miten lohkoketjutekniikkaa voisi hyödyntää tulevaisuudessa.

Hilla-ohjelma ehdotti sovelluksen aiheeksi, kuinka älykkäitä sopimuksia voitaisiin hyödyntää klassisen auton elinkaaren seurantaan. Johtuen Blockchain Hackathon -ryhmäni jäsenten elämäntilanteista emme voineet tehdä työtä yhdessä ryhmänä. Halusin osallistua Hilla-ohjelman kutsuun tekemällä aiheesta opinnäytetyön, johon kuuluu käytännön työosa sekä kirjallinen osa.

Sovellusta esiteltiin Hilla-ohjelman ohjaajille aika ajoin palavereissa. Tilaaja antoi toivomuksia käyttötapauksista mitä he halusivat nähdä ohjelmistotyössä.

Ohjelmistotyön tavoitteena oli tehdä lopullinen versio esiteltäväksi Hilla-ohjelman konferenssa, joka pidettiin Oulun VTT:llä 4.4.2017. Tavoitteessa onnistuttiin. Työtä esiteltiin konferenssin puheohjelman jälkeen useille tekniikan alan ammattilaisille.

Ohjelmistotyön tekemiseen kului ajanseurannan mukaan 230 tuntia. Kymmenen opintopisteen laajuisen osaopinnäytetyön tavoite oli 270 tuntia. Ajanseuranta lopetettiin ohjelmistotyön valmistuttua, jonka jälkeen kirjallisen työn valmistumiseen kului kuukausia sekä arviolta monia kymmeniä työtunteja. Työn määrältään opinnäytetyön toinen osa olisi voinut olla jopa yksi 15 opintopisteen kokonaisuus.

Toinen osa oli hyvin opettavainen. Olen tyytyväinen siitä, kuinka vähän ohjausta tarvitsin työn aikana. Sain selvitettyä laajoja ohjelmistokokonaisuuksia itsenäisesti dokumentaatiota ja keskustelupalstoja lukemalla. Tilaajan kommentit ja toivomukset

käyttötapauksista antoivat työlle suuntaa. Toinen osa oli mielenkiintoinen kurkistus tulevaisuuden tekniikkaan, vaikka tekniikalla onkin varaa kehittyä. (Opinnäytetyön toinen osa on liitteessä 2.)

4 YHTEENVETO

Töiden aiheiden valinta heijastaa sen aikaista osaamistani ja haluani kehittyä ohjelmistokehittäjänä. Työn ensimmäisen osan aihetta valittaessa kiinnostuksen kohteeni ohjelmistokehityksen alalla olivat erilaisia kuin nykyään.

En ole tyytyväinen ensimmäisen osan aiheen valintaan, mutta olikin hyvä tilaisuus, että aihetta sai vielä vaihtaa tulevissa osissa. Oli tärkeä saada oppia Android-sovellusten markkinoista, ja kuinka vaikea sovelluksilla on tehdä liiketoimintaa. Työn tekniikan helppous antoi suuntaa tulevien vuosien opiskelulle, mihin tekniikoihin ja sovelluskehyksiin kannattaa panostaa oma opiskeluaikansa. Työn ensimmäisen osan jälkeen aloin opiskelussani panostamaan tekniikoihin, joiden oppiminen on vaativampaa. Työn toisessa osassa valitsinkin aiheen, jossa täytyi kehittää käytännön ohjelmistoprojekti.

On mahdollista, että Ethereum-alusta tekee tulevaisuudessa läpimurron maailmanlaajuiseen käyttöön. Silloin tässä työssä saatu osaaminen olisi hyvin hyödyllistä. Tekniikat, ohjelmointikielet ja sovelluskehykset voivat aina myös vanhentua tai korvaantua. Näin voi käydä Ethereum-alustalle, Solidity-kielelle tai Meteor-sovelluskehykselle. Vaikka niin kävisikin, työn toteutuksessa saatiin uutta kokemusta ohjelmistokehittäjän työskentelytekniikoista. Tekniikoiden ja kielien oppimista tärkeämpää on työskentelytapojen oppiminen.

LIITE 1

OULUN AMMATTIKORKEAKOULU

Lauri Miettinen

ANSAINTAMALLIT ANDROID-SOVELLUKSISSA

ANSAINTAMALLIT ANDROID-SOVELLUKSISSA

Lauri Miettinen
Opinnäytetyö, osa 1
26.2.2016
Tieto- ja viestintätekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Tässä työssä tutustutaan perusteisiin siitä, miten Android-sovelluksilla voidaan tehdä liiketoimintaa.

Maailman laajuisesti myydyistä älypuhelimista jopa 82 % pitävät sisällään Android-käyttöjärjestelmän (1). Kuluttajien suosion saavuttanut Android on myös saavuttanut sovelluskehittäjien suosion. Android-sovelluksia on helppo kehittää ja testata. Koska kehittäjien yhteisö on suuri, Android-ohjelmointiin on helppo saada apua, jos ohjelmistoyrityksellä tulee ongelmia vastaan. Nämä asiat yhdessä tekevät Androidista varteenotettavan vaihtoehdon alkavan yrityksen mobiilisovelluksen alustana.

Pelit omistavat hyvin suuren osan Android-sovelluksen markkinaosuudesta. Tämä työ, ja siinä käsitellyt esimerkit keskittyvät hyötysovelluksiin, vaikka työssä kuvattuja oppeja voidaan yhtä hyvin soveltaa myös pelisovelluksiin.

Työssä käydään läpi eri ansaintamalleja, vertaillaan niitä ja tutkitaan, minkälaisissa sovelluksissa niitä voidaan käyttää. Ansaintamalleja ja niiden hyviä ja huonoja puolia eritellään ja analysoidaan.

Työssä käsiteltävä ohjelmistotekniikka keskittyy Android-sovelluksiin. Tutkitaan, miten ansaintamallin sisältävä sovellus toteutetaan Android Studio-kehitysalustalla, ja miten toteutus tehdään Google Play -kaupan ohjelmistorajapinnoilla.

Jokaisesta työhön otettavasta ansaintamallista tehdään selvitys, minkälaista tekniikkaa ja mitä rajapintoja Google Play -kaupassa niiden toteutus vaatii. Tutkitaan, mitä työtä ohjelmistokehittäjän on tehtävä, jotta kukin ansaintamalli saadaan toteutettua.

Työssä tehdään selvitys mobiilisovellusmarkkinoista. Johtopäätöksessä tehdään yhteenveto mobiilisovellusliiketoiminnan kannattavuudesta.

2 ANSAINTAMALLIEN ERITTELY

2.1 Kertamaksusovellukset (Premium)

Yksinkertainen ansaintamalli on kertamaksusta ladattava sovellus. Android-kehittäjä tekee sovelluksen ja lähettää sen Google-kauppaan. Käyttäjä voi maksaa sovelluksesta kertamaksun, jolloin hän saa ladattua sen laitteellensa. Malli soveltuu hyvin sovelluksille, jotka ovat laajoja ja tarjoavat paljon ominaisuuksia. (2.)

Yksi kertamaksumallin huono puoli on se, että jokainen asiakas maksaa sovelluksesta vain kerran. Muut mallit, kuten sovelluksen sisäiset ostot, voivat pitkän ajan kuluessa saada aikaan enemmän tuloja asiakasta kohden. (2.)

2.2 Sovelluksen sisäiset ostokset (Freemium)

Sovelluksen voi jakaa ilmaiseksi, mutta siten, että käyttäjät voivat vaihtoehtoisesti maksaa rahaa sovelluksen sisäisissä ostoksissa. Ostokset voivat olla ominaisuuksia, jotka ovat lukittuja ilmaisessa versiossa. Maksamalla niistä voi ominaisuuden saada käyttöön. Sovelluksensisäiset ostokset voivat myös olla kulutettavia, esimerkiksi pelinsisäisen valuutan ostaminen.

Tämä malli voi olla yritykselle hyvin antoisa. Käyttäjät, jotka ostavat sovelluksensisäisiä ostoksia, ovat hyvin kiinnostuneita ja aktiivisia. Heidän hintaherkkyytensä sovelluksensisäisiin ostoksiin on pieni, eli he ovat valmiita kuluttamaan suuriakin määriä rahaa. Koska sovelluksen peruskäyttö onnistuu maksamatta, uusi käyttäjä voi tutustua sovellukseen kuluttamatta rahaa. Tällöin ostaja tuntee sovelluksen peruskäytön, ja oston jälkeen asiakastyytyväisyys on korkeampi. (3, s. 176.)

2.3 Tilaukset

Tilaus tarkoittaa sitä, että käyttäjä saa sovelluksen ominaisuuksia käyttöönsä, jos hän maksaa ajoittaisen tilausmaksun. Esimerkiksi uutissovelluksen tilaus voi maksaa kerran kuukaudessa.

Tämän mallin ansiosta yritykselle tulee jokaisesta tilaajasta jatkuva rahavirta, mikä voi olla hyvin tuottoisaa. Sovelluksen on oltava hyvin laadukas ja hyödyllinen, jotta tilaajat pysyisivät maksavina asiakkaina. (4.)

2.4 Mainokset

Sovelluksen lataus ja käyttö voidaan myös tehdä täysin ilmaiseksi. Tällöin sovellukseen voidaan lisätä mainoksia. Google on kehittänyt Admob-palvelun, jolla kehittäjä voi lisätä Android-sovelluksiinsa mainosbannereita. Jokaisesta kerrasta kun mainos ladataan käyttäjän Android-sovellukseen, se lasketaan yhdeksi katselukerraksi, eli impressioksi. Mainostajan maksamasta hinnasta voidaan käyttää CPM-lukua (Cost Per Mille, missä mille tarkoittaa kreikaksi tuhatta), mikä tarkoittaa mainonnan hintaa tuhannesta katselukerrasta. Mainostajat maksavat Admobille CPM-luvun mukaisesti, saadakseen mainoksensa näkyviin palveluun. Admob maksaa palkkion kehittäjille joiden sovellusten mainokset keräävät suuriä määriä katselukertoja. (4; 5, s 393; 7)

3 ANSAINTAMALLIEN TOTEUTUS

Google kehittää jatkuvasti uusia Android-käyttöjärjestelmän versioita. Jokaisessa käyttöjärjestelmäversiossa tulee uusia ominaisuuksia, joita Android-kehittäjä voi hyödyntää sovelluksia kehittäessä.

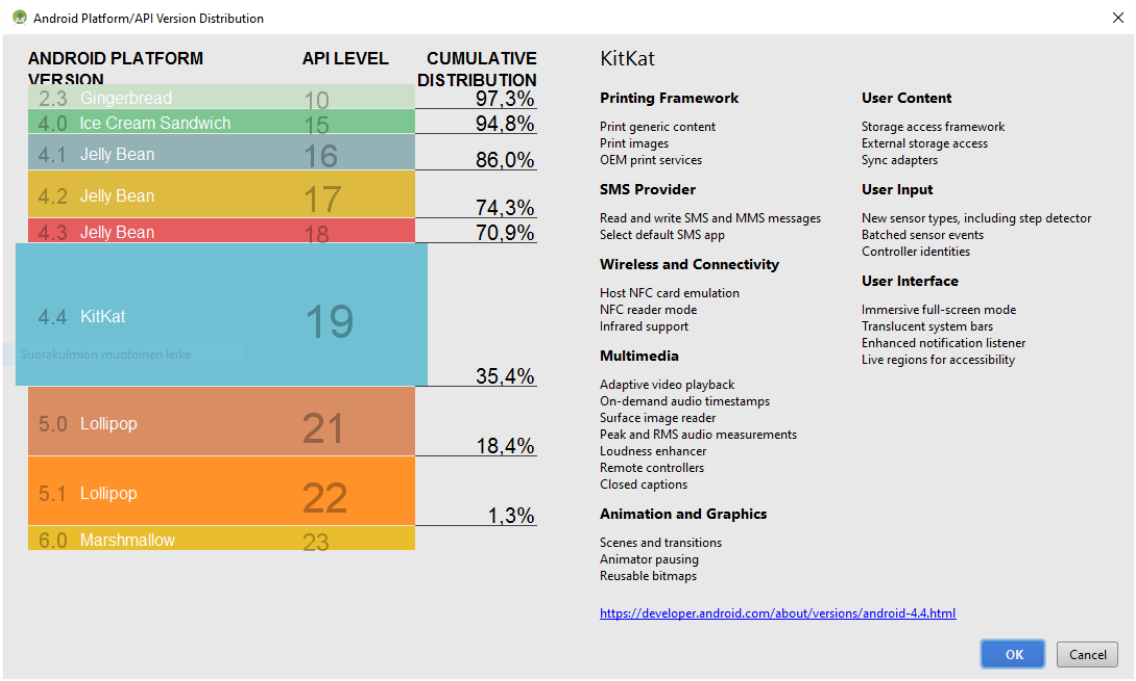
Jos liian vanhalla käyttöjärjestelmäversiolla yrittää avata sovelluksen, jonka kohde käyttöjärjestelmätaso on korkeampi, sovelluksesta saattaa puuttua ominaisuuksia. Liian vanhalla Android-käyttöjärjestelmällä sovellus voi kaatuilla jatkuvasti ja olla käyttökelvoton.

Ei myöskään ole kannattavaa suunnitella sovellusta yhteensopivaksi liian matalan ohjelmointirajapintatason kanssa. Uudemmissa tasoissa on paljon uusia funktioita ja luokkia, jotka voivat olla hyvin hyödyllisiä sovellusta kehittäessä.

Android Studio -projektia tehtäessä, kehittäjän täytyy valita sovelluksen minimi ohjelmointirajapintataso. Taso täytyy valita siten, että mahdollisimman moni sovelluskaupan käyttäjä pystyy käyttämään sovellusta, mutta kumminkin siten, että taso sisältää mahdollisimman uudet luokkakirjastot (kuva 1).

