TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kevin Pajula 205845

Plokiahela tehnoloogia analüüs ja arendamine detsentraliseeritud ühisrahastusplatvormi prototüübi näitel ettevõttes Eesti Energia AS

Bakalaureusetöö

Juhendaja: Liisa Jõgiste
magistrikraad

Tallinn 2023

Autorideklaratsioon

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

Autor: Kevin Pajula

17.05.2023

Annotatsioon

2022. aasta sügisel alustati ettevõttes Eesti Energia AS detsentraliseeritud ühisrahastusplatvormi prototüübi arendamist. 16-nädalase praktika lõpuks valmis ainult osaliselt töötav lahendus. Rakenduse kekskseks elemendiks on plokiahel, mis vastutab andmete hoidmise eest ning nutileping, mis vastutab rakenduse toimimise eest. Käesoleva bakalaureusetöö eesmärk oli analüüsida prototüübis kasutatud plokiahela tehnoloogilisi lahendusi ning arendada puuduolev funktsionaalsus, mis muudaks rakenduse töövõime terviklikuks.

Töö autor implementeeris lisaks puuduvale funktsionaalsusele nutilepingus arhitektuurilised muudatused, rakendades tehase ja kloonimise mustrit. Samuti analüüsis ühisrahastuses plokiahela kasutamisest tulenevaid muresid andmete hoiustamisega ning kasutajate anonüümsusega. Töö autor pakkus välja mõned võimalused nende murede lahendamiseks ning ideed prototüübi edasiseks arendamiseks, mis muudaks rakenduse efektiivsemaks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 6 peatükki, 11 joonist, 2 tabelit.

Abstract

Analysis and development of blockchain technology on the example of decentralized crowdfunding platform prototype in Eesti Energia AS

In the fall of 2022, a group of six students started developing a decentralized crowdfunding platform prototype for the company Eesti Energia AS. After a 16-week internship period, the group had a partially functional application, allowing users to create new campaigns and make investments. The key element of the application is blockchain, which is responsible for storing data, and smart contract, which is responsible for the application's operation capability.

The aim of this bachelor's thesis was to analyze blockchain technology used in the application and fully develop the missing functionality, which included refunding money to users, distributing money to campaign creators and investors. In addition to the missing functionality, author also implemented architectural changes to the smart contract by applying factory and cloning patterns.

The use of blockchain in crowdfunding platforms eliminates the need for a third party and offers transaction transparency, but may also create new concerns. Some of them could be anonymity of users, data integrity, storage of data in the blockchain and distribution of the generated income. Author analyzed and proposed some methods that could resolve some of these problems using third-party solutions. Author also proposed ideas to be researched for further development of the application. Database integration with blockchain methodology and tokenization of assets used in crowdfunding projects could make the application more effective.

The thesis is in Estonian and contains 33 pages of text, 6 chapters, 11 figures, 2 tables.

Lühendite ja mõistete sõnastik

Baitkood	Arvuti objekti kood, mis teisendatakse arvuti jaoks loetavaks koodiks
Bitcoin	*Bitcoini* plokiahelas kasutatav krüptovaluuta
CID	*Content Identifier* ehk unikaalne faili identifikaator *IPFS* võrgustikus
Crowdfund	Ühisrahastuse toimimiseks vajalikku koodi sisaldav nutileping
Crowdfund (*master*)	Kloonimise mustris loodav esimene nutileping, millele ülejäänud lepingud oma kutsed edasi delegeerivad
Draft pull request	Algeline koodi liitmistaotlus, millega soovitakse integreerida ühte koodi haru teisega
Eesrakendus	Kasutajaliides
Ethereum	*Ethereum'i* plokiahelas kasutatav krüptovaluuta
Factory	Tehase mustris kasutatav nutileping, mis vastutab uute *Crowdfund* nutilepingute loomise eest
Gaasitasu	*Ethereum'*i plokiahelas teostavate transaktsioonide teenustasu
IPFS	*Interplanetary File System* ehk protokollide komplekt andmete hoidmiseks ja edastamiseks, mis töötab plokiahela põhimõttel
JSON	*JavaScript Object Notation* ehk andmeid organiseeritult hoidev formaat
Kahendmuutuja	Andmetüüp, mille väärtus on tõene või väär
Metamask	Krüptorahakott, mis võimaldab sooritada tehinguid plokiahelaga
Münt	Kirjeldab igasugust vara plokiahelas, millega on kasutajatel võimalik sooritada digitaalseid ülekandeid
Operatsioonikood	Masinale arusaadav juhis, mis määrab soovitava tegevuse sooritamise
Partnervõrk	Võrgustik, kus arvutid on võimelised omavahel suhtlema kasutamata keskset serverit
Proksi	Abstraktne vahelüli, mis võimaldab delegeerimise funktsiooni

Proof of work	Mehhanism plokiahelas, kus osapooled tõestavad riistvara arvutusliku jõu kasutamist
Proof of stake	*Ethereum*'i plokiahelas kasutatav mehhanism, kus kaevandajad pandivad krüptoraha, et tekiks konsensus
REST rakendusliides	*Representational State Transfer* ehk tarkvaraarhitektuuri laad, mis määratleb toimingute komplekti ressursside kohtlemiseks
Räsi	Ühesuunaline tehe andmete krüpteerimiseks kasutades matemaatilisi tuletusi
Silumine	Koodivigade avastamise protsess
Stablecoin	Krüptovaluuta, mille väärtus on seotud mõne teise valuutaga eesmärgiga säilitada stabiilne hind
Sõlm	Osalised *IPFS* võrgustikus, mis hoiustavad andmeid ning võimaldavad võrgustikuga ühendamist
Tagarakendus	Rakenduse tagumine osa, mis vahendab suhtlust kasutajaliidese ja plokiahela vahel
Tekstsõne	Andmetüüp, milles hoitakse teksti
Tokeniseerimine	Protsess plokiahelas, kus varade õigused ja omaniku staatus teisendatakse digitaalseks vormiks
Web3	Kolmas interneti põlvkond, mis on detsentraliseeritud ning rajatud plokiahela tehnoloogiale
Võrguvärav	Vahelüli, mis võimaldab kasutajatele ligipääsu *IPFS* võrgustikus asuvatele failidele

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ühisrahastus on järjest enam populaarsust koguv tehnoloogiline lahendus, mis aitab koguda raha mõne projekti või idee teostamiseks. Traditsiooniline kapitali kaasamine kasutades pangalaenu, aktsiate või võlakirjade emissiooni, võib osutuda väiksemate ettevõtete, idufirmade või indiviidide jaoks keeruliseks. Seetõttu kasutavad sageli just nemad ühisrahastuse võimalust, mis aitab nendel luua või katsetada innovaatilisi lahendusi [1]. Teatud tüüpi projektid, näiteks kinnisvaraarendused, võivad nõuda mitmeid sadu tuhandeid eurosid. Ühisrahastus pakub investoritele võimaluse investeerida sellistesse objektidesse väiksemate summadega, näiteks Eestis tegutsev ühisrahastusplatvorm Estateguru pakub investoritele investeerimisvõimalust alates viiekümnest eurost [2].

Tsentraliseeritud ühisrahastusplatvormid kasutavad rahaliste tehingute sooritamiseks kolmandat osapoolt, näiteks kommertspanga või mõne muu teenusepakkuja makselahendusi. Plokiahela tehnoloogia integreerimine võimaldab kaotada vajaduse kolmanda osapoole kasutamiseks, pakkudes turvalisi ning läbipaistvaid rahaliste ülekannete võimalusi. Plokiahela tehnoloogia alustas maailmas populaarsuse kogumist 2008. aastal, kui Satoshi Nakamoto avaldas *Bitcoin*'i teemalise artikli pealkirjaga „*Bitcoin: A Peer-to-Peer Electronic Cash System*" [3]. *Bitcoin*'i populaarsuse tõttu seostatakse plokiahela tehnoloogiat just *Bitcoin*'i plokiahelaga, kuid ajaga on lisaks tekkinud teisi plokiahelaid, millest kõige populaarsem on *Ethereum*'i plokiahel. *Ethereum*'i plokiahel opereerib kasutades digitaalseid lepinguid, mis on tuntud nutilepingute nime all. Nutilepingut võib defineerida muutmatu üldotstarbelise arvutiprogrammina, mis kajastab programmi ja lõppkasutaja vahelisi tehinguid [4]. Nende kasutamine ühisrahastusplatvormi kontekstis võimaldab luua kogu ühisrahastuse toimimiseks vajaliku funktsionaalsuse, mis lubab sooritada partnervõrgu põhimõttel toimivaid tehinguid.

1.1 Probleem

Nutilepingud leiavad järjest enam kasutust, kui ettevõtted avastavad uusi viise, kuidas kasumlikult ära kasutada tehnoloogia võimalusi ärilahenduste tegemisel. Kui ärilahenduste sihtrühmaks on tavainimesed, siis võivad tekkida mõningad ohud tulenevalt inimeste teadmatusest konkreetse tehnoloogia osas. Sageli on nutilepingud seotud rahaülekannetega, mistõttu on eriti oluline pöörata tähelepanu turvalisusele. Vastasel korral saavad pahatahtlikud ründajad kuritarvitada vigast nutilepingu lähtekoodi ning selle tulemusel võivad inimesed kaotada raha. Üks suurimaid krüptovargusi, *DAO* rünnak, võimaldas kurjategijatel kuritarvitada nutilepingut ja varastada 3,6 miljonit *ethereum*'i krüptoraha, mille tänase päeva hinnanguline väärtus oleks 6 miljardit eurot [5].

