Lauri Miettinen

**KLASSISTEN AUTOJEN ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

**KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

Lauri Miettinen

Opinnäytetyö

Kevät 2017

Kone- ja tuotantotekniikan koulutusohjelma

Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu

Koulutusohjelma, suuntautumisvaihtoehto

Tekijä(t):

Opinnäytetyön nimi:

Työn ohjaaja(t):

Työn valmistumislukukausi ja -vuosi: Sivumäärä:60 + 5 liitettä

Tiivistelmä on suppea, itsenäinen esitys, joka antaa lukijalle kuvan tehdystä työstä. Se kirjoitetaan sitten, kun työ on valmis. Tiivistelmä kirjoitetaan täydellisin virkkein ja passiivimuodossa. Siinä selostetaan lyhyesti työn aihe, tavoitteet, työn keskeisin toteutustapa ja olennaiset tulokset.

Sisältö jakaantuu yleensä kolmeen kappaleeseen:

1. työn aihe ja tavoitteet

2. menetelmät, toteutustapa ja eteneminen

3. tulokset ja esitetyt johtopäätökset.

Tiivistelmän tulee mahtua yhdelle sivulle rivivälillä 1.

Korvaa nämä tekstit omillasi.

Asiasanat: x, x, x

(Kirjoita 3–7 kpl työtäsi kuvaavaa asiasanaa*.* Käytä hyväksesi esimerkiksi yleistä suomalaista asiasanastoa YSA, <http://onki.fi/fi/browser/overview/ysa>.)

ABSTRACT

Oulu University of Applied Sciences

Degree programme, option

Author(s):

Title of thesis:

Supervisor(s):

Term and year when the thesis was submitted: Pages: 60 + 5 appendices

Vieraskielinen tiivistelmä (Abstract) tehdään työn tilaajan niin vaatiessa tai osaston käytänteiden mukaisesti. Sen ei tarvitse olla suora käännös suomenkielisestä tiivistelmästä, vaan sen voi kirjoittaa kansainväliselle lukijalle.

Remember to include the following in the abstract:

Why – background for the thesis, assigner

What – objective of the thesis

How – background knowledge applied, research methods, material used

Achievements – main results

Exploitability – conclusions, development proposals, further measures

The length of the abstract is 1 page.

Keywords: x, x, x

(List 3–7 keywords that describe your thesis. Use for example keywords from the following controlled vocabulary thesauri:

MeSH <http://www.yso.fi/onto/mesh/conceptscheme>

Agriforest <http://www-db.helsinki.fi/agri/agrisanasto/Welcome_eng.html>

Helecon <http://helecon3.hkkk.fi/helevoc/?lang=eng&dbname=MIX>)

ALKULAUSE

Raportissa voi olla alkusanat, joissa kiitetään toimeksiantajaa, ohjaajia ja mahdollisesti muita tukijoita tai avustajia. Alkusanat voi sisältää lyhyesti myös muuta työhön liittyvää, lukijaa kiinnostavaa tietoa, joka ei muuten tule raportissa esiin. Alkusanat päivätään ja allekirjoitetaan.

Korvaa nämä tekstit omillasi.

SISÄLLYS

Contents

[1 JOHDANTO 8](#_Toc483218772)

[2 SUUNNITTELU JA KÄYTTÖTAPAUKSET 10](#_Toc483218773)

[2.1 Lohkoketjun hyöty sovelluksessa 11](#_Toc483218774)

[3 OHJELMISTO 12](#_Toc483218775)

[3.1 Työn kannalta oleellinen ohjelmisto 12](#_Toc483218776)

[3.1.1 Ethereum-alusta 12](#_Toc483218777)

[3.1.2 Ethereum-virtuaalikone 12](#_Toc483218778)

[3.1.3 Solidity-kääntäjä 12](#_Toc483218779)

[3.1.4 Web3-kirjasto 13](#_Toc483218780)

[3.2 Ohjelmointityössä käytetty ohjelmisto 13](#_Toc483218781)

[3.2.1 Browser Solidity 13](#_Toc483218782)

[3.2.2 TestRPC 13](#_Toc483218783)

[3.2.3 Truffle-sovelluskehys 14](#_Toc483218784)

[3.2.4 Meteor 14](#_Toc483218785)

[3.3 Muu työhön liittyvä ohjelmisto 14](#_Toc483218786)