Kitkat 4.4 -käyttöjärjestelmä on yleisin Android-puhelimissa esiintyvä käyttöjärjestelmä. Tämä käyttöjärjestelmä tukee ohjelmointirajapinnan tasoa 19 ja on taaksepäin yhteensopiva kaikkien aikaisempien Android-versioiden kanssa. 70,9 % Google Play -kaupassa aktiivisista puhelimista tukevat tasoa 19. Kuvan 1 oikeassa laidassa kuvaillaan, mitä uusia ominaisuuksia ohjelmointirajapintatasoon 19 on tullut aikaisempiin tasoihin verrattuna.

Sovelluskehittäjän kannattaa tehdä sovelluksestaan yhteensopiva ohjelmointirajapinnan tason 19 kanssa. Jos kehittäjä määrittelee projektin Android-manifestissa alimman rajapintatason tasolle 19, Android Studio huomauttaa kehittäjää, jos hän vahingossa käyttää metodeja, jotka eivät ole yhteensopivia alimmalla tasolla. Myös Android-dokumentaatiossa kuvaillaan jokaisen metodin kohdalla, minkä tasoisen rajapinnan kanssa ne ovat yhteensopivia. (7.)



Kuva 1. Android Studio -ohjelmassa ohjelmointirajapintatason valintaa opastava kuvaaja.

3.1 GOOGLE PLAY-KAUPPA

Google Play-kaupan käyttöönotto vaatii Google Play julkaisijatilin, sekä kauppiastilin luomista. Google Play-kauppaan pystyy lähettämään sovelluksia kuka tahansa, mutta maksullisten sovellusten tekeminen vaatii kauppiastilin. (8.)

Google Play kauppiastili vaatii käyttöön ottaessa 25 $:n rekisteröitymismaksun. Maksu on hyvin edullinen. Microsoftin mobiilikehittäjämaksu vaatii 20 $ kertamaksun. Apple vaatii kalliin, 100 $ vuodessa maksun yksityishenkilöltä, joka haluaa julkaista iOS-sovelluksia. (9; 10)

Google tarjoaa Google Play -kehityskonsolin, jota voi käyttää internet-selaimella. Konsolin kautta kehittäjä voi julkaista Android Studiolla kehittämänsä sovellukset. Sovelluksen julkaisemisen jälkeen kehittäjäkonsoli tarjoaa tilastoja siitä, millä tavoin asiakkaat käyttävät sovellusta. Kehittäjä voi julkaista palautteen avulla päivityksiä sovellukseensa. Lisäksi tilastot voivat auttaa kehittäjää suunnittelemaan tulevia sovelluksiaan.

Kun sovellus on valmis, kehittäjä lataa sovelluspaketin (APK, Application Package) Google Play -kehittäjäkonsoliin. Kehittäjä lataa valmistellut kuvakaappaukset, videot sekä sovelluksen kuvaustekstit. Kauppasivun on vakuutettava asiakas sovelluksen nimellä, kuvaustekstillä, esittelyvideolla ja kuvakaappauksilla. Google Playssa kävijät voivat arvostella lataamiaan sovelluksia. Tyytyväiset käyttäjät jättävät positiivisia arvosteluja, houkuttaen yhä useammat käyttäjät lataamaan sovelluksen, mikä tarkoittaa kehittäjälle yhä enemmän maksavia asiakkaita.

Ennen julkaisua kehittäjä määrittelee, onko sovellus ilmainen vai maksullinen, ja määrittää sovelluksen hinnan. Sovelluksen julkaiseminen vaatii kehittäjältä perinpohjaista testaamista. (8.)

Google Play-kauppa ei suinkaan ole ainoa vaihtoehto sovelluksen julkaisemiselle. On myös mahdollista julkaista Android-sovellus omalla verkkosivuilla tai julkaista se jossakin toisessa internetissä olevassa sovelluskaupassa. Esimerkiksi jos ohjelmistoyritys haluaa julkaista sovelluksen ainoastaan ammattilaiskäyttöön, laajaan levitykseen suunniteltu Google Play ei ehkä ole ihanteellinen julkaisualusta. (11.)

3.2 SOVELLUKSENSISÄISET OSTOKSET

Kun sovelluksessa pyritään tekemään ostos, sovellus kommunikoi puhelimessa olevan Google Play -sovelluksen kanssa. Jos käyttäjä on tallentanut maksukeinon Google Play -sovellukseen, sovellus käsittelee ostopyynnön ja välittää tiedon ostoksen onnistumisesta takaisin sovellukselle jossa osto tehtiin. Tällöin käyttäjä näkee, kuinka ostettu tuote ilmestyy sovellukseen käytettäväksi. (12.)

Järjestelmässä on monia etuja. Yksi etu kehittäjän kannalta on, että Android-kehittäjän ei tarvitse lisätä sovellukseensa koodia, joka käsittelee maksutapahtumia. Kehittäjä ei tarvitse esimerkiksi maksutapahtumien tietoturvaan liittyvää kokemusta, sillä maksuliikenne tapahtuu aina Google Play -sovelluksen kautta. Käyttäjän kannalta ostamiskokemus on aina samanlainen jokaisessa Android-sovelluksessa. (12.)

Google Play tukee kahdenlaisia sovelluksensisäisiä ostoksia: hallinnoituja ostoksia (managed in-app products) sekä tilauksia (subscriptions). (13.)

3.2.1 Hallinnoidut sovelluksen sisäiset ostokset

Hallinnoidut ostokset tallentuvat Google Playlle ostohetkellä. Sovelluksen sisältä voidaan lähettää kyselyjä Google Playlle, mitä kaikkea käyttäjä on ostanut. Esimerkiksi sovellusta käynnistettäessä voidaan kysellä, onko käyttäjä ostanut premium-version sovelluksesta. Jos on, sovelluksen käyttöliittymää voidaan muuttaa premium-sovelluksen käyttäjälle asianmukaisesti.

Google Playlle voi myös lähettää kyselyjä tuotteen kuluttamisesta. Sovelluskehittäjä voi haluta tehdä kulutettavia tuotteita esimerkiksi videosovelluksessa, jossa käyttäjät voivat ostaa videoiden tai elokuvien katselukertoja.

Aina kun käyttäjä ostaa tuotteen, Google Play asettaa ostetun tuotteen omistettu-tilaan. Omistettu-tilassa olevaa tuotetta ei voi ostaa uudelleen. Google Playlle voi lähettää kyselyn tuotteen kuluttamisesta, jonka jälkeen tuote asetetaan ei-omistettu-tilaan, onka jälkeen tuotteen voi ostaa uudestaan.

3.2.2 Ostotapahtuman eteneminen

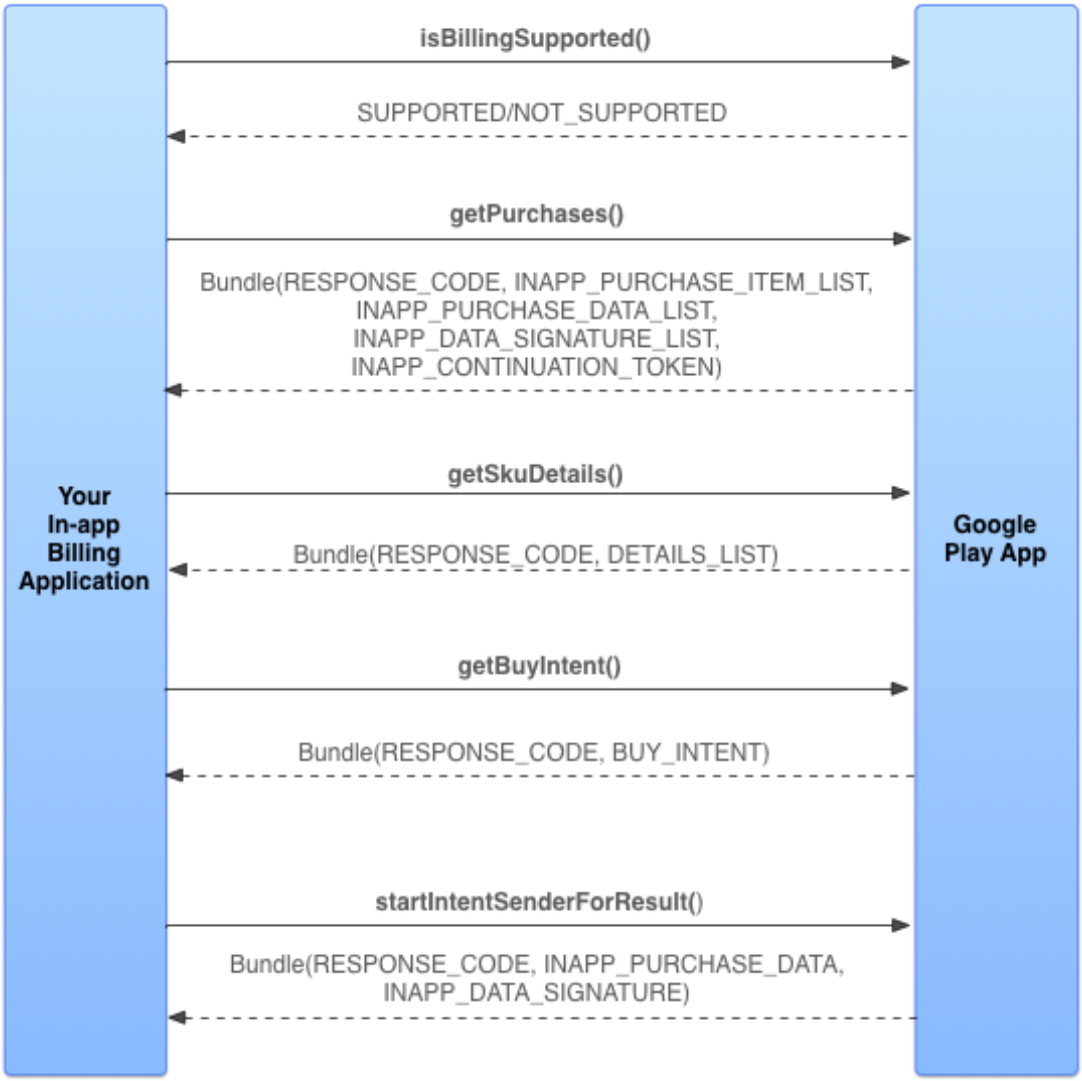
Ostotapahtuman tietoliikenne kuvaillaan kuvassa 2. Kehittäjän tekemän sovellus on kuvan vasemmalla puolella, oikealla puolella on Google Play -sovellus. Aika alkaa ylhäältä ja etenee alaspäin. isBillingSupported-metodilla kysytään, onko sovelluksen käyttämä sovelluksen sisäisten ostosten rajapinta Google Playn tukema.

getPurchases-metodilla kysytään mitä tuotteita käyttäjä omistaa. Vastaus palauttaa paketin jossa on listattu käyttäjän omistamat tuotteet.

getSkuDetails-metodilla haetaan lista kaikista tuotteista jotka on mahdollista ostaa. ”Sku” tarkoittaa Stock Keeping Unit, eli varastointinimike, mikä tarkoittaa tuotteen ainutlaatuista tunnistetta. Kehittäjä voi määritellä tuotteet kuvauksineen ja hintoineen Google Play -kehittäjäkonsolilla.

getBuyIntent-metodille annetaan ostettavan tuotteen tunniste ID, jolla itse ostotapahtuma aloitetaan. Google Play palauttaa PendingIntent-tyyppisen olion, joka luo sovelluksen käyttöliittymässä Google Play-kaupan ponnahdusikkunan, jossa osto voidaan hyväksyä.

Kun ostotapahtuma on päättynyt ja Google Playn ostoaktiviteetti päättyy, Google Play lähettää Intent-tyyppisen olion aktiviteetin onActivityResult-metodiin. onActivity-Result-metodissa on resultCode-kokonaislukumuuttuja, jonka arvo tiedottaa ostota-pahtuman onnistumisesta. Kehittäjä voi lukea tiedot jotka ovat onActivityResult-metodissa, ja tehdä sovelluksessa muutokset joilla käyttäjä saa ostamansa tuotteen käyttöön.



Kuva 2. Ostotapahtuman tietoliikenne. (13.)

3.2.3 Tilaukset

Samaan tapaan kuin sovelluksen sisäiset tuotteet, tilaukset määritellään Google Play kehittäjäkonsolissa. Kehittäjä voi määritellä tilauksen hinnan, sekä tilauksen rahastuksen tiheyden. (9.)

Ostotapahtuman eteneminen toimii samaan tapaan tilausta tehtäessä kuin hallinnoituja sovelluksen sisäisiä ostoksia tehtäessä. Ainoa ero on, että tilausta tehtäessä koodissa on annettava getBuyIntent-metodille tuotteen tyyppi -parametriksi merkkijono ”subs” (13):

```
Bundle bundle = mService.getBuyIntent(3, "com.example.myapp",
   MY_SKU, "subs", developerPayload);
```

Sovelluksessa voidaan kysellä aktiivisia tilauksia getPurchases-metodilla (13):

```
Bundle activeSubs = mService.getPurchases(3, "com.example.myapp",
                  "subs", continueToken);
```

3.3 MAINOKSET

3.3.1 Käyttöönotto

Admob-mainokset on helppo lisätä omaan Android-sovellukseen. Build.gradle-tiedoston dependencies-osioon on lisättävä seuraava (14):

```
compile 'com.google.android.gms:play-services-ads:8.4.0'
```

Tällöin Android Studio lisää projektiin Google Admob -kirjastot.

Kehittäjä voi vaikuttaa mainosten sisältöön Google Play -kehittäjäkonsolilla. Halutessaan kehittäjä voi suodattaa pois sovelluksensa mainonnasta tiettyihin aihealueisiin kuuluvat mainokset. Konsolilla voi myös nähdä tilastoja mainosten kannattavuudesta.

3.3.2 Mainosbannerit

Sovelluksen käyttöliittymän määritteleviin xml-tiedostoihin voidaan lisätä adView-näkymä, joka näyttää mainosbannerin sisällön käyttäjälle. Mainoksiin voidaan hakea sisältö aktiviteetin onCreate-metodissa (14).