Eesti Energias arendatava detsentraliseeritud ühisrahastusplatvormi prototüübi põhiline funktsionaalsus toimub just läbi nutilepingu ning sisaldab ka rahaülekandeid. Eesti Energias ei ole varasemalt loodud ühisrahastusplatvormi prototüüpi plokiahela tehnoloogia abil, mistõttu on põhjalikumalt analüüsimata seonduvad riskid ning head valdkonnapõhised praktikad. Samuti on nutilepingu oluline funktsionaalsus lõpuni välja arendamata, mis keskendub raha tagastamisele ning välja jagamisele kampaania algatajatele ja investoritele.

1.2 Eesmärk

Antud lõputöö eesmärgiks on Eesti Energias arendatava detsentraliseeritud ühisrahastusplatvormi prototüübi näitel analüüsida plokiahela tehnoloogiat ning lõpuni välja arendada puuduolev nutilepingu funktsionaalsus. Autor on tööprotsessi jaotanud alameesmärkideks, mis toetavad eelnevalt mainitud eesmärki.

Jaotatud alameesmärgid on järgnevad:

- analüüsida põhjalikult eelnevalt valminud nutilepingut ning muid valminud plokiahela tehnoloogilisi lahendusi;
- analüüsida plokiahela tehnoloogia levinud kasutamispraktikaid ning -võimalusi prototüübi näitel;

- refaktoorida nutilepingu lähtekood ja muud plokitehnoloogiaga seonduvad lahendused vastavalt teostatud analüüsile;

- realiseerida puuduolev nutilepingu funktsionaalsus vastavalt esitatud nõuetele ning teostatud analüüsile.

1.3 Töö struktuur

Antud bakalaureusetöö peamine sisu jaguneb kuueks suuremaks peatükiks. Sissejuhatus sisaldab üldist tausta, töö probleemi ning eesmärki. Varasema kirjanduse peatükk selgitab plokiahela ja nutilepingu tähendust ning tööpõhimõtet. Metoodikas kirjeldatakse töö raames kasutatavat prototüüpi, tulemuste saavutamiseks kasutatud tööriistasid ning töö- ja arendusprotsessi. Töö autor kirjeldab tulemuste osas prototüübi arendamisele määratud funktsionaalseid ja mittefunktsionaalseid nõudeid, valminud tulemusi, nutilepingute arhitektuuri, nende testimist ning piltide lisamist *IPFS* tehnoloogia abil. Analüüsis hindab töö autor realisatsiooni vastavust nõuetele ning põhjendab nutilepingute arhitektuurilise lahenduse teostust. Samuti kirjeldab plokiahela kasutamisega kaasnevaid murekohti, nende lahendamise võimalusi ning pakub välja alternatiivsed võimalused prototüübi edasiseks arendamiseks. Kokkuvõte sisaldab kogu töö lühikirjeldust.

2 Varasema kirjanduse ülevaade

Antud peatükis tutvustab töö autor detailsemalt plokiahelat, nutilepinguid ning nende tööpõhimõtteid.

2.1 Plokiahel

Plokiahelat võib pidada andmekirjete loendiks, mis töötab detsentraliseeritud digitaalse arvestusraamatuna. See võimaldab kasutajatel salvestada ja jagada ühist vaadet süsteemi olekutest hajutatud võrgus. Plokiahelas jaotatakse andmed plokkideks kasutades kronoloogilist järjestust ning krüptograafiat [6]. Iga plokk salvestab eelneva ploki andmete ja tehingute koopia, mille abil säilib ahela terviklik ajalugu. Plokkide tekkimine sõltub plokiahela tüübist ja selle kaevandamisprotsessist.

Bitcoin'i plokiahelas luuakse plokke konsensuse protokolli põhimõttel *proof of work*. Iga kaevandaja peab lahendama matemaatilise probleemi, ploki räsikoodi (*hash code*) arvutamise, mis sisaldab suvalist arvu „*nonce*". Probleemi on võimalik lahendada ainult kõikide räsikoodi võimaluste proovimisel. Kui esimene kaevandaja on jõudnud probleemi lahenduseni, valideerivad ülejäänud kaevandajad räsikoodi õigsuse ning uus plokk salvestatakse plokiahelasse. Kuna kaevandamisprotsess nõuab rohkelt riistvara jõudlust ja kasutab märkimisväärselt elektrit, jagatakse kaevandajatele vastutasuks krüptoraha. Plokkide ühenduse ja kaevandamisprotsessi keerukuse tõttu on ründajatel praktiliselt võimatu manipuleerida transaktsioonide ajalooga ning muuta kontodel paiknevat saldot [7].

Ethereum'i plokiahel kasutas algselt samuti *proof of work* protokolli, kuid 2022. aastal muudeti see *proof-of-stake* mehhanismile. Kui eelnevas protokollis tõendasid kaevandajad energia kulutamist, siis uuemas protokollis panditakse nutilepingusse krüptoraha. Sellise lahendusega muutub energiakulu väiksemaks ning leeveneb sisenemisbarjäär kaevandamisprotsessi alustamiseks [8]. Rohkemate kaevandajate lisandumisel kasvab plokiahela turvalisus veelgi suuremaks. Siiski on plokiahelaga seotud ka mõned turvariskid. Kaevandajatel, kes ühiselt kontrollivad üle 50% võrgustiku plokkidest, võimaldatakse takistada uute plokkide valideerimist ning varasemate transaktsioonide kustutamist või ümberlükkamist. Samuti on ohuks inimeste krüptorahakoti paroolide kaotamine, mille taastamine ei ole võimalik.

2.2 Nutileping

Nick Szabo, nutilepingute idee autor, kirjeldab nutilepinguid kui arvutipõhiseid tehinguprotokolle, mis täidavad lepingu tingimusi. Nutilepingute eesmärgiks on rahuldada traditsioonilisi lepingutingimusi, näiteks maksetingimused, pant, konfidentsiaalsus ja lepingu täitmine. Samuti minimeerivad nutilepingud nii pahatahtlikud kui ka juhuslikud erandid ning vajaduse usaldusväärsete vahendajate järele [9].

Nutilepingud paiknevad plokiahelas ning omavad unikaalset aadressi. Neid on võimalik rakendada kasutades seda aadressi, mille tagajärjel teostab see iseseisvalt ja automaatselt vajalikud tehingud ettenähtud viisil. Tehingud sooritatakse ahela igas plokis vastavalt andmetele, mis sisestati transaktsiooni käivitamisel. See tähendab, et nutilepinguid rakendavas plokiahelas peab igas plokis asuma virtuaalmasin, mis suudab jooksutada vajalikku funktsionaalsust [10].

Nutilepingud töötavad lubaduste põhimõttel kasutades kahendmuutuja loogikat, mille rakendamisel tekib igal protsessil tõene või väär väärtus. Selle abil on võimalik luua suurel hulgal kontrolle või erinevaid tingimusi, millele sisendandmed või transaktsioonid peavad vastama, et tekiks soovitud tulemus [11]. Nutilepingu suurimaks tugevuseks on võime garanteerida sisestatud koodi täitmine, mis võib osutuda eriti kasulikuks olukorras, kus on vajalik sooritada rahalisi tehinguid.

3 Metoodika

Antud peatükk kirjeldab bakalaureusetöö raames arendatavat prototüüpi ning meetodeid, mida töö autor kasutab eesmärkide täitmiseks. Täpsemalt antakse ülevaade prototüübist, kasutatavatest tööriistadest, tööprotsessist ning bakalaureusetööle eelnenud praktikal valminud funktsionaalsusest.

3.1 Prototüübi kirjeldus

Antud bakalaureusetöö raames käsitletakse prototüübina veebirakendust, mille arendamine sai alguse TalTechi aine „ITB1706 Infosüsteemide arendamise meeskonnaprojekt: tellimus“ raames ettevõttes Eesti Energia AS. Prototüübi arendamine toimus kuueliikmelise meeskonna poolt, kelle üheks osaliseks oli ka töö autor. Kogu arendustegevuse käigus keskendus töö autor eelkõige plokiahela tehnoloogiale.

Prototüübi eesmärgiks oli eksperimenteerida plokiahela tehnoloogiaga, luues ühisrahastusplatvormi roheenergia projektide rahastamiseks. Töötav lahendus võimaldaks potentsiaalsetel investoritel investeerida roheenergia tootmisesse taskukohaste summadega ning muuta potentsiaalsetel roheenergia arendajatel rahastuse kaasamine lihtsamaks. Eesmärgi saavutamine võimaldaks suurendada taastuvenergia kättesaadavust ja populaarsust skaleeritaval tasemel.

3.1.1 Varasemalt valminud funktsionaalsus

Meeskond arendas 16-nädalase praktika jooksul ainult osaliselt töötava prototüübi. Põhilised toimivad funktsionaalsused olid järgnevad (Joonis 1):

- rakendus võimaldas kasutajal end autentida kasutades selleks *Metamask*’i krüptorahakotti;
- kasutajal oli võimalik luua uusi kampaaniaid;
- kasutajal oli võimalik sooritada investeeringuid olemasolevatesse kampaaniatesse kasutades selleks *Metamask*’i krüptorahakotti;

- kasutajal oli võimalik kinnitada plokiahelasse saadetavaid tehinguid kasutades selleks *Metamask*'i krüptorahakotti;
- rakendus võimaldas nutilepingul saata vastav teade transaktsioonide õnnestumise või ebaõnnestumise korral ning kuvada neid kasutajaliideses;
- nutileping hoiustas kõiki loodud kampaaniaid ja tehtud investeeringuid;
- kasutajaliideses oli võimalik näha nii kõiki loodud kampaaniaid kui ka kasutaja poolt loodud kampaaniaid ja tehtud investeeringuid eraldi;
- kogu funktsionaalsus toimis läbi ühe nutilepingu.