[3.3.1 GoEthereum (geth) 14](#_Toc483218787)

[3.3.2 Ethereum Wallet 15](#_Toc483218788)

[4 TYÖN KULKU 16](#_Toc483218789)

[5 LOPPUTULOKSET 18](#_Toc483218790)

[5.1 Toteutuneet käyttötapaukset 18](#_Toc483218791)

[5.2 Toteutumattomia käyttötapauksia 19](#_Toc483218792)

[6 ARVIO TEKNIIKASTA 21](#_Toc483218793)

[7 LOPPUSANAT 24](#_Toc483218794)

SANASTO

Jos työssä toistuvat samat lyhenteet tai suureiden, yksiköiden tai piirrosten merkinnät ja jos niitä on paljon, niistä kootaan luettelo selityksineen. Otsikkona voi olla sisällön mukaan esimerkiksi LYHENTEET, MERKKIEN SELITYKSET tai SANASTO. Luettelo laaditaan aakkosjärjestykseen. Aakkostuksessa noudatetaan SFS-standardia 4600 ja SI-järjestelmää.

Posta tämä sivu, jos et tarvitse.

# JOHDANTO

Pääasia ensin, sitten avataan?

Lohkoketjutekniikka sai alkunsa vuonna 2009, kun Satoshi Nakamoto kirjoitti paperin, jossa hän ehdotti tietomallin puolueettoman valuutan luomisesta. Nykyään valuutta liikkuu pankin tai muun laitoksen kautta. Bitcoinin idea oli luoda valuutta, josta ei ole vastuussa kukaan keskitetty taho, jossa maksajan ei tarvitse luottaa keneenkään. Nakamoto suunnitteli hajautetun järjestelmän rahan lähettämiselle ja maksutapahtumien todentamiselle. (1)

Bitcoin- maksut tallennetaan hajautettuun tietokantaan, jota kutsutaan lohkoketjuksi. Kukaan 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 laajan hajautetun ohjelmointikielen lohkoketjuille, mutta päättikin tehdä yksinkertaisemman järjestelmän, sillä ymmärsi tekniikan olevan kokeellinen ja haastava luonteeltaan. (2.)

Siitä Vitalik Buterin keksi tehdä Ethereumin, ohjelmointikielen ja sovellusalustan, jonka sovellukset toimivat lohkoketjun avulla (3.) Kenties joku parempi lähde tähän. Ehkä se alkuperäinen Buterinin paperi?

Lohkoketju on arvokkaan tiedon tallentamiseen perustuva järjestelmä. Bitcoinin tapauksessa arvokas tieto on varallisuus ja valuutta. Klassiset autot ovat arvokkaita. Niistä voidaan käydä satojen tuhansien, ja joskus jopa miljoonien eurojen kauppoja. Autoharrastajat kokevat, että autot ovat maansa kulttuurille merkityksellisiä, sillä kussakin klassisessa autossa on viitteitä aikansa kulttuurista.

Klassiset autot liittyvät myös liikennöinnin alaan. Liikennöinnin alallakin on paljon tulevaisuuden sovelluksia, joissa voitaisiin hyödyntää lohkoketjuja. Koska tämän työn aihe liittyy oleellisesti myös liikennöintiin, tässä työssä saadut tulokset pätevät myös liikennöinnin alalla.

Autoja huoltaessa huoltotietojen tallennus olisi haviteltava ominaisuus. Sekä tavallisille autoille, että klassisille autoille. Tavallisilla autoilla huoltotietoja voivat hyödyntää vakuutusyhtiöt, 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 (kilometrimittari) tallennettaisiin lohkoketjuun, auton omistaja voisi vakuuttaa huutokaupoissa ostajan autonsa arvosta.

Nykyään älykkäät sopimukset eivät vielä ole lyöneet läpi maailmanlaajuisille markkinoille. Lohkoketjutekniikkaa tutkitaan vielä yliopistoissa. Älykkäitä sopimuksia ei juurikaan käytetä kaupallisesti. (viite)

Tässä työssä pohditaan liiketoimintamalli ja käyttötapaukset lohkoketjutekniikkaa hyödyntävälle sovellukselle. Tavoite oli luoda Ethereum -älykäs sopimus, jota voitaisiin hyödyntää klassisen auton elinkaaren seurantaan.