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        AdView mAdView = (AdView) findViewById(R.id.adView);
        AdRequest adRequest = new AdRequest.Builder().build();
        mAdView.loadAd(adRequest);

}
```

3.3.3 Koko näytön mainokset

Admob tukee myös koko näytön mainoksia (interstitial ads), joita voidaan näyttää aina kun käyttäjä haluaa siirtyä sovelluksessa näkymästä toiseen. Kun mainos suljetaan, käyttäjä pääsee seuraavaan näkymään. (15.)

Java-koodissa InterstitialAd-luokassa on setAdListener-metodi, jolla voidaan määritellä, mitä tapahtuu, kun käyttäjä sulkee mainoksen (esimerkiksi toiseen näkymään siirtyminen). requestNewInterstitial-metodilla palvelu hakee uuden kokonäytön mainoksen. Metodia kannattaa kutsua heti mainoksen sulkeutuessa, sillä järjestelmä kykenee lataamaan taustalla uutta mainosta asynkronisesti. Jos mainosta alettaisiin hakemaan, kun uusi mainos ilmestyy ruutuun, käyttäjän täytyisi odottaa sen latautumista. (15.)

4 JOHTOPÄÄTÖKSET

Tämän työn tarkoitus oli arvioida mobiilisovellusmarkkinoita liiketoiminnan kannalta. Kuten liiketoiminnassa aina, liiketoimintasuunnitelma on tehtävä hyvin huolellisesti riskianalyyseineen, ja on harkittava tarkkaan tilastojen avulla kannattaako yritystoimintaan lähteä.

Vaikka Android-sovellusten kehittäminen on helppoa ja nopeaa, sovelluksen tekeminen yksinään on silti työlästä, ja voi kestää kuukausia tai jopa vuosia. Kuukausittaisen toimeentulon tienaaminen Android-sovelluksella ei riitä, sillä kehittäjän täytyy kyetä tienaamaan vuosia kestäneen sovelluskehityksen kulut takaisin. Liiketoimintasuunnitelman ja riskianalyysin laiminlyöminen johtaa kaikin puolin kannattamattomaan yritystoimintaan.

Mobiilisovelluskehittäminen on aina suuri riski. 64 % Android-sovellusten kehittäjistä tienaavat vähemmän kuin 500$ jokaista julkaisemaansa sovellustaan kohden kuukaudessa (15). Yritystoimintaa suunnittelevalle luku voi vaikuttaa lannistavalta.

Tuotetta kehittäessä, sekä yritystoimintaa aloittaessa, yrittäjän on aina muistettava, että riski on osa yritystoimintaa. Jos riskejä ei uskalla ottaa, tuotteesta tulee liian tavanomainen, eivätkä asiakkaat ole kiinnostuneita siitä. Liiketoimintaa voi olla mahdoton ennustaa, ja mahdollisuus menestymisestä on aina olemassa. Jopa tuotteesta, jonka pitäisi epäonnistua kaikkien laskelmien ja ennusteiden mukaan, voi tulla suuri menestys.

LÄHTEET

1. van der Meulen, Rob — Rivera, Janessa 2015. Gartner Says Worldwide Smartphone Sales Recorded Slowest Growth Rate Since. Gartner, Inc. Saatavissa: http://www.gartner.com/newsroom/id/3115517 Hakupäivä: 18.4.2016
2. Earn. Android Developers. Saatavissa: http://developer.android.com/distribute/monetize/index.html Hakupäivä: 5.2.2016
3. Vannieuwenborg, Frederic — Mainil, Laurent — Verbrugge, Sofie — Pickavet Mario — Colle, Didier 2012. Business models for the mobile application market from a developer's viewpoint. 16th International Conference on Intelligence in Next Generation Networks. Saatavissa: IEEE Explore -tietokanta (vaatii käyttöoikeuden). Hakupäivä 12.2.2016.
4. Adding AdMob into an Existing App. Google Developers. 2016. Saatavissa: https://developers.google.com/admob/android/existing-app Hakupäivä: 16.4.2016
5. Chaffey, Dave 2006 — Internet Marketing: Strategy, Implementation and Practice. Harlow: Pearson Education Limited. Saatavissa: https://books.google.fi/books?id=G9smMWZ-DWgC (rajoitettu lukuoikeus). Hakupäivä: 16.4.2016
6. Why AdMob?, Admob By Google, Saatavissa: https://www.google.com/admob/monetize.html Hakupäivä: 16.4.2016
7. Android platform/API version distribution, Android Studio 1.5. Saatavissa: http://developer.android.com/sdk/index.html Hakupäivä: 19.3.2016.
8. Launch Checklist. Android Developers. Saatavissa: http://developer.android.com/distribute/tools/launch-checklist.html Hakupäivä: 19.3.2016.
9. Developer program FAQ, Microsoft Developer resources, Saatavissa: https://dev.windows.com/en-us/programs/faq Hakupäivä 16.4.2016
10. Program Membership Details, Apple Developer Program. Saatavissa: https://developer.apple.com/programs/whats-included Hakupäivä 16.4.2016
11. Publishing Overview. Android developers. Saatavissa: http://developer.android.com/tools/publishing/publishing_overview.html
12. In-app Billing. Android developers. Saatavissa: http://developer.android.com/google/play/billing/index.html Hakupäivä 16.4.2016

13. In-app Billing API. Android developers. Saatavissa: http://developer.android.com/google/play/billing/api.html Hakupäivä 16.4.2016
14. Get Started in Android Studio. AdMob for Android. Saatavissa: https://developers.google.com/admob/android/quick-start Hakupäivä: 16.4.2016
15. Interstitial Ads. AdMob for Android. Saatavissa: https://developers.google.com/admob/android/interstitial Hakupäivä: 16.4.2016
16. Wilcox, Mark 1014. State of the Developer Nation: The App Economy Consolidates Before the Next Gold Rush. Saatavissa: http://www.developereconomics.com/the-app-economy-consolidates-before-the-next-gold-rush/ Hakupäivä 16.4.2016

LIITE 2

OULUN AMMATTIKORKEAKOULU

Lauri Miettinen

KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA

KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA

Lauri Miettinen
Opinnäytetyö, osa 2
Syksy 2017
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehityksen suuntautumisvaihtoehto
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Lauri Miettinen
Opinnäytetyön nimi: Klassisen auton elinkaaren seuranta älykkäällä sopimuksella
Työn ohjaajat: Veijo Väisänen
Työn valmistumislukukausi ja -vuosi: syksy 2017 Sivumäärä: 29 + 1 liite

---

Lohkoketjutekniikka sai alkunsa vuonna 2009, kun Satoshi Nakamoto kirjoitti tieteellisen julkaisun, jossa hän suunnitteli tietomallin hajautetulle valuutalle. Nakamoton suunnitteleman järjestelmän pohjalta toteutettiin BitCoin-kryptovaluutta. Bitcoin-maksut tallennetaan hajautettuun tietokantaan, jota kutsutaan lohkoketjuksi. Siitä Vitalik Buterin keksi tehdä Ethereumin, hajautetun sovellusalustan, jonka älykkäät sopimukset ja tiedonsiirtotapahtumat tallennetaan lohkoketjuun.

Lohkoketju on arvokkaan tiedon tallentamiseen perustuva järjestelmä. Bitcoinin tapauksessa arvokas tieto on varallisuus ja valuutta. Klassiset autot ovat arvoesineitä. Niistä voidaan käydä satojen tuhansien ja joskus jopa miljoonien eurojen kauppoja. Tässä työssä suunniteltiin liiketoimintamalli ja luotiin sovellus, jolla voisi esitellä havainnollisesti, miten tulevaisuudessa voitaisiin hyödyntää älykkäitä sopimuksia liikennöinnin alalla. Työn tilaaja on Hilla-ohjelma, joka edesauttaa tulevaisuuden tekniikoiden tutkimusta ja kehitystä Suomessa.

Ohjelmistotyön ensimmäinen osa oli luoda Ethereum-alustalle älykäs sopimus, jota voitaisiin käyttää klassisen auton elinkaaren seurantaan. Toinen osa oli luoda käyttöliittymä, jolla sopimuksen kanssa voisi vuorovaikuttaa. Älykäs sopimus kirjoitettiin Solidity-kielellä, käyttöliittymä luotiin Meteor-sovelluskehyksellä.

Työssä onnistuttiin toteuttamaan suurin osa suunnitelluista käyttötapauksista. Työskentelyä vaikeuttivat eniten Solidity-kielen puutteelliset ominaisuudet. Rajoitteet hidastavat kehitystä ja rajoittavat innovaatioita, joita Ethereumin älykkäillä sopimuksilla voi kehittää. Solidity-kieli on silti helppokäyttöinen ja se on helppo oppia. Meteor on hyvin helppokäyttöinen sovelluskehys. Se soveltuu hyvin Ethereum-alustaan pohjautuviin sovelluksien kehittämiseen.

Asiasanat: ohjelmistokehitys, lohkoketju, BitCoin, Ethereum, JavaScript

ABSTRACT

Oulu University of Applied Sciences
Degree programme in information technology, software development

Author: Lauri Miettinen
Title of thesis: Developing a smart contract for tracking the life cycle of a classic car
Supervisor: Veijo Väisänen
Term and year when the thesis was submitted: fall 2017 Pages: 29 + 1 appendices

The block chain technology began in 2009 when Satoshi Nakamoto wrote a paper on BitCoin, a system for a decentralized currency. BitCoin-transactions are saved into a decentralized data based called the block chain. Years later, a man named Vitalik Buterin invented Ethereum, a decentralized computing platform whose smart contracts and information transactions are saved into the block chain.

The block chain is a system for saving any valuable information. In BitCoin's case, the valuable information is wealth and currency. Classic cars are valuable. Classic car sales can cost hundreds of thousands, even millions of euros. In this thesis a business model was designed, and a decentralized application was created. The goal of the project was to show in a concrete way how smart contracts could be used in the future. This thesis was made for the Hilla-program – a Finnish research project for studying future technologies.

The first part of the application was to create a smart contract for the Ethereum-platform for tracking the life cycle of a classic car. The second part was to create an interface to allows users to interact with the smart contract. The smart contract was written with the Solidity-language and the interface was created with the Meteor-framework.

Most of the planned use-cases were successfully implemented. The greatest obstacle in the project was the lacking features of the Solidity-language. The limitations will slow down developers and restrict innovations that could be developed with smart contracts. Solidity-language is, however, easy to learn. Meteor is a very useful and easy-to-use framework. It can be easily applied to interact with Ethereum -smart contracts.

Keywords: software development, block chain, BitCoin, Ethereum, JavaScript

SISÄLLYS

1 JOHDANTO

Lohkoketjutekniikka sai alkunsa vuonna 2009, kun Satoshi Nakamoto kirjoitti tieteellisen julkaisun, jossa hän suunnitteli tietomallin hajautetulle valuutalle. Nykymaailmassa valuutta liikkuu pankin tai muun laitoksen kautta. Nakamoton idea oli luoda valuutta, josta ei ole vastuussa kukaan keskitetty taho, jossa maksajan ei tarvitse luottaa pankin tai valuuttapalvelun tietoturvaan ja palveluun. Nakamoto suunnitteli hajautetun järjestelmän rahan lähettämiselle ja maksutapahtumien todentamiselle. Nakamoton suunnitteleman järjestelmän pohjalta toteutettiin BitCoin-kryptovaluutta. (1.)

Bitcoin-maksut tallennetaan hajautettuun tietokantaan, jota kutsutaan lohkoketjuksi. Mikään yksi yritys tai yksi taho ei ole vastuussa Bitcoin-maksuista. Kukin maksutapahtuma lähetetään vertaisverkkoon, jossa se todennetaan. Kuka tahansa voi liittää tietokoneensa Bitcoin-verkkoon todentamaan maksutapahtumia. Maksutapahtumat todennetaan menetelmällä, joka vaatii paljon laskentatehoa. Järjestelmän huijaaminen vaatisi hyökkääjältä enemmän laskentatehoa kuin mitä on kaikilla verkossa olevilla rehellisillä osallisilla. Bitcoin-verkkoon hyökkääjän on ainakin teoriassa mahdollista luoda itselleen rahaa, jota hänellä ei oikeasti ole. Käytännössä hyökkääjällä pitäisi silloin olla käytössään enemmän laskentatehoa kuin koko muulla maailmalla. (1.)

Tekniikka herätti maailmalla mielenkiintoa. Nakamoto pohti hajautettua valuuttaa suunnitellessaan, että olisi suunnitellut hajautetun ohjelmointikielen, mutta päättikin tehdä yksinkertaisemman järjestelmän, sillä ymmärsi tekniikan olevan kokeellinen ja haastava luonteeltaan. Siitä Vitalik Buterin keksi tehdä Ethereumin, hajautetun sovellusalustan, jonka tiedonsiirtotapahtumat tallennetaan lohkoketjuun. (2.)

Lohkoketju on arvokkaan tiedon tallentamiseen perustuva järjestelmä. Bitcoinin tapauksessa arvokas tieto on varallisuus ja valuutta. Klassiset autot ovat arvoesineitä. Niistä voidaan käydä satojen tuhansien ja joskus jopa miljoonien eurojen kauppoja. Tässä työssä pohditaan liiketoimintamalli ja käyttötapaukset lohkoketjutekniikkaa hyödyntävälle sovellukselle. Ohjelmistotyön ensimmäinen osa oli luoda Ethereum-alustalle älykäs sopimus, jota voitaisiin hyödyntää klassisen auton elinkaaren seurantaan.

Autoja huollettaessa huoltotietojen tallennus olisi haviteltava ominaisuus sekä tavallisille autoille että klassisille autoille. Tavallisilla autoilla varmennettuja ja arkistoituja huoltotietoja voivat hyödyntää vakuutusyhtiöt, huoltoyhtiöt sekä auton omistajat.

Klassisia autoja on huollettava, sillä ne voivat olla kymmeniä vuosia vanhoja ja niiden osat kuluvat väistämättä. Lohkoketjuun tallennettua tietoa on vaikea muokata jälkeenpäin. Jos klassisen auton huoltotiedot ja käyttötilastot tallennettaisiin lohkoketjuun, auton omistaja voisi vakuuttaa huutokaupoissa ostajan autonsa arvosta.

Klassiset autot liittyvät myös liikennöinnin alaan. Liikennöinnin alalla on paljon tulevaisuuden sovelluksia, joissa voitaisiin hyödyntää lohkoketjutekniikkaa. Koska tämän työn aihe liittyy oleellisesti myös liikennöintiin, tässä työssä tehdyt havainnot ja tulokset pätevät myös liikennöinnin alalla yleisesti.

Toinen osa työtä oli luoda käyttöliittymä, jolla lohkoketjuun tehtäviä merkintöjä voi tehdä ja tarkastella. Tässä työssä kuvaillaan työn suunnittelua, toteutusta ja lopputulosta.

Työn tilaaja on Hilla-ohjelma, joka edesauttaa tulevaisuuden tekniikoiden tutkimusta ja kehitystä Suomessa. Työssä oli tavoitteena tehdä sovellus, jonka voi esitellä yhdellä tietokoneella, ja joka näyttää havainnollisesti, mitä älykkäillä sopimuksilla ja lohkoketjuilla voidaan tehdä tulevaisuudessa.

2 TEKNIIKKAAN TUTUSTUMINEN

Älykkään sopimuksen koodaaminen vaati Ethereum-alustan perustoiminnallisuuden ymmärrystä. Koska Ethereum-alusta hyödyntää lohkoketjua, paras tapa ymmärtää lohkoketjujen toiminta on ymmärtää alkuperäisen lohkoketjusovelluksen, BitCoinin, perustoiminnallisuus. Työn alussa tutustuttiin BitCoin-sovelluksen, lohkoketjutekniikan ja Ethereum-alustan perusteisiin.

2.1 Bitcoin ja lohkoketju

Lohkoketjussa kukin tiedonsiirto tallennetaan lohkoon. Yhdessä lohkossa on monia tiedonsiirtotapahtumia, kuten Bitcoinin tapauksessa rahan siirtotapahtumia. Kullakin lohkolla on viittaus sitä edeltäneeseen lohkoon. Sana *lohkoketju* on kielikuva verkon toimintatavasta. (1.)

Kun uusi tiedonsiirto tehdään, lohkoketjuverkkoon lähetetään tieto tiedonsiirrosta. Vertaisverkossa tieto leviää solmulta toiselle. Bitcoin-verkossa uudet tiedonsiirtotapahtumat ovat ensin todentamattomien tapahtumien rekisterissä. Louhijasolmujen tehtävänä on luoda uusia lohkoja, joihin tiedonsiirtotapahtumat tallennetaan. Uuden kelvollisen lohkon löytäminen on sattumanvaraista, ja vaatii monia kokeiluja. Uusi tiedonsiirtotapahtuma päätyy lohkoketjuun seuraavasti (1):

1) Uusi tiedonsiirtotapahtuma lähetetään solmulta toiselle. Tiedonsiirron informaatio leviää koko verkon louhijasolmuille.
2) Kukin louhijasolmu kerää uusia tiedonsiirtotapahtumia lohkoon.
3) Kukin solmu yrittää laskea kelvollisen tiivisteen (engl. hash) lohkollensa.
4) Kun solmu onnistuu laatimaan kelvollisen lohkon, se lähettää tiedon uudesta lohkosta verkon muille solmuille.
5) Solmut tarkistavat, ovatko ilmoitetut maksutapahtumat kelvollisia eli oliko lähettäjällä tarpeeksi varallisuutta lähettää rahoja, joita hän lähetti.
6) Mikäli solmu toteaa lohkon olevan kelvollinen, se voi hyväksyä lohkon oman lohkoketjunsa uusimmaksi lohkoksi. Solmu voi aloittaa keräämään uusia maksutapahtumia taas seuraavaan lohkoon.

Järjestelmä toimii niin kauan kuin suurin osa verkon solmujen laskentatehosta pysyy rehellisinä. Koska lohkoketjusta on kopio jokaisella verkon tietokoneella ja koska uuden lohkon tekeminen vaatii paljon laskentatehoa, lohkoon päätyneen tiedon muuttaminen jälkeenpäin on lähes mahdotonta. Jokainen louhijasolmu tarkistaa, onko lähettäjällä varaa tehdä rahansiirtoa. (1.)

Lohkon laskenut solmu saa palkkioksi Bitcoineja uuden lohkon luomisesta. Tähän palkkioon perustuu lohkoketjukonesalien liiketoiminta. Kun louhijasolmu onnistuu luomaan uuden lohkon, palkkioksi saatu kryptovaluutta voidaan myydä ostajalle, joka haluaa vaihtaa tavallista valuuttaa kryptovaluutaksi. (1.)

Epärehellinen louhijasolmu voisi yrittää huijata muita solmuja lähettämällä uuden lohko verkostoon, vaikka sen maksutapahtumat eivät olisi kelvollisia. Tämän solmun omistaja saisi petoksestaan palkkioksi kryptovaluuttaa. Kun lohko lähetetään verkkoon, kaikki muut solmut huomaavat, etteivät sen maksutapahtumat olekaan kelvollisia. Louhijasolmujen kannattaa siis olla rehellisiä, koska petoksen tekemisen jälkeen heidän varallisuutensa katoaisi välittömästi. (1.)