Joonis 1. Esialgse rakenduse kasutusjuhtude diagramm

Esialgselt valminud rakenduse arhitektuur jagunes kolmeks osaks: tagarakendus, eesrakendus ja *hardhat*. Tagarakendus sisaldas *REST* rakendusliidese lahendust, mis vahendas päringuid kasutajaliidese ja plokiahela vahel. Eesrakendus hoolitses peamiselt kasutajaliidese eest, kuid sisaldas ka võimalust plokiahelaga suhtlemiseks tagarakenduseta. *Hardhat*'i osa sisaldas nutilepingut, selle teste ning vajalikke konfiguratsioone plokiahela tehnoloogia töötamise jaoks.

3.1.2 Edasiarendus

Praktika järgselt tegi ettevõte arendusmeeskonnale pakkumise jätkata prototüübi arendamisega. Kõik meeskonnaliikmed võtsid pakkumise vastu ning ühiselt asuti planeerima edasist tegevuskava.

Edasiarenduse skoobiks määrati oluliste funktsionaalsuste lõpuni välja arendamine, milleks oli kasutajatele raha tagastamine ning raha välja jagamine nii kampaania algatajatele kui ka investoritele. Samuti soovis ettevõte lisada nutilepingusse kampaaniate kinnitamise funktsionaalsuse, mis võimaldaks rakenduse administraatoril kinnitada või tagasi lükata kasutajate poolt algatatud kampaaniaid. Ettevõttesiseselt arendusmeeskonnale määratud neli mentorit leidsid, et tuleks muuta nutilepingute arhitektuuri ühest nutilepingust mitme eraldiseisva nutilepingu peale, rakendades selleks tehase mustrit. Antud töö keskendub eelkõige nutilepingule ning muudele plokiahelaga seotud tehnoloogilistele lahendustele.

3.2 Kasutatud tööriistad

Plokiahelaga suhtlemiseks oli vajalik rakendada ja kasutada mitmeid erinevaid teeke, programmeerimiskeeli ning rakendusi. Põhilise programmeerimiskeelena on rakenduses kasutatud *JavaScript*'i, kuid nutilepingu kodeerimisel on kasutatud *Solidity* keelt. Tagarakenduses on kasutatud *IPFS* tehnoloogia rakendamiseks selleks vajalikku teeki *ipfs-core*. Projektis on loodud võimalus kasutada *Moralis*'t, mis pakub erinevaid teenuseid, et aidata arendajatel mugavamalt integreerida *Web3* mõne muu tehnoloogiaga [12]. Kasutaja autentimiseks ning plokiahelaga suhtlemiseks kasutatakse *Metamask*'i krüptorahakotti. Eesrakenduses kasutatav *ethers* teek võimaldab suhelda *Ethereum*'i plokiahela ja selle ökosüsteemiga [13]. Rakenduse versioonide haldamiseks on kasutatud *GitHub*'i keskkonda.

3.2.1 Hardhat

Rakenduse üheks oluliseks raamistikuks on *Hardhat*, mis pakub arendajatele plokiahelal põhineva tehnoloogia arendamiseks spetsiaalse töökeskkonna. Sellega on võimalik nutilepinguid kompileerida, siluda, testida, juurutada plokiahelasse ning nendega suhelda [14]. Prototüübis kasutatavad nutilepingud on testitud *Hardhat*'i töökeskkonnas kasutades abiteeki *chai-matchers*.

Prototüübi arendusprotsessi üheks oluliseks komponendiks on lokaalse plokiahela kasutamine. Kui *Ethereum*'i plokiahel on avalikult kasutatav ning tehingud toimuvad *ethereum*'i krüptovaluutaga, millel on seos reaalse valuutaga, siis lokaalne plokiahel töötab konkreetses riistvaras, kus see luuakse. *Hardhat* genereerib lokaalse plokiahela jaoks 20 erinevat kasutajakontot, millel on 10 000 *ethereum*'i krüptoraha. Krüptoraha on võimalik kasutada tehingute tegemiseks, kuid plokiahela sulgemisel kustub kogu protsess.

3.2.2 Remix

Remix on integreeritud programmeerimiskeskkond, mis toetab kogu nutilepingute loomise ja kasutamise tsüklit. *Remix* erineb *Hardhat*'i poolest kasutajamugavusega. Kui *Hardhat*'is tuleb toiminguid realiseerida läbi käsurea, siis *Remix* pakub võimaluse veebis nutilepinguid luua, kompileerida ja nendega suhelda. Töö autor kasutas arendamisel olulisel määral *Remix*'i veebiversiooni, sest see muutis arendusprotsessi mugavamaks ning efektiivsemaks. *Remix* võimaldab kasutada erinevaid plokiahelate keskkondasid, näiteks *Ethereum*'i põhilist plokiahelat, aga ka lokaalselt loodud plokiahelat. Tänu sellele on võimalik rakenduses lokaalselt loodavaid nutilepinguid vaadata ja kontrollida mugavas kasutajaliideses.

3.3 Tööprotsess

Prototüübi arendamisel rakendas arendusmeeskond agiilset arendusmetoodikat *Scrum*, mis aitab arendajatel tööd teha tükkide kaupa ning saada reaalajas tagasisidet tehtavale tööle [15].

Arendustsükkel on võimalik jaotada sprintideks. Ettevõtte prototüübi tooteomanik koos mentoritega kaardistasid prototüübi tööks vajalikud funktsionaalsused. Arendusmeeskond osales iga sprindi alguses tooteomaniku ja mentoritega koosolekul,

kus ühiselt planeeriti järgneva sprindi tegevused ning hinnati nende ajalist kulu. Ajalise kulu hindamise jaoks kasutati *planning poker*'i meetodit, kus iga arendust vajava tegevuse kohta hindasid meeskonnaliikmed individuaalselt ülesande keerukust ning ajalist kulu. Hindamine toimus anonüümselt ning tulemuste avalikustamisel arutlesid meeskonnaliikmed omavahel tekkinud erisuste üle ning seati meeskondlik hinnang. Hinnangu seadmine sel viisil aitas ühtlustada iga meeskonnaliikme arusaama vajalikest ülesannetest ning nende raskusastmest.

Prototüübi arendamisel lähtus arendusmeeskond sprindi planeerimisel kavandatud tegevustest. Iga tööpäeva hommikul toimusid 15-minutilised stand-up koosolekud, kus iga meeskonnaliige pidi vastama kolmele küsimusele:

1) mida tehti eile, et panustada meeskonna tegevusse;

2) mida tehakse täna, et panustada meeskonna tegevusse;

3) kas on mingeid probleeme, mis takistavad töö tegemist?

Koosolekud aitasid hoida kursis nii meeskonda kui ka ettevõtte tooteomanikku koos mentoritega meeskonna progressist ülesannete lahendamisel. See oli ka hea võimalus, et leppida kokku päevane tegevuskava või järgnevad koosolekud.

Iga sprindi lõpus demonstreeris meeskond prototüübi rakendust, mis sisaldas sprindi jooksul tehtud ülesandeid. Ettevõtte tooteomanik koos mentoritega andsid meeskonnale tagasisidet ning juhiseid edasiste paranduste või muudatuste tegemiseks. Samuti toimusid iga sprindi lõpus meeskonnasisesed koosolekud, kus hinnati senist arendusprotsessi ning jagati omavahelist tagasisidet ning soovitusi, mis tõstaks meeskonnatöö kvaliteeti.

Eraldiseisvate ülesannete jaoks tekitati *GitHub*'is eraldi haru, mis keskenduks ainult konkreetse ülesande lahendamisele. Sel viisil said meeskonnaliikmed teha iseseisvalt tööd eraldi harudes. Selleks, et eraldatud harust jõuaks kood põhilisse harusse, tuli läbida koodi ülevaatamise protsess.

Olles ülesande edukalt ära lahendanud, tuli luua *GitHub*'is uus *draft pull request*, mille aktsepteerimisel liideti lisatud kood sihtkohaks valitud harusse. *Draft pull request* pakub mugava koodi ülevaatamise võimaluse, kus ülevaatajatel on võimalik näha tehtud muudatusi, esitada küsimusi, tuua esile koodiridu ning osaleda kommentaariumi

aruteludes. Esmalt oli vajalik koodi ülevaatamine ning aktsepteerimine meeskonnaliikmete poolt. Kui meeskonnaliikmed kiitsid muudatuse heaks, tuli muuta *draft pull request* ametlikuks *pull request*'iks ning lisada ülevaatajate hulka ka neli mentorit. Koodi ülevaatuse läbimiseks oli vaja vähemalt kahe mentori kinnitust vastava muudatuse integreerimiseks.

4 Peamised tulemused

Antud peatükk kirjeldab funktsionaalseid ja mittefunktsionaalseid nõudeid prototüübi rakendusele, valminud funktsionaalsusi, nutilepingu arhitektuuri muutmist, selle testimist ning piltide lisamist plokiahelasse.