Toinen osa työtä oli luoda web-käyttöliittymä, jolla lohkoketjuun tehtäviä merkintöjä voi tehdä ja tarkastella. Tässä työssä kuvaillaan työn suunnittelusta, totetuksesta ja lopputuloksesta.

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.

# SUUNNITTELU JA KÄYTTÖTAPAUKSET

Aliotsikko!

Auton omistajalla on käytössään auton elinkaarta seuraava älykäs sopimus.

Yhdelle autolle tehdään yksi älykäs sopimus.

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

Ajatuksena oli tehdä yleisiä merkintöjä. Merkinnöistä käytetään nimeä kohokohta, eli highlight.

Ulkopuoliset käyttäjät voivat tehdä verkkosovelluksella kohokohtapyyntöjä (highlight request) älykkääseen sopimukseen. Kohokohtapyyntönä voi olla esimerkiksi auton huoltaminen korjaamolla tai asiantuntija-arvio auton autenttisuudesta alkuperälleen. Kohokohtapyynnön tekijä voi pyytää pienen rahallisen korvauksen vaivannäöstä. Kohokohtapyynnössä on lomake, jossa on mm tekstikenttä. Esimerkiksi auton korjaava mekaanikko voisi tehdä kohokohtapyynnön, jossa lukee ”Korjasin tämän auton Välivainion Bensiksellä.”, ja pyytää lomakkeen täyttämisen vaivannäöstä viisi euroa. Lomakkeen täyttämiseen kuluu muutama minuutti, mutta sekin vaatii pientä vaivannäköä. Ihmiset ovat suostuvaisempia muutaman minuutin vaivannäköön, jos siitä saa pienen rahallisen korvauksen.

Merkintöjen tavoite on vakuuttaa tuleva ostaja auton arvokkuudesta. Kukin kohokohtapyyntö maksaisi auton omistajalle useita euroja tai kymmeniäkin euroja per merkintä. Ne silti maksavat itsensä takaisin moninkertaisesti, mikäli auton ostaja päättää tarjota huutokaupassa suuren summan autosta, johtuen todennetusta arvosta, jonka merkinnät tuovat autolle.

Lisäksi kuka tahansa ihminen maailmassa voisi tehdä tarjouksen autosta. Kaikki tarjoukset ovat julkisia, ja kaikki pystyvät tarkastelemaan niitä. Auton omistaja voi hyväksyä tarjouksen, jolloin hän voi toimittaa auton ostajalle. Sopimuksen omistajuusoikeus siirtyy autokaupan yhteydessä.

## Lohkoketjun hyöty sovelluksessa

Mitä hyötyä on juuri lohkoketjusovelluksella? Miksei tätä voisi tehdä perinteisenä sovelluksena, joka toimii tavallisella palvelimella, ja jonka merkinnät tallennetaan tietokantaan?

Yksi etu lohkoketjussa on tietysti se, että lohkoketjusovelluksessa ei ole lainkaan palvelinkustannuksia. Tavallisessa palvelussa palvelinkustannukset täytyy tienata takaisin liiketoiminnalla, kuten mainoksilla. Tämä palvelu on luonteeltaan sellainen, että sitä ei välttämättä käytetä kovin usein, jolloin liiketoiminnan tekeminen voi olla haastavaa.

Toinen etu on lohkoketjumerkintöjen pitkäikäisyys. Klassisen auton elinkaari voi olla vuosikymmeniä, eivätkä kovin monet yritykset ole olemassa niin kauaa. Jos tavallinen palvelu lopettaa toimintansa, tietokannassa olevat merkinnät voivat kadota. Lohkoketju on paljon pitkäikäisempi, ja koska kopiot tiedoista ovat läsnä jokaisella ketjua käyttävällä tietokoneella, tiedot eivät edes 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. Maineen menetyksen pelko voi olla suuri. Mikäli tällainen sovellus on oikeasti käytössä tulevaisuudessa, autonomistajat varmasti ymmärtävät, että on kannattavaa pysyä rehellisenä lisättäessä kohokohtia.

# OHJELMISTO

Laita kaikkiin jokin viite mistä sen ohjelman saa ladattua

## Työn kannalta oleellinen ohjelmisto

Kuvaillaan työn kannalta oleellista ohjelmistotekniikkaa.