Samasta lohkoketjusta on kopioita lukemattomilla tietokoneilla. Tämäkin piirre tekee lohkoketjun hakkeroinnista käytännössä hyvin vaikeaa. Jos hyökkääjä haluaisi muunnella ketjussa olevia maksutapahtumia, hänen täytyisi silloin murtautua yhtäaikaisesti kaikkiin tietokoneisiin maailmassa, joissa on kopio lohkoketjusta. (3.)

2.2 Hajautetut sovellukset

Ethereum ja Bitcoin ovat hajautettuja sovelluksia. Hajautetulle sovellukselle on monia toisistaan eroavia määritelmiä. Määritelmien kesken yhteisenä tekijänä on sovellusten piirre, jossa sovellukset tallentavat tietonsa vertaisverkkoon. Lohkoketju on yksi tiedon tallentamiseen suunniteltu vertaisverkko. Toinen piirre on se, ettei hajautetuilla sovelluksilla ole ketään yhtä auktoriteettia, tahoa tai ylläpitäjää, jolla on päätäntävalta sovelluksessa. (4.)

Hajautetuissa sovelluksissa on monia hyötyjä verrattuna perinteisiin, keskitettyihin sovelluksiin. Yksi hyöty on se, että ne eivät koskaan ole pois käytöstä. Perinteisissä palveluissa voi tulla palvelinvikoja. Hajautettu sovellus ei voi kokonaan poistua toiminnasta, elleivät kaikki vertaisverkon solmut lopeta toimintaansa yhtä aikaa. (5.)

Vertaisverkkosovellukset eivät ole uusi, ennennäkemätön keksintö tietotekniikan historiassa. Esimerkiksi BitComet, joka on tiedostonsiirtoon käytetty vertaisverkko. Monet ohjelmistot jakavat ohjelmistopäivityksensä vertaisverkossa. Muun muassa Windows-käyttöjärjestelmän päivitykset tulevat vertaisverkosta. (6.)

Bitcoinin historiassa ohjelmoijayhteisö alkoi kehittämään hajautettua alustaa, joka perustuu Bitcoin-verkkoon. Näille alustoille kehitettiin tukia monen eri tyyppisille tiedonsiirtotapahtumille. Ajan mittaan ohjelmoijayhteisö keksi uusia käyttötapauksia, jolloin olemassa olevien tiedonsiirtotapahtumatyyppien rajat tulivat vastaan. Silloin alustan kehittäjien täytyi taas luoda uusia tiedonsiirtotapahtumatyyppejä. Näin alustalla kehitettävät innovaatiot olivat riippuvaisia siitä, minkä tyyppisiä tiedonsiirtotapahtumia kehitettiin ohjelmoijayhteisössä. (7.)

2.3 Ethereum-alusta

Bitcoin-verkossa voi ainoastaan lähettää valuuttaa tililtä toiselle. Ethereum-alustassa on myös tilejä, valuuttaa ja rahansiirtotapahtumia, mutta Ethereum-lohkoketjuun voi myös tallentaa koodia. Kehittäjät voivat kirjoittaa koodia, ja luoda omia hajautettuja sovelluksiaan. Ethereum-alustalle kehitettyjä sovelluksia kutsutaan *älykkäiksi sopimuksiksi*. (4; 8.)

Solidity on suosituin ohjelmointikieli Ethereum -älykkäiden sopimusten kirjoitusta varten. Muita suosittuja kieliä ovat muun muassa LLL ja Serpent. (5.)

Ethereum-solmuun, oli kyseessä yksityisen käyttäjän solmu tai louhijasolmu, kuuluu Ethereum-virtuaalikone, joka on älykkäiden sopimusten ajoympäristö. Ethereum-virtuaalikone voi ajaa koodia, joka suorittaa loogisia operaatioita. Ethereum-alusta luo vertaisverkon, jossa käyttäjän koneella käynnissä oleva virtuaalikone lähettää tiedonsiirtotapahtumia Ethereum-verkkoon. Ethereum-verkko koostuu maailmanlaajuisesti yhteen liittyneiden Ethereum-solmujen vertaisverkosta. Jos Ethereum-verkkoa haluaa käyttää, tulee omalle tietokoneelle käynnistää Ethereum-solmu, esimerkiksi GoEthereum (ks. 4.3.1).

Ethereum-alustalla voi ajaa minkä tahansa protokollan missä tahansa lohkoketjussa. Ethereum-alusta ei tarvitse vain tietyllä tekniikalla toteutettua lohkoketjua, vaan alusta voi hyödyntää minkä tahansa mallista lohkoketjua, mikä saatetaan kehittää tulevai-

suudessa. Ethereum-virtuaalikone ei ole riippuvainen mistään tietystä ohjelmointikielestä, eikä virtuaalikoneen ajama koodi ole riippuvainen Soliditystä. Ethereum ei tarvitse mitään tiettyä tiedonsiirtoprotokollaa ollakseen yhteydessä vertaisverkkoon. Nämä periaatteet olivat taustalla, kun Buterin suunnitteli Ethereum-alustaa. (4; 7.)

Ethereum-virtuaalikone on Turing-täydellinen, joten virtuaalikoneella ei ole mitään loogisia rajoitteita liittyen siihen, mitä sillä voi tehdä. Turing-täydellisyys tarkoittaa sitä, että ohjelmointikielellä voi ratkaista minkä tahansa laskennallisen ongelman. Turing-täydellistä ohjelmointikieltä rajoittaa ainoastaan tietokoneen muistin määrä. (9.)

BitCoin-vertaisverkossa on bitcoin-rahayksikön siirtämistä varten tiedonsiirtotapahtumia. Ethereum-verkossa vastaava rahayksikkö on eetteri (engl. ether). Eetteri on bitcoinin kaltainen kryptovaluuttarahayksikkö, jota voi käyttää maksamiseen Ethereum-verkossa. (8.)

Kun tiedonsiirtotapahtuma tehdään tietokoneella, tietokoneella oleva Ethereum-solmu lähettää tiedon siitä koko Ethereum-vertaisverkkoon. Ethereum-verkossa maksutapahtuma todennetaan. Minkä tahansa maksutapahtuman tekeminen vaatii kaikkia Ethereum-verkon louhijasolmuja ajamaan saman maksutapahtuman sekä sen sisältämät loogiset operaatiot omalla tietokoneellaan. Maksutapahtuman alkuperäisen lähettäjän on maksettava tämä hinta korvauksena kaikille louhijasolmuille. Tätä kutsutaan kaasumaksuksi (gas price). Työläämmät laskentatoimet vaativat suuremman kaasumaksun tiedonsiirtotapahtuman lähettäjältä. (8; 7.)

Ethereum-maksutapahtumilla on maksimikaasurajoite, kuinka paljon ne saavat kuluttaa resursseja Ethereum-verkon louhijasolmuissa. Jos verkossa on sopimus, joka ajaa raskaita laskuoperaatioita, siihen tehdyt tiedonsiirtotapahtumat voidaan hylätä, jos niiden suorittaminen ylittää maksimikaasurajoitteen. Ethereum-verkon kaasurajoitteen takia verkko ei mene tukkoon, vaikka sinne lähetettäisiinkin älykkäitä sopimuksia, jotka vaativat liian raskasta laskentaa. (10; 7.)

Ethereum-virtuaalikone, kaikki siihen liittyvä ohjelmisto ja kaikki alustan kehittämiseen liittyvä ohjelmisto on avoimen lähdekoodin ohjelmistoa. Ethereum-ohjelmistoja kehitetään maailmanlaajuisesti ja hajautetusti GitHub-palvelun sekä muiden versionhallintapalveluiden avulla. (4.)

3 SUUNNITTELU JA KÄYTTÖTAPAUKSET

Suunnitteluvaiheessa pohdiskeltiin työssä luotavan sovelluksen käyttötapauksia. Sovellus pyrittiin suunnittelemaan siten, että se hyödyntää mahdollisimman paljon lohkoketjujen ja Ethereum-alustan piirteitä.

3.1 Kuvaus käyttötapauksista

Auton omistajalla on käytössään auton elinkaarta seuraava älykäs sopimus. Auton omistajan tavoitteena on kerätä auton älykkääseen sopimukseen merkintöjä auton elinkaaressa tapahtuneista merkityksellisistä tapahtumista. Tällaisia tapahtumia voivat olla esimerkiksi auton huoltotoimenpiteet, kilpailuvoitot ja asiantuntija-arviot. Merkintöjen tavoite on vakuuttaa tuleva ostaja auton arvokkuudesta. Tässä työssä merkinnöistä käytetään nimeä *kohokohta* eli *highlight*.

Ajatuksena verkkosovelluksessa on, että kuka tahansa ihminen maailmassa voisi mennä auton älykkään sopimuksen kotisivuille tarkastelemaan auton nykytilaa, siinä olevia kohokohtamerkintöjä ja historiaa.

Käyttäjät voivat tehdä verkkosovelluksella kohokohtapyyntöjä (highlight request) älykkääseen sopimukseen. Kohokohtapyynnön voisi esimerkiksi tehdä auton korjannut mekaanikko tai asiantuntija, joka tekee auton autenttisuudesta ja kunnosta arvion. Kohokohtapyynnön tekijä voi pyytää pienen rahallisen korvauksen vaivannäöstä. Esimerkiksi auton korjaava mekaanikko voisi tehdä kohokohtapyynnön, jossa lukee ”Huolsin tämän auton Välivainion huoltoasemalla.” Hän voisi pyytää lomakkeen täyttämisen vaivannäöstä viisi euroa. Lomakkeen täyttämiseen kuluu muutama minuutti, mutta sekin vaatii pientä vaivannäköä. Ihmiset ovat suostuvaisempia muutaman minuutin vaivannäköön, jos he saavat siitä pienen rahallisen korvauksen.

Kukin kohokohtapyyntö voi maksaa auton omistajalle useita euroja tai kymmeniäkin euroja jokaisesta merkinnästä. Merkinnät voivat silti todistaa auton arvokkuuden tulevalle ostajalle. Huutokaupassa tarjouksen tekijä saattaa tehdä suuren niiden merkintöjen perusteella, jotka ovat lohkoketjussa. Tällöin auton myyjän maksamat kymmenien eurojen summat maksavat itsensä moninkertaisesti takaisin.

Lisäksi kuka tahansa ihminen maailmassa voisi tehdä tarjouksen autosta. Kaikki tarjoukset ovat julkisia, ja kaikki pystyvät tarkastelemaan niitä. Julkisista tarjouksista voisi tehdä tilastoja, jotka antavat suuntaa-antavan kuvan auton arvosta. Auton omistaja voi hyväksyä tarjouksen, jolloin hän voi toimittaa auton ostajalle. Sopimuksen omistajuusoikeus siirtyy autokaupan yhteydessä.

3.2 Älykkään sopimuksen hyöty sovelluksessa

Autosovelluksen voisi kehittää perinteisenä palveluna, joka toimii tavallisella palvelimella, ja jonka merkinnät tallennetaan perinteiseen tietokantaan. Lohkoketjussa ja älykkäissä sopimuksissa on piirteitä, joista tämä sovellus voi hyötyä.

Toinen etu on lohkoketjumerkintöjen pitkäikäisyys. Klassisen auton elinkaari voi olla vuosikymmeniä, eivätkä kovin monet yritykset ole olemassa niin kauan. Jos tavallinen palvelu lopettaa toimintansa, tietokannassa olevat merkinnät voivat kadota. Lohkoketju on paljon pitkäikäisempi kuin perinteinen tietokanta. Kopiot tiedoista ovat jokaisella Ethereum-verkon tietokoneella, joten tiedot eivät voi tuhoutua lopullisesti.

Lohkoketjussa kaikki maksutapahtumat ovat julkisia ja läpinäkyviä. Läpinäkyvyys sopii tämän sovelluksen käyttötapauksiin hyvin. Jos kaikki auton elinkaaressa tulleet tapahtumat ovat julkisia, on huijausten ja väärinkäytösten tekeminen kannattamatonta auton omistajalle. Sovelluksella tehtyjä merkintöjä tarkkailevat voivat tehdä tutkimusta ja havaita, jos auton elinkaaren merkinnöissä esiintyy väärää tietoa tai ristiriitoja. Sosiaalisen median aikakautena tieto petollisesta auton omistajasta leviää nopeasti ihmiseltä ihmiselle. Omistajan maineen menetyksen pelko voi olla suuri. Mikäli tällainen sovellus on oikeasti käytössä tulevaisuudessa, autonomistajat varmasti ymmärtävät, että on kannattavaa pysyä rehellisenä lisättäessä kohokohtia.

Lohkoketjuissa on piirre, että lisättyjä merkintöjä ja tiedonsiirtoja ei voi poistaa jälkikäteen. Epärehellinen autonomistaja ei siis pysty peittelemään jälkiään poistamalla tai muokkaamalla vanhoja merkintöjä. Tämä vahvistaa luottamusta sovelluksen käyttäjäkunnassa auton omistajan ja autosta kiinnostuneiden ostajien välillä.

4 TYÖHÖN LIITTYVÄ OHJELMISTO

4.1 Solidity-kääntäjä

Solidity-kääntäjä kääntää Solidity-kielen Ethereum-tavukoodiksi, jonka Ethereum-virtuaalikone muuntaa konekäskyiksi. Solidity-kieli on hyvin yksinkertainen ja helposti opittava. Kirjoitusasultaan se muistuttaa JavaScriptiä. Ohjelmoijayhteisö on kehittänyt monia Solidity-kääntäjiä. Yksi kääntäjä on Browser Solidity (ks. 4.3). Kääntäjiä on myös Truffle-sovelluskehyksessä (ks. 4.5) sekä Ethereum Wallet -ohjelmassa (ks. 4.8).

4.2 Web3-kirjasto

Web3 on javascript-kirjasto, joka pystyy lähettämään kutsuja paikalliselle ethereum-solmulle. Web3:n avulla internet-selaimessa olevat sovellukset voivat vuorovaikuttaa Ethereum-verkon kanssa. Web3:sta on olemassa npm-paketti, jonka voi ottaa käyttöön helposti Meteorissa (ks. 4.6). Kirjasto on hyvin dokumentoitu esimerkkeineen Ethereumin GitHubin wiki-sivuilla. (11.)

4.3 Browser Solidity

Browser Solidity on selaimessa käytettävä kääntäjä. Sen etuna on, että se toimii kaikilla tietokoneilla ilman ylimääräisten ohjelmien asennusta. (12.)

4.4 TestRPC

TestRPC on Ethereum älykkäiden sopimusten kehittämistä varten tehty node-palvelinsovellus, joka on tehty mukailemaan hyvin tarkasti Ethereum-lohkoketjun toimintaa. TestRPC toimii kehittäjän paikallisella tietokoneella. Maksutapahtumien todentaminen tapahtuu TestRPC:ssä hyvin nopeasti, eikä maksutapahtumista tarvitse maksaa oikeaa, rahanarvoista eetteriä. (13.)

Tätä työtä kehittäessä TestRPC:n asentaminen Windows-käyttöjärjestelmälle vaati ohjelmistoa, jota ei ollut valmiina Windowsissa. Asentamisessa käytettiin Microsoft Visual Studio Community Edition -ohjelman mukana tulleita resursseja. TestRPC:n kehittäjien mukaan 25.10.2017 Windows-asennus on helpottunut, eikä ylimääräisten

resurssien asennusta vaadita enää. TestRPC on yhdistynyt Truffle-sovelluskehyksen (Ks. 4.5) ominaisuudeksi, ja ottanut nimekseen Ganache CLI. (14; 15.)

4.5 Truffle-sovelluskehys

Truffle on Ethereumin älykkäiden sopimusten kehittämistä varten luotu sovelluskehys. Sen on tarkoitus helpottaa Solidity-kielellä älykkäiden sopimusten koodin kirjoittamista. Sovelluskehyksen avulla voi tehdä älykkäiden sopimusten jatkuvaa integraatiota Ethereum-lohkoketjuun. Trufflen avulla voi myös kirjoittaa automatisoituja testejä sopimuksille. (16.)

4.6 Meteor

Meteor on sovelluskehys, joka yhdistää Node-palvelimen ja MongoDB -tietokannan hyvin helppokäyttöisesti. Meteor on saanut maailmalla huomiota, koska sen avulla voi tehdä samanaikaisesti verkkosivuja sekä mobiilisovelluksia. Meteorilla on helppo luoda reaktiivisia käyttöliittymiä, jolloin sovelluksen käyttöliittymä näyttää reaaliaikaisesti kaikki päivitykset, mitkä tulevat tietokantaan. (17.)

4.7 GoEthereum (geth)

Go-kielellä ohjelmoitu Ethereum-virtuaalikone. Tämän työn sovellusta kehittäessä geth-solmua ei juurikaan käytetty, koska kehityksessä käytettiin lähinnä TestRPC-testiverkkoa. Geth-solmua tarvittaisiin, jos työssä kehitetty älykäs sopimus haluttaisiin sijoittaa oikeaan Ethereum-verkkoon. (18; 19.)

4.8 Ethereum Wallet

Ethereum Wallet on helppokäyttöinen ohjelma, jolla käyttäjä voi vuorovaikuttaa Ethereum-verkon kanssa. Ohjelma käynnistää tietokoneelle geth-solmun automaattisesti, vaatimatta käyttäjältä ymmärrystä tekniikan yksityiskohdista. Ethereum Walletin graafisella käyttöliittymällä voi tehdä maksutapahtumia Ethereum-verkkoon. Ohjelmassa on Solidity-kääntäjä, ja ohjelmalla voi lähettää sopimuksensa Ethereum-verkkoon. Valitettavasti Ethereum Wallet ei toimi TestRPC:n kanssa, koska TestRPC toteuttaa vain osan kaikista rajapinnoista, joita geth-solmussa on. Maksutapahtumia lähettäessä Ethereum Wallet käyttää rajapintaa *signAndSendTransaction*, jota ei ole toteutettu TestRPC:ssä. (20.)

5 TYÖN KULKU

Aloituspalaverin pitämisen jälkeen aloitettiin pohtimaan työn tarkoitusta. Tässä vaiheessa keksittiin työn idea, siitä tehtiin vaatimusmäärittelydokumentti, jonka työn tilaaja hyväksyi.

Tutustuttiin Ethereumiin käyttämällä Ethereum Wallet -ohjelmaa. Työtä varten perustettiin git-repositorio (21). ClassicCarChain-älykkään sopimuksen koodaus aloitettiin. (Lopullinen ohjelmakoodi on liitteessä 1.)

Työskenneltiin geth-testiverkon parissa, kunnes todettiin, että nopeaa, iteroivaa työskentelytapaa varten testiverkko oli liian hidas. Tämän vuoksi asennettiin TestRPC-testiverkko tietokoneelle, jonka jälkeen älykkään sopimuksen suorituskyky näytti paranevan.

Työssä kokeiltiin kehittää automaattisia testejä Truffle-sovelluskehyksellä. Automaattisten testien koodaus oli hyvin työlästä, joten niiden kehittämisestä luovuttiin myöhemmin. Työskentelyn ajan Trufflea käytettiin lähettämään (engl. *deploy*) sopimus TestRPC-testiverkkoon.

Meteor-sovelluskehyksen käyttö aloitettiin. Sillä tehtiin Ethereumiin liittymättömiä harjoitustöitä, jotta kehittämisen perusteet tulisivat tutuiksi.

Älykkään sopimuksen ja Meteor-sovelluksen välisessä rajapinnassa alkoi ilmetä ongelmia, jotka johtuivat Solidity-kielen ja virtuaalikoneen teknisistä rajoitteista. Ongelmien ratkaisemiseksi koodattiin uudelleen älykästä sopimusta ja kokeiltiin eri ratkaisuja.

Solidity-kielessä ole null-käsitettä. Jos funktiota GetHighlight kutsutaan kysyen kokonaislukuavaimella, jota ei vastaa mikään alustettu kohokohta, silloin funktio palauttaa Highlight-tietueen, jonka kaikki arvot ovat oletusarvoissaan. Muussa ohjelmointikielessä kutsu olisi saanut aikaan poikkeuksen. Solidity-kielen toteutus vaikeuttaa tilannetta, kun halutaan tarkistaa, onko mapping-tietorakenteen arvoja alustettu. Ongelma ratkaistiin lisäämällä highlights mapping -tietorakenteeseen tietue *initialized* (alustettu). Tämä tietue asetetaan arvoon *tosi* aina, kun tietue alustetaan. Tietue on *epätosi,*

jos arvoa ei ole alustettu. Koodissa voidaan tarkistaa erikseen, palauttiko funktio arvoa, joka oli alustettu. Huoltotiedot (maintenanceData) ovat yksi tietueiden joukossa, mutta huoltotietoja ei käytetty lopullisessa sovelluksessa. (Kuva 1.)