4.1 Nõuded

Nõuded baseeruvad ettevõtte tooteomaniku ja mentorite poolt seatud eesmärkidele, mida sooviti saavutada prototüübi arendamisega. Töö autor toob välja ainult sellised nõuded, mis seostuvad nutilepingu või muu plokiahela tehnoloogilise lahendusega. Samuti on töö autor koostöös ettevõttega mõningaid nõudeid korrigeerinud ning vajadusel muutnud või täiendanud.

4.1.1 Funktsionaalsed nõuded

Funktsionaalsete nõuete koostamisel lähtuti põhimõttest, et need defineerivad punktid, mis on vajalikud rakenduse korrapäraseks tööks. Funktsionaalsed nõuded kirjeldavad, kuidas rakendus peab töötama, et täita kasutaja vajadusi.

- Kampaania objekt peab sisaldama kampaania kategooriat, roheenergia projekti tüüpi, kampaania looja eesnime, perenime, e-maili, telefoninumbrit ning omama staatuse väärtust, mis on „*Pending*“, „*Approved*“ või „*Rejected*“.
- Kampaania loomisel peab kampaania staatus saavutama väärtuse „*Pending*“.
- Loodud kampaania, mille staatus on „*Pending*“ väärtusega, ei tohi olla avalikult nähtav, välja arvatud rakenduse administraatorile ning kampaania loojale.
- Rakenduse administraatoril peab olema võimalik määrata „*Pending*“ staatusega kampaaniaid „*Approved*“ või „*Rejected*“ olekusse.
- Rakendus peab võimaldama kasutajatel küsida raha tagasi käimasolevatest kampaaniatest, kuhu ta on varasemalt investeerinud.
- Kasutaja peab raha tagasi küsimisel kinnitama transaktsiooni plokiahelaga ning nõustuma tehingu maksumusega, kasutades selleks *Metamask*’i krüptorahakotti.

- Nutileping peab tagastama vastava teate raha tagasi küsimise õnnestumise või ebaõnnestumise korral.

- Rakenduse administraatoril peab olema võimalik algatada lõppenud ning õnnestunud kampaania raha välja jagamise protsessi.

- Rakenduse administraator peab raha välja jagamiseks kampaania algatajale kinnitama transaktsiooni plokiahelaga ning nõustuma maksumusega, kasutades selleks Metamask'i krüptorahakotti.

- Nutileping peab tagastama vastava teate raha välja jagamise õnnestumise või ebaõnnestumise korral.

4.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsete nõuete koostamisel lähtuti põhimõttest, et need nõuded ei mõjutaks rakenduse funktsionaalsust ning näitaks, kuidas ja millisele kvaliteedile peavad vastama lahendused, mis on realiseeritud funktsionaalsetele nõuete täitmiseks. Samuti kirjeldavad mittefunktsionaalsed nõuded süsteemi piiranguid [16].

- Rakenduse funktsionaalsus peab toimima läbi mitme nutilepingu, kasutades selleks nutilepingute tehase mustrit.

- Rakenduse implementeeritud tehase muster peab olema optimeeritud, kasutades selleks kloonimise mustrit.

- Kogu nutilepingu funktsionaalsus peab olema kaetud ühiktestidega.

- Rakendus peab kasutama spetsiaalset testolukordadeks mõeldud *Goerli* plokiahelat.

- Rakendus peab salvestama kampaania loomisel lisatud pilte, kasutades selleks *IPFS* tehnoloogiat.

- Rakendus peab kasutama plokiahela jälgimistehnoloogiat, mis võimaldab näha plokiahelaga tehtud transaktsioonide ajalugu ning seonduvat detailset informatsiooni.

4.2 Realiseeritud funktsionaalsused

Prototüübi funktsionaalsuse põhiliseks aluseks on nutileping, mis vahendab suhtlust kasutaja ja plokiahela vahel. Töö autor on realiseeritud funktsionaalsuse saavutamiseks leidnud inspiratsiooni ühisrahastuse loogikat sisaldava nutilepingu koostamise õpetusest [17]. Töö autor täiendas edasiarenduse raames nutilepingus kampaania objekti järgnevate atribuutidega:

- *propertyInfo* – sisaldab kampaania kategooriat, asukohta ning roheenergia objekti tüüpi;
- *descriptions* – sisaldab lühikest ja pikka kirjeldust kampaania kohta;
- *creatorInfo* – sisaldab kampaania looja eesnime, perenime, e-maili ning telefoninumbrit;
- *status* – kirjeldab kampaania staatust „*Pending*“, „*Approved*“ või „*Rejected*“ väärtusega.

4.2.1 Raha tagastamine

Nutilepingus on lõpuni välja arendatud raha tagastamiseks mõeldud funktsioon *refund*. Funktsioon kutsutakse välja rakenduse administraatori poolt juhul, kui kampaania ei saavuta rahalist kogumiseesmärki. Funktsiooni kasutamiseks on tagarakenduses loodud lahendus, mis filtreerib nutilepingu logidest kõik kampaania investorid ning seejärel rakendab tsükliga iga investori kohta nutilepingu funktsiooni *refund*. Selle tulemusel saavad kõik investorid oma investeeritud summa tagasi. Funktsiooni õnnestumiseks peab olemasoleva kampaania staatus vastama „*Approved*“ olekule, olemasoleva kampaania kohta ei tohi olla rakendatud rahastuse välja jagamise funktsiooni *claim*, kampaania peab olema lõppenud ning kampaania ei tohi olla saavutanud oma rahalist kogumiseesmärki. Õnnestunud kontrolli läbimisel võetakse funktsiooni sisendist investori krüptorahakoti aadress ning loetakse tema investeeritud summa konkreetses kampaanias. Seejärel nullitakse investeeringuid hoitavas andmestruktuuris tema investeeringute summa ning tagastatakse raha investori kontole. Lõpuks kutsutakse välja *Refund* sündmus, mis tähistab funktsiooni lõppu ning logib plokiahelasse konkreetse lepingu kohta investori aadressi ning tagastatud summa väärtuse (Joonis 2).

```
    function refund(address _investor) external {
        require(msg.sender == owner,
"Message sender is not the owner");
        require(campaign.status == Status.Approved,
"Campaign is not approved");
        require(!campaign.claimed,
"Campaign has already finished");
        require(block.timestamp > campaign.endAt,
"Campaign isn't over");
        require(
            campaign.pledged < campaign.goal,
            "Can't refund if campaign goal was achieved"
        );

        uint256 balance = pledgedAmount[_investor];
        pledgedAmount[_investor] = 0;
        payable(_investor).transfer(balance);

        emit Refund(_investor, balance);
    }
```

Joonis 2. Raha tagastamise funktsioon

4.2.2 Raha välja jagamine kampaania loojale

Teiseks nutilepingus lõpuni välja arendatud funktsiooniks on raha välja jagamise funktsioon *claim*. Seda funktsiooni on võimalik edukalt välja kutsuda ainult konkreetse nutilepingu omanikul ehk kasutajal, kes on selle nutilepingu esmakordselt plokiahelasse juurutanud. Prototüübi mõistes on selleks kasutajaks rakenduse administraator. Funktsiooni kutsumisel kontrollitakse olemasoleva kampaania staatust, mis peab vastama „*Approved*“ väärtusele. Seejärel kontrollitakse, et kampaania ajaline kestvus oleks lõppenud, rahaline eesmärk oleks saavutatud ning konkreetses nutilepingus ei oleks varasemalt õnnestunult välja kutsutud samasugust funktsiooni, st. *claim* funktsiooni saab õnnestunult välja kutsuda ainult ühel korral. Kui kontrollid on läbitud, siis määratakse kampaania objekti *claim* atribuudi väärtus tõeseks ning saadetakse kogutud rahasumma kampaania algataja krüptorahakoti aadressile. Lõpuks kutsutakse välja *Claim* sündmus, mis tähistab funktsiooni lõppu ning logib plokiahelasse konkreetse lepingu kohta kampaania identifikaatori väärtuse (Joonis 3).

```
    function claim() external {
        require(msg.sender == owner,
"Message sender is not the owner");
        require(campaign.status == Status.Approved,
"Campaign is not approved");
        require(block.timestamp >= campaign.endAt,
"Campaign has to be over in order to claim funds");
        require(campaign.pledged >= campaign.goal,
"Campaign has not achieved the goal");
        require(!campaign.claimed, "Funds are already claimed");

        campaign.claimed = true;
        payable(campaign.creator).transfer(campaign.pledged);

        emit Claim(id);
    }
```

Joonis 3. Kampaania loojale raha välja jagamise funktsioon

4.2.3 Raha välja jagamine investoritele

Kampaania investoritele raha välja jagamise eest hoolitseb funktsioon *distribute*. Funktsiooni sisendiks võetakse investori krüptorahakoti aadress ning transaktsiooni sisendist rahaline väärtus, mida soovitakse edastada investorile. Funktsiooni õnnestumiseks peab väljakutsuja olema nutilepingu omanik, kampaania staatus vastama „*Approved*“ olekule ning kampaania *claimed* atribuudi väärtus olema tõene (Joonis 4).

```
    function distribute(address _investor) external payable {
        require(msg.sender == owner,
"Message sender is not the owner");
        require(campaign.status == Status.Approved,
"Campaign is not approved");
        require(campaign.claimed == true,
"Campaign is has not been claimed");

        payable(_investor).transfer(msg.value);
        emit Distribute(_investor, msg.value);
    }
```

Joonis 4. Investoritele raha välja jagamise funktsioon

Nutilepingu funktsioon sooritab korraga ainult ühe ülekande. Tagarakenduses on loodud lahendus, mis loeb nutilepingu logidest konkreetse kampaania tehtud investeeringute andmed. Seejärel rakendatakse tsüklit, kus arvutatakse iga investori kohta tema investeeringu suuruse protsent kogutavast rahalisest eesmärgist ning kutsutakse välja nutilepingust funktsioon *distribute*.