### Ethereum-alusta

Ethereumiin kuuluu lohkoketju, joka toimii tietokantana. Alustaan kuuluu virtuaalikone, joka on älykkäiden sopimusten ajoympäristö. Ethereum-alusta luo vertaisverkon, jossa käyttäjän koneella käynnissä oleva virtuaalikone on yhteydessä muualla maailmassa oleviin Ethereum-virtuaalikoneisiin. Ethereum-verkko koostuu maailmanlaajuisesti yhteen liittyneiden Ethereum-solmujen vertaisverkosta. Jos Ethereum-verkkoa haluaa käyttää, tulee omalle tietokoneelle käynnistää Ethereum-solmu, esimerkiksi GoEthereum (ks. 3.3.1).

### Ethereum-virtuaalikone

Ethereum-virtuaalikone sisältää Ethereum-koodiajoympäristön. Ethereum-alustan alkuaikoina joku äijä teki teknisen spesifikaation siitä, mitä käskyjä Ethereum-virtuaalikoneen kuuluu toteuttaa (Linkkaa spesifikaatioon. Löytyyköhän sitä mistään? Jos ei löydy, niin sano sori.). Täten Ethereum-virtuaalikoneen voi kuka tahansa toteuttaa, lähes millä tahansa ohjelmointikielellä. Virtuaalikone on tehty esimerkiksi Go-kielellä (ks. GoEthereum), sekä Pythonilla.

### Solidity-kääntäjä

Solidity-kääntäjä kääntää Solidity-kielen Ethereum-tavukoodiksi, jonka Ethereum-virtuaalikone muuntaa konekäskyiksi. Solidity-kieli on hyvin yksinkertainen ja helposti opittava. Se muistuttaa Javascriptiä. Ohjelmoijayhteisö on kehittänyt monia Solidity-kääntäjiä. Yksi kääntäjä on Browser Solidity (ks. 3.2.1). Kääntäjiä on myös Truffle-sovelluskehyksessä sekä Ethereum Wallet -ohjelmassa.

### 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. Kirjasto on hyvin dokumentoitu esimerkkeineen Web3:n GitHubin wiki-sivuilla.

## Ohjelmointityössä käytetty ohjelmisto

Tässä kuvaillaan ohjelmistoa, jota päädyttiin käyttämään lopullisessa työssä

### Browser Solidity

Browser solidity on selaimessa käytettävä kääntäjä. Javascriptin suorituskyvystä johtuen browser solidity voi olla hidas. Sen etuna on, että se toimii kaikilla tietokoneilla ilman ylimääräisten ohjelmien asennusta.

### TestRPC

TestRPC on node-palvelinsovellus, joka on tehty mukailemaan hyvin tarkasti Ethereum-lohkoketjun toimintaa.

Maksutapahtumien todentaminen tapahtuu testrpc:ssä hyvin nopeasti.

TestRPC:n asentaminen on helppoa linuxilla, käyttäen Node pakettihallintaohjelmaa (Node package manager, npm). Windowsilla asentaminen on vaikeampaa. Windowsille asentaessa on kannattavaa lukea testrpc-repositorion lueminut (readme) -tiedostossa olevia ohjeita. Kaikki Windows-koneelle tarvittavat resurssit on mahdollista asentaa ilmaisen Microsoft Visual Studio Community Edition -asennuksen yhteydessä.

### Truffle-sovelluskehys

Truffle on Ethereum -älykkäiden sopimusten kehittämistä varten luotu sovelluskehys, jonka on tarkoitus helpottaa Solidity-kielellä älykkäiden sopimusten koodin kirjoittamista. Sovelluskehyksen avulla voi tehdä älykkäiden sopimusten jatkuvaa integraatiota Ethereum-lohkoketjuun. Trufflen avulla voi myös koodata automatisoituja testejä sopimuksille.

### Meteor

Meteor on sovelluskehys, joka yhdistää node-palvelimen mongodb -tietokannan hyvin helppokäyttöisesti. Meteor on saanut maailmalla huomiota, koska sen avulla voi tehdä samanaikaisesti verkkosivuja sekä mobiilisovelluksia. Meteoriin on helppo ohjelmoida reaktiivisia käyttöliittymiä, eli tehdä sovelluksesta sellainen, että sen käyttöliittymä näyttää reaaliaikaisesti kaikki päivitykset mitä tulevat tietokantaan. Meteor-sovelluskehyksellä on X seuraajaa githubissa, tehden siitä Y:nneksi suosituimman repositorion.