```
411     struct Highlight{
412         uint id;
413         bool initialized;
414         uint highlightType;
415 
416         address maker;
417         uint requestCreationDateTime;
418         uint reward;
419         string message;
420 
421         bool approvedToChain;
422         bool madeByOwner;
423         uint additionToChainDateTime;
424 
425         MaintenanceTasks maintenanceData;
426     }
```

KUVA 1. ClassicCarChain.sol-tiedostossa lopulliset tietueet, joita on kohokohdassa.

Vastaan tuli ongelma käyttötapauksessa ”vierailijan pitää kyetä näkemään kaikki kohokohdat, mitä autolla on”. Web3-rajapinnassa ei ole keinoa hakea kaikki mapping-tietorakenteen alkioita yhdellä funktiokutsulla. Ongelma ratkaistiin tekemällä Meteor-sovellukseen kiertokyselyrakenne, joka hakee yksitellen kaikki alkiot sopimuksesta. (Kuva 2.)

Solidity-koodilla pystyy hakemaan yksittäisiä tietueita mapping-rakenteesta. Sopimukseen tehtiin mapping-rakenne, jossa kokonaislukuavainta (uint) vastasi Highlight struct-tietorakenne. Mapping-rakenteesta pystyi hakemaan yhden tietueen, jos funktiolle annettiin sitä vastaava kokonaislukuavaimen arvo (_id)

```
110     ////////////////////////////////////////
111
112     mapping(uint => CCClib.Highlight) private highlights;
113
114     ////////////////////////////////////////
```

```
209     function GetHighlight(uint _id) public returns (
210         uint _highlightType,
211
212         bool _initialized,
213
214         address _maker,
215         uint _requestCreationDateTime,
216         uint _reward,
217         string _message,
218
219         bool _approvedToChain,
220         bool _madeByOwner,
221         uint _additionToChainDateTime
222         ) {
223
224         CCClib.Highlight h = highlights[_id];
225
226         _highlightType = h.highlightType;
227
228         _initialized = h.initialized;
229         _maker = h.maker;
230         _requestCreationDateTime = h.requestCreationDateTime;
231         _reward = h.reward;
232         _message = h.message;
233
234         _approvedToChain= h.approvedToChain;
235         _madeByOwner = h.madeByOwner;
236         _additionToChainDateTime = h.additionToChainDateTime;
237     }
```

KUVA 2. Kohokohdan haku älykkäästä sopimuksesta tiedostossa ClassicCar-Chain.sol.

Kiertokyselyrakenne on JavaScript-koodissa 03-eth-highlights.js (kuva 3). Koodi ajetaan joka kerta, kun havaitaan, että uusi lohko on lisätty Ethereum-lohkoketjuun (rivi 41, web3.eth.filter('latest').watch). Jotta kaikki kohokohdat saataisiin haettua, täytyy ensin hakea älykkäästä sopimuksesta kohokohtien määrä (rivillä 47, highlightIndex-funktio). Tämän jälkeen käydään läpi kohokohtia for-lauseessa ja kutsutaan yksitellen jokaista indeksiarvoa kohden sopimuksesta kohokohdan tiedot (rivi 56, GetHighlight-funktiokutsu).