4.2.4 Kampaaniate kinnitamine

Töö autor lisas kampaaniate kinnitamise nõude realiseerimiseks nutilepingusse *enum* tüüpi andmestruktuuri nimega *Status*, mis omab kolme väärtust: „*Pending*“, „*Approved*“ ja „*Rejected*“. Kampaania loomisel määratakse kampaania staatuseks automaatselt „*Pending*“ väärtus. Kampaania staatuse muutmiseks on loodud kaks funktsiooni, millest esimene määrab kampaania staatuse „*Approved*“ ning teine „*Rejected*“ olekusse. Mõlema funktsiooni lõppedes kutsutakse välja sündmus, mis tähistab funktsiooni lõppu ning logib plokiahelasse konkreetse lepingu kohta kampaania identifikaatori väärtuse. Mõlemat funktsiooni on võimalik välja kutsuda ainult konkreetse nutilepingu omaniku poolt (Joonis 5).

```
function approve() external {
   require(msg.sender == owner,
          "Message sender is not the owner");
   campaign.status = Status.Approved;
   emit Approve(id);
}
function reject() external {
   require(msg.sender == owner,
          "Message sender is not the owner");
   campaign.status = Status.Rejected;
   emit Reject(id);
}
```

Joonis 5. Kampaania staatuse kinnitamise ja keeldumise funktsioonid

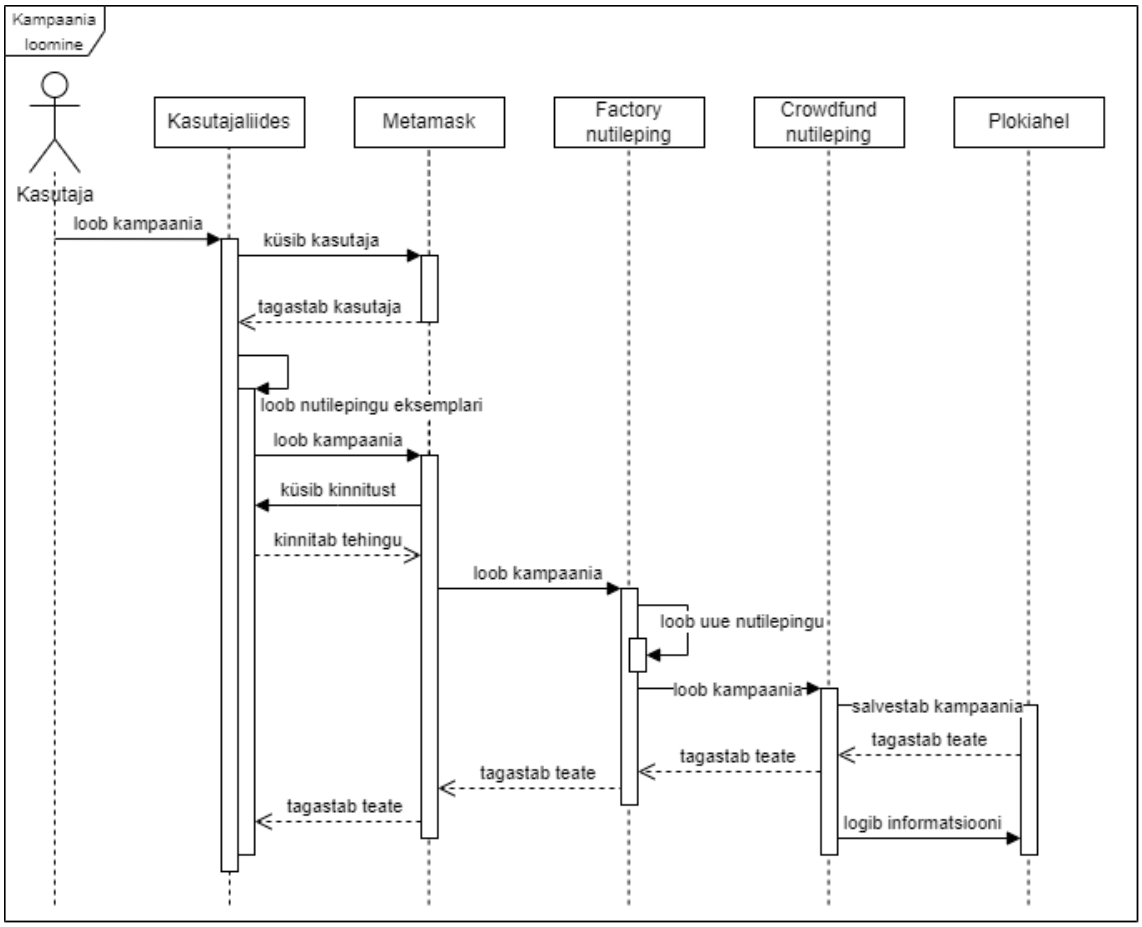
4.3 Nutilepingu arhitektuur

Algselt oli rakenduses kasutusel ainult üks nutileping, mis hoolitses kogu senise funktsionaalsuse eest. Prototüübi võimalikult efektiivse rakendamise huvides oli vajalik muuta nutilepingu arhitektuuri, mis võimaldaks rakenduse skaleerimist.

4.3.1 Tehase muster

Objektorienteeritud programmeerimise maailmas on tehaseks mingi objekt, mis suudab luua teisi objekte [18]. Loodavatele objektidele saab defineerida samade funktsioonide sisu erinevalt, mis on võimalik tänu tehase mustrist tulenevale abstraktsioonile. Nutilepingu puhul kasutatakse tehase mustrit, et tekiks üks eraldiseisev nutileping, mis on võimeline looma ja juurutama uusi nutilepinguid.

Tehase mustri rakendamise tulemusel tekkis kaks erinevat nutilepingut. Esimese nutilepingu, *Factory* (Lisa 2), põhiliseks ülesandeks on luua ja juurutada uusi *Crowdfund* (Lisa 3) tüüpi nutilepinguid. Kui kasutaja soovib luua uut kampaaniat, kutsutakse välja *Factory* nutilepingust funktsioon *launch*, mis loob uue *Crowdfund* tüüpi nutilepingu ning kutsub omakorda loodud nutilepingust välja funktsiooni *launch*. Õnnestunud funktsiooni läbimise korral lisatakse *Factory* nutilepingus loodud *Crowdfund* leping *mapping* andmestruktuuri, mille ülesandeks on hoida kõiki *Crowdfund* nutilepinguid. Samuti kutsutakse *Factory* nutilepingus välja sündmus *Launch*, mis logib *Factory* nutilepingu kohta kampaania loomise informatsiooni (Joonis 6). *Factory* nutileping sisaldab ka *count* atribuuti, mis näitab kõikide *Crowdfund* lepingute summaarset kogust.

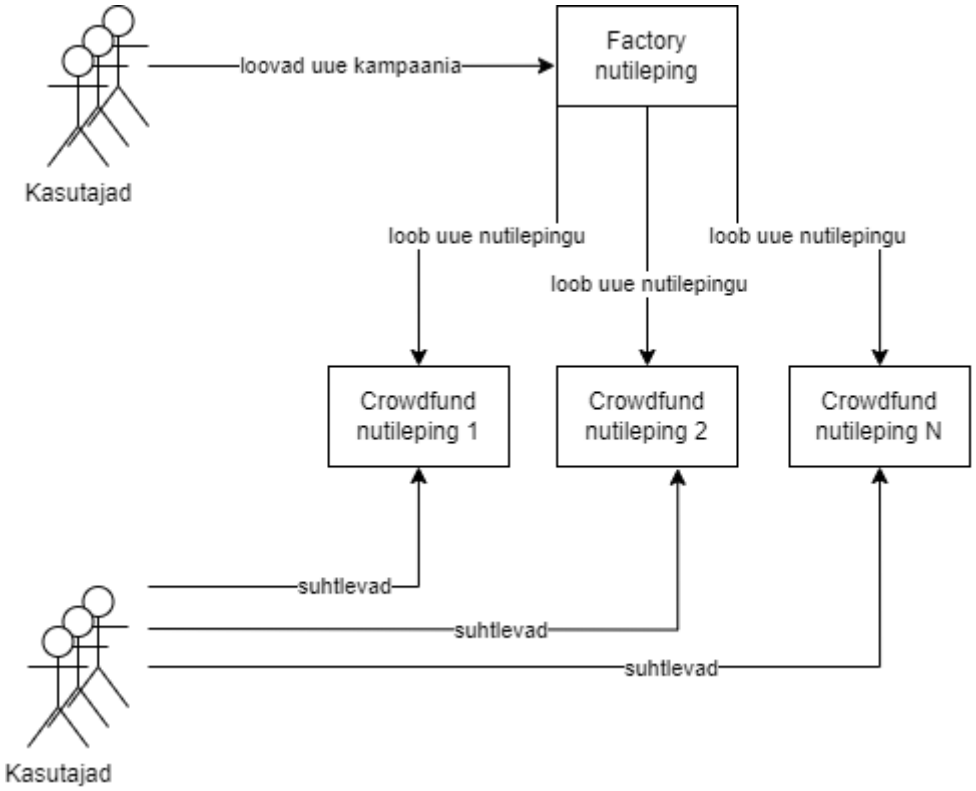


Joonis 6. Kampaania loomise protsessi jadadiagramm

Factory nutileping sisaldab ka *pledge* funktsiooni, mis võimaldab kasutajal sooritada investeeringuid. Funktsioon võtab sisendiks kampaania identifikaatori väärtuse ning kutsub läbi *Crowdfund* nutilepingu välja *pledge* funktsiooni. Funktsiooni õnnestumise

korral sooritatakse investeeringu tehing ning logitakse *Factory* nutilepingu kaudu investeeringu informatsioon.