## Muu työhön liittyvä ohjelmisto

### GoEthereum (geth)

Go-kielellä ohjelmoitu Ethereum-virtuaalikone. Tätä 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.

### Ethereum Wallet

Valitettavasti Ethereum Wallet ei toimi testrpc:n kanssa, koska testrpc toteuttaa vain osan kaikista rajapinnoista, joita geth-solmussa on. EthereumWallet käyttää rajapintaa signAndSendTransaction, mitä ei ole toteutettu testrpc:ssä.

Ethereum Walletissa on Solidity-kääntäjä, jonka suorituskyky on parempi kuin browser solidityssä. Tätä varten Ethereum Wallet voi olla hyödyllinen.

# TYÖN KULKU

Aloituspalaverin pitämisen jälkeen aloitettiin pohtimaan työn tarkoitusta. Ohjelmistotyön idea keksittiin, siitä tehtiin vaatimusmäärittely, jonka työn tilaaja hyväksyi.

Tutustuttiin Ethereumiin käyttämällä Ethereum Wallet -ohjelmaa. Pystytettiin geth-testiverkko tietokoneelle. ClassicCarChain-älykkään sopimuksen koodaus aloitettiin.

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

Truffle-sovelluskehyksen käyttö aloitettiin. Kokeiltiin automaattisten testejä. Automaattisten testien koodaus oli kumminkin hyvin työlästä, joten niiden kehittämisestä luovuttiin myöhemmin. Työn lopun ajan Trufflea käytettiin lähettämään (deploy) sopimus lohkoketjuun.

Meteor-sovelluskehyksen käyttö aloitettiin. Tehtiin Meteorilla Ethereumiin liittymättömiä harjoitustöitä, jotta kehittämisen perusteet tulisivat tutuiksi.

Älykkään sopimuksen ja Meteor-sovelluksen välisessä rajapinnassa alkoi ilmetä ongelmia, johtuen Solidity-kielen ja virtuaalikoneen teknisistä rajoitteista. Älykästä sopimusta uudelleen koodattiin ja eri ratkaisuja kokeiltiin.

Vastaan tuli ongelma käyttötapauksessa ”vierailijan pitää kyetä näkemään kaikki kohokohdat, mitä autolla on”, sillä Solidity-kielessä ei ole tietorakennetta, jonka kaikki alkiot voisi hakea yhtä aikaa web3-rajapinnan kautta Meteor-käyttöliittymään. Ongelma ratkaistiin tekemällä meteoriin kiertokyselyrakenne, joka hakee yksitellen kaikki alkiot sopimuksesta.

Hyviä ohjelmistosuunnitteluperiaatteita pyrittiin seuraamaan suunnittelemalla sopimus uudelleen siten, että jokainen kohokohtamerkintä olisi Ethereum-lohkoketjussa oma alisopimuksensa, joihin ClassicCarChain-pääsopimus tekisi kutsuja. Yhteyden todentaminen onnistui, eli funktiokutsuja alisopimuksessa pystyy tekemään ainoastaan tietty sopimus. Suunnitelmasta jouduttiin kumminkin 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).

Lopuksi tehtiin käyttöliittymää. Ethereum-yhteisön kehittämää dappstyles.styl-tyyliä kokeiltiin. Työssä päädyttiin tekemään oma yksinkertainen css-tyyli.

# LOPPUTULOKSET

Kuvassa X on auton omistajan näkymä sovelluksen etusivuilta. Yläpalkissa on sovelluksessa navigointilinkit, oikeassa yläkulmassa aktiivisen tilin vaihtaminen. Yläpalkin alapuolella lomake kohokohdan lisäämistä varten. Sininen laatikko on yksi kohokohta, jossa auton omistaja näkee poistamispainikkeen, sekä tekstikentän, johon voi kirjoittaa syyn merkinnän poistamiselle.



Kuva X. Lopullinen sovellus kohokohtasivulta.