```
web3.eth.filter('latest').watch(function(e) {
    if(!e) {

        //HighlightIndex:
        var m_highlightIndex=0;

        var m_highlightIndex = Helpers.convertBigNumber(
        f_contractInstance().highlightIndex());

        Session.set(keyHighlightIndex,m_highlightIndex);

        //Loop through all of the highlights, save them to
        an array in this module.
        var iteratedHighlights = [];

        for (var i = 0; i < m_highlightIndex; i++){

            var hArray = f_contractInstance().GetHighlight.
            call(i);

            var newH = new Highlight(i,hArray);

            if (newH.initialized){
                iteratedHighlights.push(newH);

            }
        }

        highlights = iteratedHighlights;
        Session.set(keyHighlights, highlights);
    }
});
```

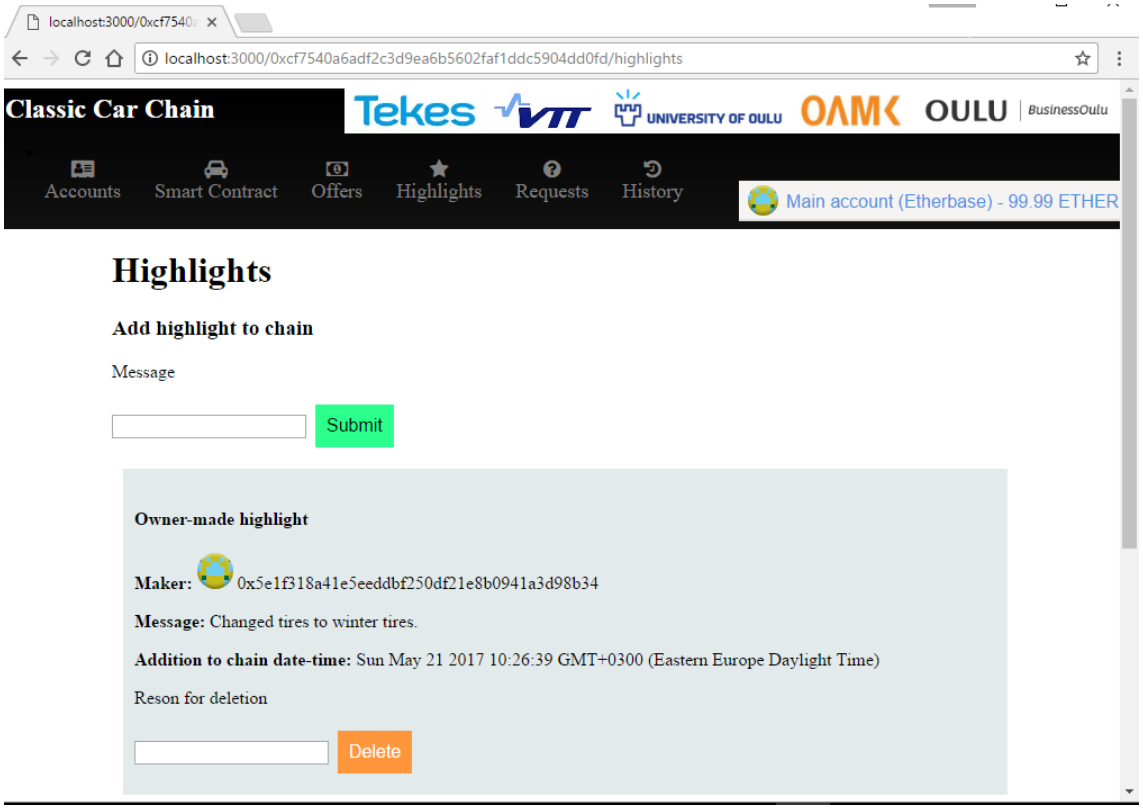
KUVA 3. Näyte tiedostosta 03-eth-highlights.js. Koodissa on kiertokyselyrakenne, jolla haetaan kaikki kohokohdat Meteor-sovellukseen.

Hyviä ohjelmistosuunnitteluperiaatteita pyrittiin seuraamaan suunnittelemalla sopimus uudelleen siten, että jokainen kohokohtamerkintä olisi Ethereum-lohkoketjussa oma alisopimuksensa, joihin ClassicCarChain-pääsopimus tekisi kutsuja. Suunnitelmasta jouduttiin luopumaan, koska Ethereum-virtuaalikoneessa sopimus ei voi hakea toisesta sopimuksesta arvoa, jonka pituus on muuttuva. Tällaisia tietueita ovat muun muassa merkkijonot (string) sekä taulukot (array). (22.)

Lopuksi tehtiin käyttöliittymää. Ethereum-yhteisön kehittämää dappstyles.styl-tyyliä kokeiltiin (23). Työssä päädyttiin tekemään oma yksinkertainen CSS-tyyli.

6 LOPPUTULOKSET

Älykkään sopimuksen Solidity-koodi on liitteessä 1. Työn GitHub-repositorion luemi-nut-tiedostossa on ohjeita Meteor-sovelluksen käynnistämistä varten (21). Kuvassa 4 on auton omistajan näkymä sovelluksen etusivuilta. Yläpalkissa on navigointilinkit, oikeassa yläkulmassa aktiivisen tilin vaihtaminen. Yläpalkin alapuolella on lomake kohokohdan lisäämistä varten. Sininen laatikko on yksi kohokohta, jossa auton omis-taja näkee poistamispainikkeen sekä tekstikentän, johon hän voi kirjoittaa syyn mer-kinnän poistamiselle.



KUVA 4. Lopullinen sovellus kohokohtasivulta.

6.1 Toteutuneet käyttötapaukset

Työssä suunnitelluista käyttötapauksista toteutui suurin osa. Kaikki tärkeimmät käyt-tötapaukset, kuten kohokohtien ja -pyyntöjen lisääminen, toteutuivat.

- Auton omistaja näkee sovelluksessa eri toimintoja kuin muut käyttäjät.

- Kun käyttäjä käynnistää sovelluksen, hän pystyy vaihtamaan käyttäjätiliään. Kun käyttäjätiliä vaihtaa, käyttöliittymä päivittyy välittömästi näyttämään käyttäjälle ne ominaisuudet ja tiedot, mitä käyttäjällä on oikeus nähdä.
- Todennus tapahtuu Ethereum-verkossa. Kuka tahansa voisi esimerkiksi kutsua verkkosivuilla funktiota, jolla saisi auton omistajuuden (gainOwnership). Tällöin Ethereum-verkossa havaitaan, ettei kutsun lähettänyt tili olekaan auton omistaja, jolloin mitään ei tapahdu.
- Omistaja pystyy lisäämään kohokohtia omaan autoonsa. Kohokohdassa on tietueina muun muassa merkinnän lisäämisaika, viesti ja merkinnän tekijä.
- Omistaja voi poistaa kohokohtamerkintöjä.
- Muut tilit pystyvät lisäämään kohokohtapyyntöjä. He voivat pyytää rahasumman palkkioksi auton omistajalta siitä vaivannäöstä, että he tekivät kohokohtapyynnön.
- Auton omistaja pystyy lähettämään rahaa sopimuksen saldoon omalta tililtään. Kun omistaja hyväksyy kohokohtapyynnön, raha siirtyy sopimuksen saldosta pyynnön tekijälle. Jos sopimuksessa ei ole tarpeeksi saldoa, kohokohtapyyntöä ei voi hyväksyä. Tämä ominaisuus on sovelluksessa Ethereum-alustan turvallisuusrajoitteiden takia.
- Auton omistaja voi nostaa rahaa sopimuksen saldosta. Tätä varten on olemassa funktio sopimuksessa sekä verkkosovelluksessa, mutta käyttöliittymää ei ole toteutettu. Rahan noston voi tehdä selaimen JavaScript-konsolilla.
- Omistaja pystyy hyväksymään kohokohtapyynnön, jolloin se lisätään auton kohokohtien joukkoon. Samalla sopimuksen saldosta siirtyy pyydetty summa kohokohtapyynnön tekijälle.
- Omistaja voi kieltäytyä kohokohtapyynnöstä, jolloin se poistetaan.
- Muu kuin auton omistaja pystyy tekemään tarjouksen autosta.
- Tarjouksen tekijä voi myös poistaa tarjouksensa, jos hän muuttaa mieltään.
- Auton omistaja voi kieltäytyä tarjouksesta, jolloin se poistuu tarjousten listalta.
- Omistaja pystyy hyväksymään tarjouksen, jolloin sopimus siirtyy ”omistajuus on vaihtumassa” -tilaan. Tässä tilassa sovelluksen etusivuilla näkyy tiedote omistajuuden siirtymisestä kaikille käyttäjille. Omistajuuden vaihtumistilassa sekä auton ostaja, että myyjä voivat perua kaupan.

- Kun auton omistajuus on vaihtumassa, auton ostaja näkee painikkeen, jolla hän pystyy saamaan sopimuksen omistusoikeudet. Ostaja voi painaa tästä painikkeesta, kun auto on toimitettu hänelle.
- Painikkeen painamisen jälkeen uusi omistaja saa kaikki käyttöoikeudet, mitä edellisellä omistajalla oli. Edellinen omistaja näkee samat ominaisuudet kuin muutkin tavalliset käyttäjät.
- Lähes kaikki yllä mainittujen toimintojen teko tallentaa sopimukseen historiatietomerkinnän (Solidity-kielessä ”event”). Merkintöjä tallentuu myös silloin, kun tietoja poistetaan. Historiatietomerkintöjä voi tarkastella kuka tahansa, eikä niitä voi poistaa kukaan.

6.2 Toteutumattomia käyttötapauksia

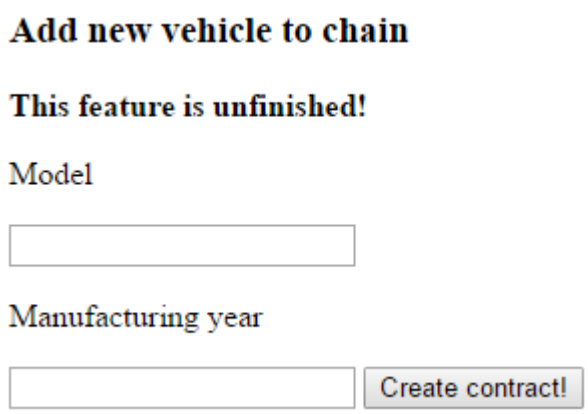
Varhaisessa työn vaiheessa suunniteltiin roskapostin estojärjestelmä kohokohtapyynnöille. Nykyisessä sovelluksessa yksikin käyttäjä pystyisi lähettämään tuhansia kohokohtapyyntöjä päivässä. Tätä varten kehitettiin työn aikana ominaisuus, jota voisi luonnehtia kohokohtien pyyntöoikeuksiksi (highlight request rights). Tavoitteena oli kehittää järjestelmä, jossa auton omistaja voisi antaa eri käyttäjille oikeudet lisätä kohokohtapyyntöjä. Ominaisuus olisi hyödyllinen esimerkiksi julkisuuden henkilölle, jonka auto on kirjattu sovellukseen. Julkinen auto saa paljon huomiota sosiaalisessa mediassa, jolloin roskaposti voi olla riesa auton omistajalle. Vähän huomiota saavan auton omistaja voisi laittaa asetuksen pois päältä, jolloin kuka tahansa käyttäjä voi lähettää kohokohtapyyntöjä. Yhdessä ohjelman versiossa tätä ominaisuutta kehitettiin, mutta kehitys lopetettiin, koska päätettiin, ettei käyttötapauksen esittely ollut tarpeeksi tärkeää työn tarkoituksen kannalta. Tämä työ toimii vain yhdellä tietokoneella, mutta mikäli sopimus lähetettäisiin varsinaiseen Ethereum-verkkoon, tämä ominaisuus olisi hyödyllinen.

Toinen työn alkuvaiheissa suunniteltu käyttötapaus on rahasummista tinkiminen. Olisi mahdollista koodata sopimukseen logiikkaa, jossa auton omistaja voisi tinkiä kohokohtapyynnöstä pyydettyä summaa. Tinkimisen voisi lisätä myös autokaupassa tarjouksen tekijän ja auton omistajan välille. Tästä käyttötapauksesta luovuttiin, koska se on työläs kehittää. Käyttötapaus vaatisi hyvin suunnitellun, hyvin testatun, virheet-

tömän koodin älykkääseen sopimukseen. Maksutapahtumia käsitellessä raha saattaa kadota ohjelmointivirheen seurauksena. Ominaisuus vaatisi lisäksi selkeän käyttöliittymän Meteor-sovellukseen, jotta käyttäjät ymmärtävät, missä vaiheessa rahanvaihto ja tinkimien ovat.

Kolmas toteuttamaton käyttötapaus on historiatietojen suodattaminen ja järjestely. Lopullisessa sovelluksessa kaikki historiatiedot ovat samassa listassa peräjälkeen. Olisi käytettävyyden kannalta parempi, jos listaan voisi lisätä suotimia ja siitä voisi tehdä hakuja.

Neljäs on uusien ajoneuvojen lisääminen sovelluksella. Sovellukseen olisi pystynyt lisäämään lomakkeella uuden auton, jolle tehdään uusi sopimus lohkoketjuun. Tätä varten tehtiin käyttöliittymä, jonka pystyy näkemään lopullisessa sovelluksessa. Käyttöliittymää pääsee katsomaan menemällä sivulle http://localhost:3000/ käynnistetyssä sovelluksessa. Toiminnallisuus jätettiin tekemättä koodissa olevien ongelmien takia. Toteutus vaatisi Meteor-sovelluksen JavaScript-koodin uudelleen suunnittelua. (Kuva 5.)



KUVA 5. Lomake, jolla uuden sopimuksen voisi luoda.

Viidentenä suunniteltiin, että huoltotiedot voisi lisätä lomakkeella kohokohtaan. Yhdestä kohokohdasta voisi tehdä tyypiltään huoltotieto-tyyppisen kohokohdan, jolloin kohokohdassa olisi tekstin ja päivämäärän lisäksi myös lisätietoja tehdystä huollosta. Tätä ominaisuutta suunniteltiin, mutta huomattiin, että se olisi ollut liian haastava toteuttaa. Solidity-kielessä on puutteita muuttuvan pituisille tietorakenteille, kuten taulu-

koille. Toteutettuna tämä käyttötapaus olisi hyvin vakuuttavalla tavalla esitellyt älykkäiden sopimusten tuomaa hyötyä liikennöinnin alalla. Lopulta kumminkin todettiin, ettei käyttötapaus ollut elintärkeä työn tarkoituksen kannalta, eikä sitä kehitetty. Huoltotiedot suunniteltiin koodattavan aluksi vain viitenä huoltotehtävänä, joille kullekin voisi merkata huollon lopputuloksen. Huollon toimitukselle voisi merkata yhden neljästä tilasta: ei tarkistettu (NotChecked), tarkistettu (Checked), tarkistettu ja vika löydetty (CheckedAndFoundFault), vika löydettu ja korjattu (FoundFaultAndFixed):