Crowdfund nutileping sarnaneb eelnevale ühelepingulisele struktuurile, kuid lepingut on modifitseeritud vastavalt sellele, et iga leping võimaldaks hoida maksimaalselt ühte kampaaniat. Selleks on loodud ka *isLaunched* atribuut, mille algne väärtus on väär, kuid kampaania loomisel määratakse atribuudi väärtus tõeseks. Kampaania loomisel kontrollitakse, et *isLaunched* atribuudi väärtus oleks väär. Tehase mustri rakendamine võimaldab rakenduse kasutajatel suhelda nii *Factory* kui ka loodavate *Crowdfund* tüüpi nutilepingutega (Joonis 7).

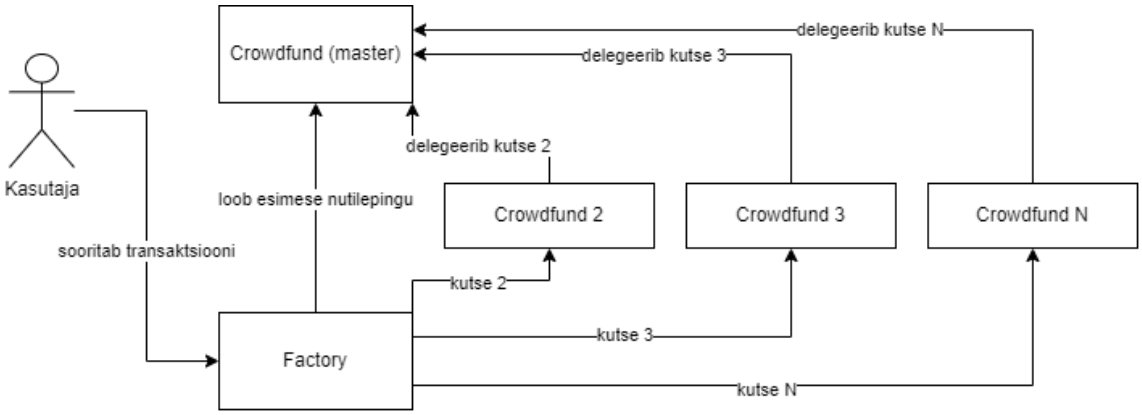


Joonis 7. Nutilepingutega suhtlemise struktuur kasutades tehase mustrit

4.3.2 Nutilepingute kloonimine

Tehase mustri rakendamisel tekib pidevalt uusi *Crowdfund* nutilepinguid, mille ainus omavaheline erinevus seisneb väärtustes, mis tekivad transaktsioonidega nende lepingute sisse. Nutilepingute kloonimine aitab vähendada uute nutilepingute loomisel ja juurutamisel tekkivaid gaasitasusid (*gas fees*). Kloonimisega juurutatakse esimene *Crowdfund* nutileping ainult ühe korra ning iga järgnev nutileping suunatakse käituma

esmalt juurutatud nutilepingu proksina, mis delegeerib kõik transaktsioonid esimesele nutilepingule (Joonis 8) [19]. Töö autor kasutas lahenduse implementeerimiseks *OpenZeppelin*'i poolt pakutavat nutilepingut, mis põhineb *ERC-1167* standardil [20].



Joonis 8. Kloonitud nutilepingute kutsete struktuur

Nutilepingute kloonimiseks oli vajalik importida *Factory* nutilepingusse *OpenZeppelin*'i raamistikus paiknev *Clone* nutileping [21]. Uute *Crowdfund* nutilepingute loomisel oli vajalik *Clone* nutilepingust tuleneva *clone* funktsiooni kasutamine, mis sisaldab kloonimise tehnilist realisatsiooni. Seoses sellega oli vajalik teostada ka mitmeid muid muudatusi mõlemas nutilepingus, et kogu funktsionaalsus püsiks sama ning nutilepingud kompileeruksid.

Factory nutilepingusse loodi konstruktor, milles loodava uue *Crowdfund* nutilepingu aadress salvestatakse *masterAddress* atribuuti. See on implementatsiooni sisaldava nutilepingu aadress, millele hakkavad järgnevad *Crowdfund* nutilepingud oma tehinguid edasi delegeerima. Veel tuli täiendada nutilepingut *setMasterAddress* funktsiooniga, millega on võimalik muuta konstruktoris seatud implementatsiooni sisaldava nutilepingu aadressi. Seda funktsiooni on võimalik õnnestunult välja kutsuda ainult *Factory* nutilepingu omanikul. Samuti tuli nutilepingusse lisada *setCrowdfundOwner* meetod, millega on võimalik muuta tekkinud nutilepingute omanikku.

Loodavatel *Crowdfund* kloonidel tekkis omaniku väärtuseks nullitud aadress, mistõttu ei olnud enam võimalik kasutada *Crowdfund* nutilepingus *Ownable* nutilepingu funktsioone, mis hoolitsesid juurdepääsu kontrollimise eest. Probleemi lahendamiseks tuli manuaalselt implementeerida juurdepääsu kontrollimine. Selleks lisati lepingusse *owner* atribuut, millele edastatakse omaniku väärtus lepingu initsialiseerimisel. Samuti ei

saa kloonida nutilepinguid, mis kasutavad konstruktorit. *Crowdfund* nutilepingust oli vajalik eemaldada konstruktor ning lisada uus funktsioon *initialize*, mis seab varasemalt konstruktoris määratud väärtused. Seda funktsiooni kutsutakse välja *Factory* lepingus uute *Crowdfund* lepingute loomisel.

4.4 Testimine

Üheks testimise efektiivsuse mõõdikuks on koodi kaetavuse protsent, mis näitab, kui suur osa projektist on kaetud ühiktestidega. Töö autor keskendus töö raames eelkõige nutilepingu testimisele, mille kaetavuse protsenti ei ole võimalik adekvaatselt hinnata, sest nutilepingute koodiridade arv moodustab tervest projektist marginaalse osa, umbes 3,5%.

Töö autor kasutas nutilepingu testimiseks *Hardhat*'i töökeskkonda ning lokaalset plokiahelat. Kogu nutilepingu funktsionaalsus on kaetud kolmekümne seitsme ühiktestiga, mille jooksutamiseks kulus keskmiselt 3 sekundit (Joonis 9).