## Toteutuneet käyttötapaukset

* 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.
* Autentikaatio tapahtuu lohkoketjussa. Kuka tahansa voisi esimerkiksi kutsua verkkosivuilla funktiota, jolla saisi auton omistajuuden (gainOwnership). Tällöin lohkoketjussa 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 pienen summan rahaa palkkioksi auton omistajalta siitä vaivannäöstä, että he tekivät kohokohtapyynnön.
* Omistaja pystyy hyväksymään kohokohtapyynnön, jolloin se lisätään auton kohokohtien joukkoon. Samalla omistajalta 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 muuttaa mieltään.
* Auton omistaja voi kieltäytyä tarjouksesta, jolloin se poistuu tarjousten listalta.
* Omistaja pystyy hyväksymään tarjouksen, jolloin sopimus siirtyy ”omistajuus on vaihtumassa” -tilaan.
* Kun auton omistajuus on vaihtumassa, auton ostaja näkee painikkeen, jolla hän pystyy saamaan sopimuksen omistusoikeudet. Ajatuksena on, että ostaja 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.
* Jokainen yllä mainittu toiminto tallentaa sopimukseen historiatietomerkinnän (Solidity-kielessä ”event”). Merkintöjä tallentuu myös silloin kun tietueita poistetaan. Historiatietomerkintöjä voi tarkastella kuka tahansa, eikä niitä voi poistaa kukaan.

## 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. Epäsuositun 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 demonstroiminen ollut tarpeeksi tärkeää työn tarkoituksen kannalta. Tämä työ toimii vain yhdellä tietokoneella, mutta mikäli sopimus lähetettäisiin varsinaiseen Ethereum-verkkoon, tämä käyttötapaus voisi olla hyödyllistä olla totetutettuna.

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, virheettömän koodin älykkääseen sopimukseen. Maksutapahtumia käsitellessä, raha saattaa kadota ohjelmointivirheen seurauksena. Vaatisi lisäksi selkeän käyttöliittymän Meteor-sovellukseen, jotta käyttäjä ymmärtää, 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 Ethereum-verkkoon. Sovellukseen olisi pystynyt lisäämään lomakkeella uuden auton, jolle tehdään uusi sopimus lohkoketjuun. Tätä varten tehtiin käyttöliittymä, jonka pystyy näkemään lopullisessa sovelluksessa. Toiminnallisuus jätettiin tekemättä. Ominaisuuden toteutus vaatisi Meteor-sovelluksen Javascript-koodin tarkempaa suunnittelua. (Kuva X)



Kuva X. Lomake, jolla uuden sopimuksen voisi lähettää.

Viidentenä suunniteltiin, että huoltotiedot voisi lisätä lomakkeella kohokohtaan. Tämä oli haastava toteuttaa, sillä Solidityssä tuki muuttuvan pituisille tietorakenteille, kuten taulukoille, on ongelmallinen. Tämä käyttötapaus olisi hyvin vakuuttavalla tavalla voinut vakuuttaa älykkäiden sopimusten hyödystä liikennöinnin alalla. Lopulta kumminkin todettiin, ettei käyttötapaus ollut elintärkeä työn tarkoituksen kannalta, eikä sitä toteutettu. Huoltotiedot suunniteltiin koodattavan aluksi vain viitenä huoltotehtävänä, joille kullekin voisi merkata huollon lopputuloksen:

**struct MaintenanceTasks {**

**MaintenaceOutcome engine;**

**MaintenaceOutcome tires;**

**MaintenaceOutcome pedals;**

**MaintenaceOutcome brakes;**

**MaintenaceOutcome interior;**

**}**

**enum MaintenaceOutcome {**

**NotChecked, Checked, CheckedAndFoundFault,**

**FoundFaultAndFixed**

**}**

Kuudentena on käyttötapaus, jota olisi syytä parannella. Lopullisessa sovelluksessa auton myynnissä ja ostossa on joitakin tietoturva-aukkoja. Nykyinen järjestelmä vaatii 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. Auton myyjä saisi koko kaupan rahat, kun ostaja on hyväksynyt kaupan (Painamalla painiketta ”Olen saanut ajoneuvon. Ota sopimuksen omistajuus”)

Viimeisenä mainitakoon, että nykyisessä järjestelmässä on mahdollista saman käyttäjän tehdä useampi tarjous. Olisi parempi, jos yksi käyttäjä voisi tehdä yhden tarjouksen, jolloin uudempi tarjous ylikirjoittaa vanhan.

# ARVIO TEKNIIKASTA

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

Meteorilla työskentely oli nopeaa ja sujuvaa. Meteorissa hyvä ominaisuus on javascript-koodien yhdistäminen ja universaalisuus koko sovelluksessa. Mallipohjat ja mallipohjien apurifunktiot ovat hyvin monipuolisia.