```
enum MaintenanceOutcome {
          NotChecked, Checked, CheckedAndFoundFault,
          FoundFaultAndFixed
}
struct MaintenanceTasks {
          MaintenanceOutcome engine;
          MaintenanceOutcome tires;
          MaintenanceOutcome pedals;
          MaintenanceOutcome brakes;
          MaintenanceOutcome interior;
}
```

Nykyinen järjestelmä vaatii auton ostajaa lähettämään koko tarjoamansa summan sopimukseen. Tämä tehdään tarjouksen tekohetkellä. Kymmenien tai satojen tuhansien eurojen ostossa on epäkäytännöllistä kiinnittää niin suuria rahasummia pitkäksi aikaa. Olisi suositeltavaa, jos sekä ostaja että myyjä lähettäisivät pienen osuuden kaupasta – esimerkiksi 5 % – auton hinnasta sopimukseen. Auton ostaja lähettäisi rahat tarjouspyyntöä tehtäessä ja myyjä lähettäisi tarjouspyynnön hyväksyessä. Kun molemmilla osapuolilla on rahaa kiinnitettynä sopimukseen, molemmilla on vähemmän motiiveja yrittää petosta autokaupan aikana. Auton myyjä saisi koko kaupan rahat, kun ostaja on hyväksynyt kaupan (painamalla painiketta ”Olen saanut ajoneuvon. Ota sopimuksen omistajuus”).

Viimeisenä mainittakoon, että nykyisessä sovelluksessa sama käyttäjä voi tehdä useamman tarjouksen. Olisi parempi, jos yksi käyttäjä voisi tehdä vain yhden tarjouksen, jolloin uudempi tarjous korvaa vanhan.

7 ARVIO TEKNIIKASTA

Truffle on varmasti hyödyllinen projektissa, jossa on tuhansia rivejä Solidity-koodia sekä suurempi kehitysryhmä, jossa ainakin yksi kehittäjä voi kuluttaa aikaa testien koodaamiseen. Tämän laajuisessa työssä Truffle-sovelluskehyksestä saatu hyöty jäi vähäiseksi.

Meteorilla työskentely oli nopeaa ja sujuvaa. Tässä sovelluksessa sekä kaikissa muissakin Ethereum-sovelluksissa Meteorin reaktiivisuus on hyvin hyödyllinen. Lohkoketjussa tapahtuvat muutokset näkyvät sovelluksen käyttöliittymässä välittömästi. Meteor sopii hyvin Ethereum-sovellusten kehittämiseen.

Solidity-kieli jätti toivomisen varaa. Kielessä on paljon teknisiä rajoitteita. Osa rajoitteista johtuu ohjelmointikielen vähäisestä kehityksestä. Tämä työ tehtiin Solidity-kielen versiolla 0.4.6. Solidity-kieli on hyvin helppokäyttöinen, mutta sitä käyttävien kehittäjien on hyvä olla tietoisia rajoitteista, joita kielessä on. Rajoitteista tietäminen säästää kehitysaikaa ja vähentää turhan työn ja kokeilujen määrää.

Solidity-kielessä ei voi hakea koko mapping-tietorakenteen kaikkia alustettuja alkioita. Kehittäjän täytyy itse kirjoittaa funktiot koko mapping-tietorakenteen alkioiden haulle. Tämä tulee hidastamaan ja vaikeuttamaan hajautettujen sovellusten kehitystä, niin kauan kuin se on olemassa.

Tuotantokäytössä olevassa älykkäässä sopimuksessa tietueita voi olla satoja tai tuhansia. Yhden tietueen hakeva funktio on niin hidas suorittaa, että tässä työssä toteutettu ratkaisu ei ole käytännöllinen tuotantokäytössä olevalle älykkäälle sopimukselle.

Vaikka tietorakenne olisikin suorituskyvyltään hidas, olisi se silti tärkeä ominaisuus, koska se lisäisi innovaatioiden määrää Ethereum-sovelluksien kehittäjien keskuudessa. Älykkäiden sopimuksien kehittäjien yhteisössä innovaatiot ovat hyvin tärkeitä Ethereum-alustan tulevaisuuden kannalta.

Solidity-kielellä ei voi kutsua vieraassa sopimuksessa olevaa funktiota, joka palauttaa string-tyyppisen muuttujan. Yhdessä vaiheessa työtä tehtiin ohjelmistosuunnitelma,

jossa pyrittiin tekemään kohokohdista omia sopimuksiaan. Kohokohtasopimuksista pääsopimus voisi lukea tietoja. Suunnitelma hylättiin tässä mainitun teknisen rajoitteen takia. Älykkään sopimuksen koodi olisi ollut yksinkertaisempaa ja helpompi jatkokehittää, jos rajoitetta ei olisi ollut.

Solidityssä ei ole tukea useille sopimuksen konstruktoreille. On mahdollista suunnitella ohjelmansa eri tavalla siten, että kehittäjä voi sopeutua tähän rajoitteeseen. Joissakin ohjelmissa tämä voi tehdä koodista epäselvää ja vaikeasti luettavaa.

On huomattava puute, että Solidity-koodissa ei ole null-käsitettä. Monissa ohjelmissa kehittäjä haluaisi tehdä funktioita, joka palauttaa arvon, kun funktion kutsu onnistuu, mutta palauttaa arvon null silloin, kun kutsu ei onnistu. Null-käsite on myös hyödyllinen virheenkäsittelyssä, kun halutaan tarkistaa, onko muuttujia alustettu.

Vaikka rajoitteet kuulostavat pieniltä, yhdessä ne rajoittavat kaikkia käyttötapauksia, mitä Ethereum-alustalla voi toteuttaa. Rajoitteet tulevat hidastamaan Ethereumin läpimurtoa yleiseen, maailmanlaajuiseen käyttöön.

8 LOPPUSANAT

Työssä oli tavoitteena luoda käytännön sovellus, joka havainnollisesti esittelee, miten tulevaisuudessa voitaisiin hyödyntää älykkäitä sopimuksia. Tätä varten tutustuttiin Ethereum-alustan ohjelmistokehitykseen.

Ethereum-alustaa varten on kehitetty monia helppokäyttöisiä kehitystyökaluja. Solidity-kielen ominaisuuksissa on parannettavaa. Rajoitteet hidastavat kehitystä ja rajoittavat innovaatioita, joita Ethereumin älykkäillä sopimuksilla voi kehittää. Solidity-kieli on silti helppokäyttöinen ja se on helppo oppia. Meteor on hyvin helppokäyttöinen sovelluskehys. Se soveltuu hyvin Ethereum-alustaan pohjautuviin sovelluksien kehittämiseen.

Lohkoketjutekniikan tutkijat puhuvat, että historia voi jakaantua ennen lohkoketjuja olleeseen ja lohkoketjujen jälkeiseen maailmaan, että lohkoketjut ovat keksintönä yhtä merkityksellinen kuin internet (24). Elämmekö tulevaisuudessa maailmassa, jossa raha, tavaroiden ja palvelujen korvaukset liikkuvat ihmiseltä ihmiselle, ilman välikäsiä, kaikki käyttäen lohkoketjualustalla toimivia älykkäitä sopimuksia? Jää nähtäväksi, tuleeko ennustuksesta totta.

LÄHTEET

1. Nakamoto, Satoshi 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Saatavissa: https://bitcoin.org/bitcoin.pdf. Hakupäivä 25.9.2017.
2. Future Thinkers Podcast. Vitalik Buterin: What is Ethereum and How to Build a Decentralized Future. Saatavissa: http://futurethinkers.org/vitalik-buterin-ethereum-decentralized-future/. Hakupäivä 22.11.2017.
3. GO-Science 2016. Block chain technology. Saatavissa: https://www.youtube.com/watch?v=4sm5LNqL5j0. Hakupäivä 22.11.2017.
4. Swan, Melanie 2015. Blockchain. Kappale: Ethereum: Turing-Complete Virtual Machine. O'Reilly Media, Inc. Saatavissa: Safari Books Online (vaatii käyttöoikeuden). Hakupäivä 22.11.2017.
5. Prusty, Narayan 2017. Building Blockchain Projects. Packt Publishing. Saatavissa: Safari Books Online (vaatii käyttöoikeuden). Hakupäivä 22.11.2017
6. Buterin, Vitalik 2015. Bitcoin 2.0 - Ideas and Applications. Bitcoinist.net. Luento. Saatavissa videolla: https://youtu.be/Fjhe0MVRHO4?t=1m54s. Hakupäivä 22.11.2017.
7. Buterin, Vitalik 2016. DEVCON1: Understanding the Ethereum Blockchain Protocol. Ethereum. Luento. Saatavissa videolla: https://youtu.be/gjwr-7PgpN8. Hakupäivä 27.10.2017.
8. Ethereum 2017. Solidity Docs - Introduction to Smart Contracts. Read the Docs, Inc. Saatavissa: https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html. Hakupäivä 25.11.2017.
9. Computerphile 2016. Turing Complete. Video. Saatavissa: https://www.youtube.com/watch?v=RPQD7-AOjMI. Hakupäivä 25.09.2017.
10. Ethereum 2017. Ether, The crypto-fuel for the Ethereum network. Saatavissa: https://ethereum.org/ether. Hakupäivä 22.11.2017.
11. Ethereum 2017. Web3 JavaScript app API. GitHub, Inc. Saatavissa: https://github.com/ethereum/wiki/wiki/JavaScript-API. Hakupäivä 22.11.2017.
12. Ethereum-yhteisö 2017. Remix - Solidity IDE. Saatavissa: https://remix.ethereum.org/. Hakupäivä 25.11.2017.
13. Ethereum-yhteisö 2017. Ganache CLI. GitHub, Inc. Saatavissa: https://github.com/trufflesuite/ganache-cli. Hakupäivä 22.11.2017.

14. Ethereum-yhteisö 2017. Ganache CLI Wiki - Installing TestRPC on Windows. GitHub, Inc. Saatavissa: https://github.com/trufflesuite/ganache-cli/wiki/Installing-TestRPC-on-Windows. Hakupäivä 25.11.2017.
15. Visual Studio. Microsoft 2017. Saatavissa: https://www.visualstudio.com/downloads/. Hakupäivä 22.11.2017.
16. Consensys 2017. Truffle - Your Ethereum Swiss Army Knife. Saatavissa: http://truffleframework.com/. Hakupäivä 22.11.2017.
17. Meteor Development Group Inc. 2017. Meteor - The fastest way to build JavaScript apps. Saatavissa: https://www.meteor.com/. Hakupäivä 22.11.2017.
18. Ethereum-yhteisö 2016. Why are there multiple Ethereum clients? Read the Docs, Inc. Saatavissa: http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html. Hakupäivä 22.11.2017.
19. Ethereum-yhteisö 2016. Ethereum Clients, go-ethereum. Read the Docs, Inc. Saatavissa: http://ethdocs.org/en/latest/ethereum-clients/go-ethereum/index.html. Hakupäivä 22.11.2017.
20. Ethereum Foundation 2017. Smart money, smart wallet. Saatavissa: https://ethereum.org/. Hakupäivä 22.11.2017.
21. Miettinen, Lauri 2017. Classic Car Chain. GitHub, Inc. Työn GitHub-repositorio. Saatavissa: https://github.com/Miettine/ClassicCarChain. Hakupäivä 22.11.2017.
22. Ethereum-yhteisö 2017. Frequently asked questions - Can you return an array or a string from a solidity function call? Read the Docs, Inc. Saatavissa: http://solidity.readthedocs.io/en/develop/frequently-asked-questions.html#can-you-return-an-array-or-a-string-from-a-solidity-function-call. Hakupäivä 22.11.2017.
23. Ethereum-yhteisö 2017. Dapp styles. GitHub, Inc. Saatavissa: https://github.com/ethereum/dapp-styles. Hakupäivä 22.11.2017.
24. Gasteiger, Daniel 2016. Block chain demystified. TEDx Talks. Saatavissa: https://www.youtube.com/watch?v=40ikEV6xGg4. Hakupäivä 22.11.2017.