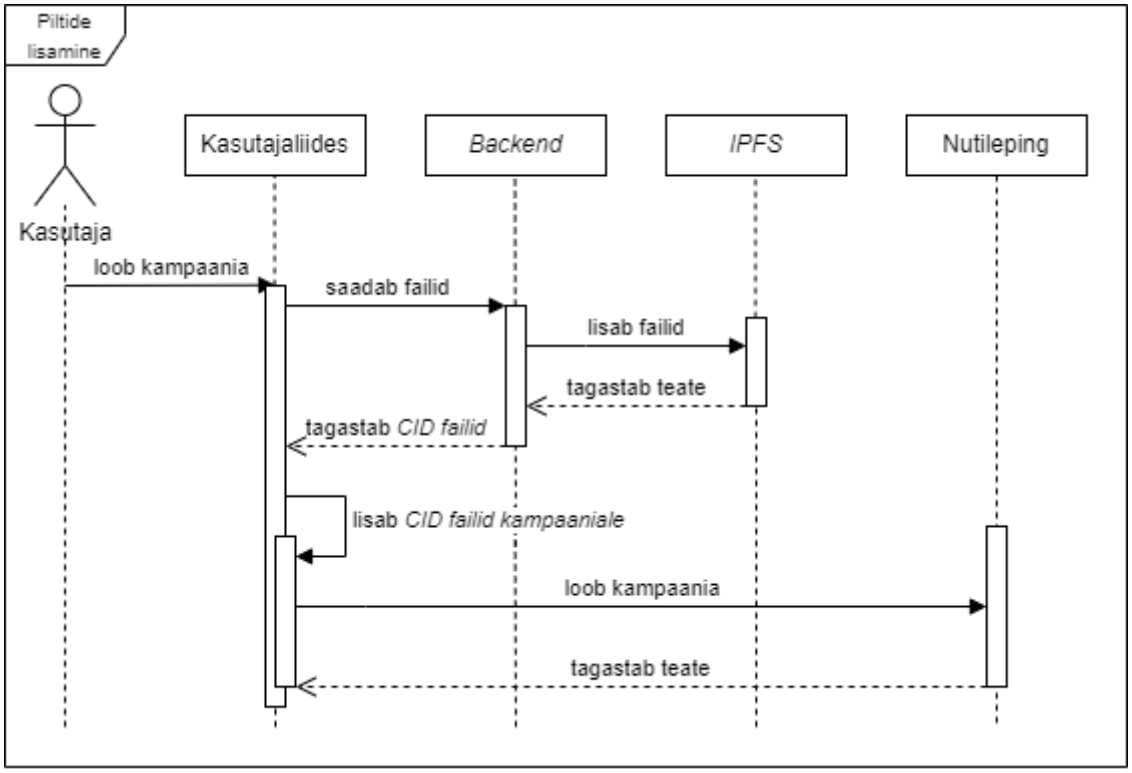
```
  Factory and Crowdfund contract
    Launching
The Hardhat Network tracing engine could not be initialized. Run Hardhat with --verbose to learn more.
      ✓ Count should be 0 when no contracts are created (1087ms)
      ✓ Only owner should be able to approve the campaign (152ms)
      ✓ Only owner should be able to reject the campaign
      ✓ Increase count after creating a new contract
      ✓ Launch campaign error when end date is smaller than start date
      ✓ Launch campaign error when duration is over 90 days
      ✓ Launch campaign error when starting time is in past (45ms)
      ✓ Launch campaign error when more than one campaign launched under one contract (51ms)
      ✓ Campaign contains data that it was launched with (46ms)
      ✓ Launching campaign emits event with correct arguments (51ms)
    Investing
      ✓ Reject investment on unapproved campaign (106ms)
      ✓ Reject empty investment
      ✓ Reject investment when campaign not launched (43ms)
      ✓ After investing 1 wei invested amount is 1 wei (115ms)
      ✓ After investing 25 wei, invested amount is 25 wei (46ms)
      ✓ After investing separately 50 and 25 wei into campaign, invested amount is 75 (52ms)
      ✓ Investing emits events with correct arguments
      ✓ Invest error when total amount is over goal
      ✓ Invest error when campaign has not started (68ms)
      ✓ Invest error when campaign has ended (64ms)
    Claiming
      ✓ Claim error when invoked by non-owner  (51ms)
      ✓ Claim error when campaign has not been approved (77ms)
      ✓ Claim error when campaign has not ended
      ✓ Contract owner gets funds on claim (82ms)
      ✓ Can't claim if funds already claimed (72ms)
      ✓ Claiming emits event with correct arguments (72ms)
    Refunding
      ✓ Can't refund if campaign has not been approved (61ms)
      ✓ Can't refund if campaign isn't over (67ms)
      ✓ Can't refund if campaign goal was achieved (58ms)
      ✓ Can't refund if campaign has already been claimed (53ms)
      ✓ Asking for refund transfers sum of own investments from contract (84ms)
      ✓ Refunding emits event with correct arguments (72ms)
    Distributing
      ✓ Only owner should be able to distribute funds (53ms)
      ✓ Can't refund if campaign has not been approved (57ms)
      ✓ Can't refund if campaign has not been claimed (64ms)
      ✓ After distributing 50 wei, investor should receive 50 wei (68ms)
      ✓ Distribute emits event with correct arguments (71ms)


  37 passing (3s)
```

Joonis 9. Ühiktestide nimekiri

4.5 Piltide lisamine plokiahelasse

Nutilepingus andmete hoiustamiseks tuleb andmete mahu pealt maksta transaktsioonide tegemisel tehingutasusid. Ühe kilobaidi suuruse andmehulga hoiustamine maksab ligikaudu 640 000 000 gaasiraha, mille ligikaudne väärtus oleks 1,10 eurot [22]. Rakenduse kasutajatel on võimalik loodavatele kampaaniatele pilte lisada, kuid piltide mahtu ei kontrollita ning erinevused võivad olla suured. Seega otsustati piltide lisamiseks kasutada *IPFS* tehnoloogiat, mis on detsentraliseeritud protokollide komplekt andmete edastamiseks ning muudab piltide salvestamise odavamaks [23].

Kampaania loomisel saadetakse pildid tagarakendusele, kus saavutatakse ühendus lokaalse *IPFS* sõlmega ning seejärel lisatakse need *IPFS* võrgustikku. Tagarakendusest saadetakse lisatud failid *CID* kujul tagasi kasutajaliidese kihti, kus lisatakse need nutilepingusse salvestamiseks (Joonis 10).



Joonis 10. Kampaania loomise ajal lisatavate piltide jadadiagramm

5 Analüüs

Antud peatükis hindab töö autor tulemuste realisatsiooni vastavust algselt seatud nõuetele ning selgitab nutilepingu arhitektuurilist teostust. Samuti selgitab plokiahela tehnoloogia kasutamisega kaasnevaid probleeme, nende lahendamise võimalusi ning pakub edasisi võimalusi prototüübi arendamisel.

5.1 Hinnang nõuete realisatsioonile

Funktsionaalsete nõuete realiseerimine lahendab rakenduse töövõimest olulise puudujäägi, milleks on raha tagastamise ja raha välja jagamise funktsionaalsus. Tänu lisatud funktsioonidele, võib pidada ühisrahastusplatvormi prototüübi töövõimet terviklikuks. Samuti on loodud lahendus, mis aitab salvestada nutilepingus rohkem informatsiooni loodava kampaania kohta.

Funktsionaalsete nõuete ainsaks murekohaks osutus „*Pending*“ staatusega kampaaniate avalikult nähtavus. Realiseeritud lahendus töötab küll kasutajaliideses, kuid petlikult. Tagarakenduses filtreeritakse „*Pending*“ kampaaniad välja ning saadetakse need kasutajaliidesesse, kus näidatakse ainult kasutajatele, kellel on õigus neid näha. Tegelikult on kõik kampaaniad plokiahelas huvilistele avalikult kättesaadavad. Avalikus plokiahelas andmete hoidmine, isegi kui neid hoitakse privaatses muutujas, ei ole tegelikkuses privaatne [24]. Delikaatsete andmete hoiustamiseks plokiahelas tuleks kasutada näiteks krüpteerimisvahendeid või privaatset plokiahelat, kuid töö raames arendatava prototüübi sihiks on kasutada rakendust avalikus plokiahelas.

Samuti realiseeris töö autor mittefunktsionaalsetest nõuetest tulenevalt nutilepingute tehase ja kloonimise mustri. Sellega on nutilepingute arhitektuur abstraktsem ning efektiivsem. Realiseeritud said veel ülejäänud mittefunktsionaalsed nõuded, välja arvatud *Goerli* plokiahela peale ümber lülitamine ning sellest tulenevalt ka plokiahela jälgimistehnoloogia rakendamine. Arendusmeeskond on otsustanud *Hardhat*’i poolt pakutava lokaalse plokiahela kasuks, sest võrreldes *Goerli* plokiahelaga pakub see piiramatul hulgal mänguraha, mis muudab arendamise kogemuse mugavamaks. Kasutades *Goerli* plokiahelat, on selle kasutajatel võimalik saada ainult 0,02 *Goerli ethereum*’i päevas [25]. Arvestades, et plokiahelaga suhtlemisel kõik tehingud maksavad, sai *Goerli* plokiahelat kasutades see raha mõne päevaga otsa. Sellest tulenevalt ei

õnnestunud kasutatava lokaalse plokiahelaga ära siduda ka jälgimistehnoloogiat. Töö autori hinnangul ei ole see suureks murekohaks, sest tulevikus rakenduse kasutamine avalikus plokiahelas kaotab ära mõlema nõude vajaduse.

5.2 Nutilepingu arhitektuuri analüüs

Nutilepingute arhitektuuri muudatus algas murekohast, et esialgne leping sisaldab liiga palju informatsiooni. Lisaks potentsiaalne mõttekäik, et kui tulevikus sooviks ettevõte rakenduse jaoks avalikkuse ette tuua enda loodud krüptoraha, siis selle kasutamine nõuaks eraldiseisvaid nutilepinguid. Ettevõtte neli mentorit, kes töö autorit koos meeskonnaga probleemide korral juhendasid, leidsid samuti, et tunduks loogiline eraldiseisvate nutilepingute tekitamine. Selline lahendus võimaldaks iga loodava kampaania kohta luua uue nutilepingu ning kogu edasine suhtlus selle seotud kampaaniaga saaks toimuda läbi eraldiseisva nutilepingu.

5.2.1 Tehase mustri efektiivsus

Esmalt realiseeris töö autor ettevõtte mentorite poolt soovitud lahenduse, kus põhiline nutileping, *Factory*, hoolitseks ainult uute nutilepingute tekitamise, juurutamise ja nende hoiustamise eest. Kampaaniaga seotud toimingute tegemiseks, võetakse *Factory* nutilepingust vastava kampaaniaga seotud *Crowdfund* nutilepingu aadress ning kogu edasine suhtlus ning funktsionaalsus toimub just selle *Crowdfund* nutilepinguga.

Töö autor pidas sellist lahendust ebaefektiivseks. Problemaatiliseks osutus rakenduse kasutajale tema loodud kampaaniate ning tehtud investeeringute näitamine. Varasemalt oli võimalik efektiivselt lugeda kogu informatsiooni plokiahela logidest, kus hoiti kasutajate aadresse indekseeritud kujul, mis võimaldas efektiivselt saavutada nende filtreerimist suurest andmehulgast. Uue lahendusena logiti endiselt vastavaid andmeid, kuid logid olid seotud eraldiseisvate nutilepingutega ning nende filtreerimine ei olnud nii efektiivselt võimalik. See tähendab, et kasutajale tema loodud kampaaniate või tehtud investeeringute näitamiseks tuli tsükliga kontrollida kõiki nutilepinguid ning kontrollida ükshaaval, millisega on kasutaja seotud. Töö autor sooritas katse, milles võrdles kahe erineva lahenduse kiirust. Tabel 1 näitab, et logidest andmete lugemine on peaaegu 2 korda kiirem tsükliga lahendusest. Katse sooritati kasutades lokaalset plokiahelat ning töö autori hinnangul võib tulemuste vahe oluliselt suureneda kasutades mõnda teist plokiahelat.