Tässä sovelluksessa, sekä kaikissa muissakin Ethereum-sovelluksissa Meteorin reaktiivisuus on hyvin hyödyllinen. Sovelluksessa lohkoketjussa tapahtuvat muutokset näkyvät käyttöliittymässä välittömästi.

Ainoa ongelma Meteor-kehityksessä oli hidas alkuunpääsy Ethereum-integraation kanssa. Internetissä on melko vähän opiskelumateriaalia.

Solidity-kieli jätti toivomisen varaa. Kielessä on paljon teknisiä rajoitteita. Osa rajoitteista johtuu ohjelmointikielen vähäisestä kehityksestä. Solidity on tällä hetkellä versiossa 0.4.7. Osa solidity-kielen rajoitteista johtuu Ethereum-virtuaalikoneen rajoitteista. Solidity-kieli on hyvin monipuolinen ja 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 on ollut menneissä versioissa turvallisuusongelmia. Turvallisuusongelmat ovat vaarantaneet koko Ethereum-alustan (4). On mahdollista, että turvallisuusongelmia ilmenee tulevaisuudessakin. Koska Ethereum-verkossa liikkuu rahaliikennettä, turvallisuusongelmat ovat huomattava ongelma alustan tulevaisuudelle.

Ei voi hakea koko mapping-tietorakenteen kaikkia alustettuja alkioita. Kehittäjän täytyy itse kirjoittaa funktiot koko mappingin alkoiden haulle. Tämä tulee hidastamaan huomattavasti hajautettujen sovellusten kehitystä, niin kauan kuin se on olemassa.

Tässä työssä tehtiin Meteor-sovelluksessa javascript-koodia, joka hakee yksitellen kaikki alkiot mapping-tietorakenteesta. Sen ohjelmoiminen ei ole vaikeaa, mutta sen koodaaminen virheettömästi on silti aikaa vievää. Vaikka mapping-tietorakenteessa on teknisiä rajoitteita, Solidity-kieleen olisi mahdollista kehittää tietorakenne, joka laajentaa mapping-tietorakenteen toiminnallisuutta, ja jossa tässä mainitut ongelmat voisivat olla korjattu. Vaikka tietorakenne olisikin suorituskyvyltään hidas, olisi se silti tärkeä ominaisuus, koska se lisäisi innovaatioiden määrää Ethereum-sovelluksien kehittäjien keskuudessa. Innovaatiot ovat hyvin tärkeitä Ethereumille.

Ei voi kutsua vieraassa sopimuksessa olevaa funktiota, joka palauttaa string-tyyppisen muuttujan. Yhdessä vaiheessa tehtiin ohjelmistosuunnitelma, jossa pyrittiin tekemään kohokohdista omia sopimuksiaan. Kohokohtasopimuksista pääsopimus voisi lukea tietoja. Suunnitelma hylättiin tässä mainitun teknisen rajoitteen takia. String-tyyppisten tietueiden näyttäminen oli ehdoton tämän suunnitelman kannalta.

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

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

Vaikka rajoitteet kuulostavat pieniltä, ne kumminkin rajoittavat suuresti kaikkia mahdollisia käyttötapauksia, mitä Ethereum-alustalla voisi muuten toteuttaa. Rajoitteet tulevat hidastamaan Ethereumin läpimurtoa yleiseen, maailmanlaajuiseen käyttöön. Tästä huolimatta, Soldity-kieltä kehitetään parhaillaan, ja joitakin hyvin oleellisia parannuksia on tehty viime aikoinakin, kuten (etsi se joku foorumi missä ihmiset toivoivat jotain ominaisuutta ja linkkaa dokumentaatiossa johonkin paikkaan, missä todistat, että ominaisuus on jo tehty).

Voi olla kannattavaa kertoa EtherSimistä, ja kertoa kuinka käsittämättömän hämmentävää on pysyä mukana Ethereum-ohjelmistokehityksen nykytilanteesta. Aina kaikkea uutta tehdään ja jotain heitetään roskakoriin. Vaikea löytää mikä on uusinta tietoa. Voit laittaa esimerkiksi sen uhkapelisovelluksen, jossa kehittäjä turhautui Mist/Ethereum Wallet sekoiluun, ja dumppasi koko sovelluksen.