```
pragma solidity ^0.4.6;

contract ClassicCarChain {

          address public vehicleOwner;

          string public vehicleModel;

          uint public originBlockNumber;

          uint public vehicleManufacturingYear;

          uint public acceptedOfferAmount=0;

          bool public ownershipBeingTransferred = false;

          uint public contractBalance = 0;
          function Withdraw(uint _amount) OnlyByOwner public returns
(bool){
                    if(msg.sender.send(_amount)){
                              return true;
                    }
                    return false;
          }

          address public upcomingOwner;

          function ClassicCarChain(string _model, uint _year) {
                    vehicleOwner = msg.sender;
                    //The one who created this contract to the net-
work becomes the first vehicle owner.

                    originBlockNumber = block.number;
                    vehicleModel = _model;
                    vehicleManufacturingYear = _year;
                    ownershipBeingTransferred = false;
          }

          /// This index is used as an identifier of Highlights. It
is incremented whenever a new highlight request is made.
          uint public highlightIndex=0;

          modifier OnlyByOwner() {
                    if (msg.sender == vehicleOwner) {
                              _;
                    }
          }

          modifier NotByOwner() {
                    if (msg.sender != vehicleOwner) {
                              _;
                    }
          }
```

```
          modifier OnlyByUpcomingOwner() {
                    if (msg.sender == upcomingOwner) {
                              _;
                    }
          }

          modifier OnlyByOwnerOrUpcomingOwner() {
                    if (msg.sender == upcomingOwner || msg.sender ==
vehicleOwner) {
                              _;
                    }
          }

          modifier OnlyIfOwnershipBeingTransferred(){
                    if (ownershipBeingTransferred) {
                              _;
                    }
          }

          modifier OnlyIfOwnershipNotBeingTransferred(){
                    if (!ownershipBeingTransferred) {
                              _;
                    }
          }

          function BeginOwnershipChange(address _upcomingOwner, uint
_amount) private OnlyIfOwnershipNotBeingTransferred {
                    upcomingOwner = _upcomingOwner;
                    acceptedOfferAmount= _amount;
                    ownershipBeingTransferred=true;

          }

          function CancelOwnershipChange() public OnlyByOwnerOrUp-
comingOwner OnlyIfOwnershipBeingTransferred{

        if (upcomingOwner.send(acceptedOfferAmount)){
            acceptedOfferAmount=0;
                    upcomingOwner=0;
                              ownershipBeingTransferred=false;
                    }
          }

          function AcceptOwnershipChange() public OnlyByUp-
comingOwner OnlyIfOwnershipBeingTransferred {

        if (vehicleOwner.send(acceptedOfferAmount)){
          GiveVehicleOwnership(upcomingOwner);
                    acceptedOfferAmount=0;
                    upcomingOwner=0;
                              ownershipBeingTransferred=false;

                    }
```

```
        }

        ////////////////////////////////////////

        mapping(uint => CCClib.Highlight) private highlights;

        ////////////////////////////////////////

        //mapping(uint => address)  allOffers ;
        mapping(uint => CCClib.Offer) private offers;

        uint public offerIndex = 0;

        function GetOffer(uint _index) public returns (bool
_initialized,address _maker, uint _amount){

            CCClib.Offer memory foundOffer = offers[_index];

            _initialized = foundOffer.initialized;
            _maker = foundOffer.maker;
            _amount = foundOffer.amount;
        }

        event EOfferRemoved(address maker, uint amount, uint
dateTime);
        event EOfferRejected(address maker, uint amount, uint
dateTime);

        function RemoveOrRejectOffer(uint _index) public returns
(bool){

            CCClib.Offer memory foundOffer = offers[_index];

            address sender = msg.sender;

            bool senderIsVehicleOwner = sender == vehicleOwner;
            bool senderIsOfferMaker = sender == foundOffer.maker;

            if (senderIsVehicleOwner || senderIsOfferMaker){

                if (foundOffer.maker.send(foundOffer.amount)){

                    delete offers[_index];

                    if (senderIsVehicleOwner) {
                            EOfferRejected(foundOffer.maker,
foundOffer.amount,now);
                    } else {
                                                EOfferRe-
moved(foundOffer.maker, foundOffer.amount,now);
                                        }

                    return true;
                }
```

```
            }
            return false;
        }

        event EOfferAccepted(address maker, uint amount, uint
dateTime);

        function AcceptOffer(uint _index)   OnlyByOwner public re-
turns (bool) {
            CCClib.Offer memory foundOffer = offers[_index];

      if (foundOffer.initialized){


                            delete offers[_index];

                            EOfferAccepted(foundOffer.maker,
foundOffer.amount, now);

                            BeginOwnershipChange(foundOffer.maker,
foundOffer.amount);
                            return true;

      }

            return false;
        }

        event EOfferMade(address maker, uint amount, uint
dateTime);

        function MakeOffer() NotByOwner public payable {


                address sender= msg.sender;

                CCClib.Offer memory newOffer =
CCClib.Offer({id:offerIndex,
                initialized:true,
                maker:sender,
                amount:msg.value});

                offers[offerIndex]=newOffer;

                offerIndex++;
                EOfferMade(msg.sender, msg.value, now);
        }


        ////////////////////////////////////////

        function AddNewToHighlights(CCClib.Highlight _h) private {

            highlights[_h.id]=_h;
```

```
        highlightIndex += 1;
        }

        function GetHighlight(uint _id) public returns (
                  uint _highlightType,

                  bool _initialized,

                  address _maker,
                  uint _requestCreationDateTime,
                  uint _reward,
                  string _message,

                  bool _approvedToChain,
                  bool _madeByOwner,
                  uint _additionToChainDateTime
                  ) {

                  CCClib.Highlight h = highlights[_id];

                  _highlightType = h.highlightType;

                  _initialized = h.initialized;
                  _maker = h.maker;
                  _requestCreationDateTime =
h.requestCreationDateTime;
                  _reward = h.reward;
                  _message = h.message;

                  _approvedToChain= h.approvedToChain;
                  _madeByOwner = h.madeByOwner;
                  _additionToChainDateTime =
h.additionToChainDateTime;
        }


        function AddHighlightAsOwner (string _message) OnlyByOwn-
er() public {

                  CCClib.Highlight memory h =
CCClib.NewHighlight(highlightIndex, _message);

                  AddNewToHighlights(h);

                  EmitEvent_HighlightSavedToChain(h);

        }

        function MakeHighlightRequest(uint _reward, string
_message) NotByOwner() public {

        CCClib.Highlight memory h = CCClib.NewHighlightRequest
(highlightIndex, _reward, _message);
```

```
                    AddNewToHighlights(h);

                    EmitEvent_HighlightRequestMade(h);


          }

          function DeleteExistingHighlight(uint _id, string
_reasonForDeletion) OnlyByOwner() public returns (bool) {
                    //Deleting a key in a mapping replaces the
struct of that
                    //key with a struct posessing default-values.

                    EmitEvent_HighlightDeleted(highlights[_id],
_reasonForDeletion);

                    delete highlights[_id];
          }

          function RejectHighlightRequest(uint _id) OnlyByOwner()  {


          EmitEvent_HighlightRequestRejected(highlights[_id]);

                    delete highlights[_id];
          }

          function AcceptHighlightRequest(uint _id)  OnlyByOwner()
payable public returns (bool)   {
                    //TODO: Find out if this function needs to have
the payable-keyword.
                    //Is there some security restriction, that a
contract cannot send funds if
                    // the message sender doesn't send them?

                    // Check if the owner actually has enough money.

               CCClib.Highlight handledRequest = highlights[_id];

                    // Send the money to the maker

                    contractBalance += msg.value;
                    uint requestedReward = handledRequest.reward;
                    uint difference = msg.value-requestedReward;

                    if ( handledRequest.maker.send(msg.value)) {

                         //
//CCClib.PromoteHighlightRequest(highlights[_id]);
                         //Disabled, because suspecting that this
causes problems

                              handledRequest.approvedToChain=true;
```

```
                                handledRe-
quest.additionToChainDateTime=now;


          EmitEvent_HighlightSavedToChain(highlights[_id]);

                                return true;
                    }

                    return false;
          }

          function GiveVehicleOwnership(address _newOwner) private {
                    if (_newOwner!=vehicleOwner){
                                address oldOwner = vehicleOwner;

                                EVehicleOwnershipPassed( oldOwner,
_newOwner, now);
                                // The now-keyword returns the current
block timestamp, as soon as this transaction finds its way into a
mined block.
                                // I remember hearing that in the real
Ethreum network, blocks are mined each 10 minutes. The timestamp is
quite accurate.
                                vehicleOwner = _newOwner;
                    }
          }

/// Events


          event EHighlightRequestMade(
                    uint highlightId,
                    address maker,
                    uint requestCreationDateTime,
                    uint requestedReward,
                    string message
          );

          function EmitEvent_HighlightRequestMade (CCClib.Highlight
_h) private {
                    EHighlightRequestMade(_h.id, _h.maker,
_h.requestCreationDateTime, _h.reward, _h.message);
          }

          event EHighlightSavedToChain(
                    uint highlightId,
                    address maker,
                    uint requestCreationDateTime,
                    uint paidReward,
                    string message,

                    bool madeByOwner,
                    uint additionToChainDateTime
```

```
        );

        function EmitEvent_HighlightSavedToChain (CCClib.Highlight
_h) private {
                    EHighlightSavedToChain(_h.id, _h.maker,
_h.requestCreationDateTime, _h.reward, _h.message, _h.madeByOwner,
_h.additionToChainDateTime);
        }

///

        event EHighlightRequestRejected(
                    uint rejectionDateTime,

                    uint highlightId,
                    address maker,
                    uint requestCreationDateTime,
                    uint requestedReward,
                    string message
                    );

        function EmitEvent_HighlightRequestRejected
(CCClib.Highlight _h) private{
                    EHighlightRequestRejected(now, _h.id, _h.maker,
_h.requestCreationDateTime, _h.reward, _h.message);
        }

        event EHighlightDeleted(
                    uint deletionDateTime,
                    string reasonForDeletion,

                    uint highlightId,
                    address maker,
                    uint requestCreationDateTime,
                    uint paidReward,
                    string message,
                    bool madeByOwner,
                    uint additionToChainDateTime
                    );

        function EmitEvent_HighlightDeleted (CCClib.Highlight _h,
string _reasonForDeletion) private{
                    EHighlightDeleted(now, _reasonForDeletion,
_h.id, _h.maker, _h.requestCreationDateTime, _h.reward, _h.message,
_h.madeByOwner, _h.additionToChainDateTime);
        }

        event EVehicleOwnershipPassed(address oldOwner, address
newOwner, uint dateTime);


}
```

```
library CCClib {


          struct Offer{
              uint id;
              bool initialized;
              address maker;
              uint amount;
          }


          struct Highlight{
                    uint id;
                    bool initialized;
                    uint highlightType;

                    address maker;
                    uint requestCreationDateTime;
                    uint reward;
                    string message;

                    bool approvedToChain;
                    bool madeByOwner;
                    uint additionToChainDateTime;

                    MaintenanceTasks maintenanceData;
          }

          struct MaintenanceTasks {
              bool engine;
              bool tires;
          bool pedals;
                    bool brakes;
                    bool interior;
          }

          function CreateMaintenanceData  (bool[] _status) internal
returns (MaintenanceTasks) {

                    if (_status.length != 5){
                              throw;
                    }

        MaintenanceTasks memory tasks = MaintenanceTasks({
            engine:_status[0],
            tires:_status[1],
            pedals:_status[2],
            brakes:_status[3],
            interior:_status[4]
        });

        return tasks;

          }
```

```
          enum HighlightTypes {
                    Review,
                    Maintenance
          }


          //https://ntgroup.studio.crasman.fi/pub/web/vianor/pdf/Via
nor_perushuolto_plus.pdf


          function GetMaintenanceStatus(Highlight _h) internal re-
turns (
              bool _eng,
              bool _tires,
              bool _pedals,
              bool _brakes,
              bool _interior
              ){

        MaintenanceTasks memory data = _h.maintenanceData;
                  _eng = data.engine;
                  _eng = data.tires;
                  _eng = data.pedals;
                  _eng = data.brakes;
                  _eng = data.interior;

          }

          function  NewHighlightRequest  (uint _id, uint
_reward,string _message)internal returns ( Highlight){
                    Highlight memory h;

                     h.id=_id;
                     h.initialized=true;
                     h.highlightType = uint(HighlightTypes.Review);

                     h.maker= msg.sender;
                     h.requestCreationDateTime=now;
                     h.reward=_reward;
                     h.message=_message;

                     h.approvedToChain=false;
                     h.madeByOwner=false;
                    return h;

          }

          function NewHighlight (uint _id, string _message) internal
returns ( Highlight){

                    Highlight memory h;

                    h.id=_id;
```

```
                    h.initialized=true;
                    h.highlightType = uint(HighlightTypes.Review);

                    h.maker= msg.sender;
                    h.requestCreationDateTime=now;
                    h.reward=0;
                    h.message=_message;

                    h.approvedToChain=true;
                    h.madeByOwner=true;
                    h.additionToChainDateTime = now;

                    return h;

          }


          function NewMaintenanceHighlightRequest  (uint _id, uint
_reward,string _message, uint[] _maints, uint[] _status) internal
returns ( Highlight){

                    Highlight memory h = NewHighlightRequest(_id,
_reward, _message);

                    h.highlightType=
uint(HighlightTypes.Maintenance);

                    //h.maintenanceData = CreateMaintenanceData(
_status) ;
                    return h;
          }

          function NewMaintenanceHighlight (uint _id, string
_message, uint[] _maints, uint[] _status) internal returns ( High-
light){

                    Highlight memory h = NewHighlight(_id,
_message);

                    h.highlightType=
uint(HighlightTypes.Maintenance);

                    //h.maintenanceData = CreateMaintenanceDa-
ta(_status);

                    return h;
          }

    function PromoteHighlightRequest ( Highlight storage h) internal
{
          h.approvedToChain=true;
                    h.additionToChainDateTime=now;
    }
}
```