Silti kukaan ei voi kieltää Solidity-kielen mahdollisuuksia. Kieli on niin yksinkertainen, ja kuka tahansa jolla on ymmärrystä ohjelmoinnista voi oppia sen. Kielen helppokäyttöisyydellä ja yksinkertaisuudella on suuri merkitys Ethereum-alustan maailmanlaajuisessa läpimurrossa.

# LOPPUSANAT

Työn aihe tähän

Lopputulos tiiviisti

Lohkoketjutekniikan tutkijat puhuvat, että historia voi jakaantua ennen lohkoketjuja olleeseen, ja lohkoketjujen jälkeiseen maailmaan, että lohkoketjut ovat keksintönä yhtä mullistava kuin internet.

Väite kuulostaa hurjalta, mutta on hyvinkin mahdollista toteutua rajoissa. Jotkut lohkoketjutekniikan tutkijat väittävät, että lohkoketju on yhtä merkityksellinen keksintö kuin internet.

Jää nähtäväksi, tuleeko ennustuksesta totta. Elämmekö tulevaisuudessa maailmassa, jossa raha, tavaroiden ja palvelujen korvaukset liikkuvat ihmiseltä ihmiselle, ilman välikäsiä, kaikki käyttäen älykkäitä sopimuksia lohkoketjualustalla?

LÄHteet

1. <https://bitcoin.org/bitcoin.pdf>
2. <https://www.youtube.com/watch?v=3PdO7zVqOwc>
3. <http://futurethinkers.org/vitalik-buterin-ethereum-decentralized-future/>
4. <https://blog.ethereum.org/2016/11/01/security-alert-solidity-variables-can-overwritten-storage/>

LIITTEET

Liite 1 Lähtötietomuistio

**Tekijä** Lauri Miettinen

**Tilaaja** Oulun Ammattikorkeakoulu, tekniikan laitos

**Tilaajan yhteyshenkilö(t) ja yhteystiedot**

Janne Kumpuoja ([Janne.Kumpuoja@oamk.fi](mailto:Janne.Kumpuoja@oamk.fi))**,**

Veijo Väisänen (Veijo.Vaisanen@oamk.fi)

**Projektin nimi**

Vintage-auton elinkaaren seuranta lohkoketjutekniikalla -opinnäytetyö

**Projektin tavoitteet**

Tavoite on luoda Ethereum-lohkoketjukehitysalustan avulla älykäs sopimus, jota voitaisiin hyödyntää Vintage-auton elinkaaren seurantaan. Tavoitteena on tehdä web-käyttöliittymä, jonka avulla käyttäjä voi vuorovaikuttaa lohkoketjun kanssa. Työn tuloksista raportoidaan opinnäytetyödokumentissa.

Työssä suunnitellaan ja esitellään liiketoimintamalli, jota valmis tuote voisi käyttää.

Opinnäytetyödokumentissa kuvaillaan luotua järjestelmää ja esitellään sen taustalla oleva tekniikka. Lukijalle esitellään lyhyesti lohkoketjutekniikan perusteet ja historia.

**Projektissa käytettävä prosessimalli (mahdollinen vaihejakoon perustuva aikataulutus)**

28.9.2016: Harjoitustyön vaatimusmäärittelydokumentti oltava valmiina.

21.10.2016: Sprint 1 release. Projektin pohjimmaiset järjestelmät, sovelluskehykset ja palvelimet oltava käyttö- ja kehityskelpoisia.

16.12.2016: Sprint 2 release. Sovelluksesta oltava valmiina versio, joka on esittelykelpoinen (minimum effort). Kirjallisessa työssä oltava kuvauksia projektin etenemisestä.

10.3.2017: Sprint 3 release. Sovelluksen minimituote (minimum viable product) oltava esittelykepoinen.

28.4.2017: Sprint 4 release. Kirjallinen työ, sovellus ja liiketoimintamalli on oltava valmiina. Pidetään lopetuspalaveri.

**Projektissa käytettävät menetelmät ja teknologiat**

Ethereum; scrum-menetelmä; Github –versionhallinta; Meteor, tai muu Javascript-pohjainen sovelluskehys.

**Projektin alustava aikataulu** Työ alkaa 13.9.2016, ja päättyy 30.4.2017