Tabel 1. Tehase mustri lahenduste kiirused andmete filtreerimisel

Loodud kampaaniate arv	**Kampaaniate arv, millest soovitakse infot saada**	**Logide kasutamine**	**Tsükli kasutamine üle kõikide lepingute**
		Keskmine kiirus (ms)	
30	15	824	1527
60	30	1474	2869
100	50	2473	4598

Järgmiseks murekohaks sellise lahendusega osutus tühjade lepingute olemasolu. Kampaania loomisel pidi kasutaja esmalt kinnitama transaktsiooni uue *Crowdfund* nutilepingu loomiseks ning siis veelkord *Crowdfund* lepingus kampaania salvestamiseks. Kui peaks juhtuma, et kampaania andmed ei vasta nutilepingu nõuetele või mõnel muul põhjusel kasutaja teine transaktsioon ebaõnnestub, siis jääb tekkinud nutileping tühjaks, sest uuel kampaania lisamisel tekitatakse juba uus *Crowdfund* nutileping.

Töö autor täiendas probleemide lahendamiseks *Factory* nutilepingut funktsioonidega, mis võimaldavad kutsuda välja funktsioone *Crowdfund* nutilepingust. Sellise lahenduse puhul ei ole tühjade nutilepingute tekkimine võimalik, sest transaktsiooni kontrollimine toimub samaaegselt uue nutilepingu loomisega. *Factory* nutilepingus on võimalik ka logida kampaania loomist ja investeeringute tegemist, mis tähendab, et logid on uuesti kättesaadavad ühest nutilepingust ning andmete eristamine toimub efektiivselt.

5.2.2 Nutilepingute kloonimise analüüs

Uute *Crowdfund* nutilepingute baitkood (*byte code*) on kõikidel samasugune. Iga nutilepingu baitkoodi salvestamine nende loomisel nõuab liigselt palju gaasitasusid, mis on ebapraktiline lahendus. Nutilepingute kloonimise eesmärgiks on:

- vähendada gaasitasusid nutilepingute loomisel ja salvestamisel;
- toetada kloonide initsialiseerimist läbi tehase mustri;
- minimeerida kloonitavate nutilepingute baidi kood.

Nutilepingute kloonimise lahendust nimetatakse *ERC-1167* standardiks [26]. Lahendust aitab teostada nutilepingute transaktsioonide delegeerimise võimalus. Delegeerimine sarnaneb tavaliste transaktsioonidega, kuid sihtaadressi kood käivitatakse kutsutava lepingu kontekstis ja sõnumi saatja ega sisu väärtus ei muutu. Nutileping võib käivitamise ajal dünaamiliselt laadida koodi teiselt aadressilt (implementatsiooni sisaldavast nutilepingust), kuid lepingu mälu, aadress ja kontoseis võetakse ikkagi kutsutavast lepingust [27].

Töö autor võrdles kloonitud ja mittekloonitud nutilepingute loomisel tekkivaid gaasitasusid. Tulemused näitavad, et kloonitud nutilepingute puhul on gaasitasud peaaegu 2 korda odavamad ning vahe suureneb kampaaniate arvu kasvades (Tabel 2). Oluline on märkida, et võrdlus toimus *Remix*'i integreeritud programmeerimiskeskkonnas kasutades *Remix*'i virtuaalmasinat. Töö autori hinnangul võib tulemuste vahe veelgi suureneda kasutades mõnda muud keskkonda või plokiahelat.

Tabel 2. Kloonitud ja tavaliste kampaaniate loomise maksumuse võrdlus

Loodavate kampaaniate arv	**Factory**	**Clone Factory**
	Keskmine tehingu maksumus (wei)	
1	5 502 463	3 943 845
5	15 443 919	7 650 829
15	40 297 559	16 918 289
30	77 578 019	30 819 479
50	122 644 549	49 354 399

5.3 Andmete hoiustamine plokiahelas

Kõik tehingud nutilepingutega maksavad midagi, olgu see tehingutasu kasutajatele või ressursside tasu riist- ja tarkvarale. Ühisrahastusplatvormi prototüübi näitel võib osutuda keeruliseks potentsiaalsetele kasutajatele tehingutasude reklaamimine. Seetõttu tuleb

nutilepingud teha mahu poolest nii väiksed kui võimalik, kuid funktsionaalsuse poolest nii mahukad kui vajalik, et minimeerida tehingutasude suurused.

5.3.1 Mälu piirang nutilepingus

Realiseeritud *Crowdfund* nutilepingus sisaldab kampaania objekt 12 atribuuti, millest 4 sisaldavad endas *JSON* objektide väärtusi, mis on viidud tekstsõne kujule. Väärtused on grupeeritud *JSON* objektideks seetõttu, et vastasel korral ei kompileeru nutilepingu kood „*Stack too deep*“ tõrke tõttu. 2020. aastal läbiviidud küsitluses, kus uuriti vihatumaid aspekte *Solidity* programmeerimiskeele juures, asetses „*Stack too deep*“ tõrge populaarsuselt viiendal kohal [28]. Viga tekib olukorras, kus lepingu kood vajab juurdepääsu mälu pesas, mis asub sügavamal kui selle 16. element, st. andmeid on võimalik hoida rohkem, kuid nende ligipääsul on piir ees.

Funktsiooni kutsumisel luuakse raam, mis sisaldab funktsiooni valijat, tagastusaadressi, funktsiooni kõige vasak- ja parempoolsemat väärtustüübi argumenti. Tõrke tekkimine sõltub vastava tegevuse tsentraalsest operatsioonikoodist. Oluline on teada operatsioonikoodi argumentide arvu ja järjekorda, sest töö käigus surub masin argumendid mälus ükshaaval alla ning hiljem proovib nendele ligi pääseda. Tõrge tekib olukorras, kus masin on mõne argumendi mälus alla surunud, kuid hiljem proovib sellele ligi pääseda eelnevast positsioonist [29].

5.3.2 IPFS tehnoloogia

IPFS protokoll on spetsiaalselt disainitud failide hoiustamiseks detsentraliseeritud kujul. Andmed hoiustatakse räsitud kujul võti-väärtus põhimõttel, mis muudab nende kättesaamise efektiivseks. Seetõttu on tehnoloogia eeliseks skaleeritavus ja jõudlus. Failide sisestamisel võrgustikku, tükeldatakse see 256. baidiks ning jaotatakse võrgustikus olevate sõlmede vahel. Iga tükk sisaldab osa objekti andmetest ja viidet. Andmete tagastamisel *IPFS* võrgustikust tagastatakse *CID* baas, mida kasutatakse failide kättesaamiseks [30].

Antud töös kirjeldatud prototüübis salvestatakse plokiahelasse lisatud piltide *CID* väärtused. Tegelikkuses on võimalik kasutada avalikke võrguväravate võimalusi *IPFS* võrgustikus asuvate failide kättesaamiseks ja kuvamiseks, kuid prototüübi näitel ei olnud võimalik sellist lahendust kasutada. Avalikud võrguväravad ei suutnud saavutada ühendust rakenduse lokaalse *IPFS* sõlme portidega. Lisaks sellele on tasuta saadavad

avalikud võrguväravad ebastabiilsed ning andmete laadimine võib võtta liigselt kaua aega, mis lõppeb ebaõnnestunud päringuga.

5.3.3 Delikaatsed andmed

Crowdfund nutileping sisaldab kampaania looja isikuandmeid. Töö autori hinnangul on olukord problemaatiline, sest kasutaja peaks rakenduse kasutamisel nõustuma sellega, et tema andmed on plokiahelas kõikidele avalikult kättesaadavad. Rakendus on mõeldud kasutamiseks avalikus plokiahelas, mistõttu tuleks mõelda delikaatsete andmete krüpteerimise peale. Krüpteerida on võimalik ainult neid andmeid, mida soovitakse lugeda väljastpoolt plokiahelat. Kui krüpteerida andmeid, mille väärtusi on vaja nutilepingu sees lugemiseks, näiteks mõne kontrolli või funktsionaalsuse toimimiseks, tuleks nutilepingusse lisada ka dekrüpteerimise võti. Dekrüpteerimise võtme olemasolu nutilepingus kaotab krüpteerimise põhimõtte, sest võti oleks samuti avalikult kättesaadav.

Plokiahela tehnoloogia pideva arengu tõttu tekib järjepidevalt uusi lahendusi, mis proovivad lahendada plokiahela konfidentsiaalsuse probleemi. Näiteks on üheks selliseks lahenduseks *ShadowEth* süsteem, mis on loodud, et hoiustada nutilepinguid privaatselt avalikus plokiahelas. *ShadowEth*'i rakendamisel eraldatakse nutilepingust kontrollimise protsess selle käivitamisest. Avalikku plokiahelasse sisestatakse ainult transaktsioonide kontrollimiseks mõeldud kood, kuid lepingu funktsionaalsus jooksutatakse läbi privaatse plokiahela, kus kommunikatsioon on krüpteeritud [31]. Töö autori hinnangul tuleks prototüübi edasisel arendamisel kindlasti arvesse võtta isikuandmete privaatsuse aspekti.

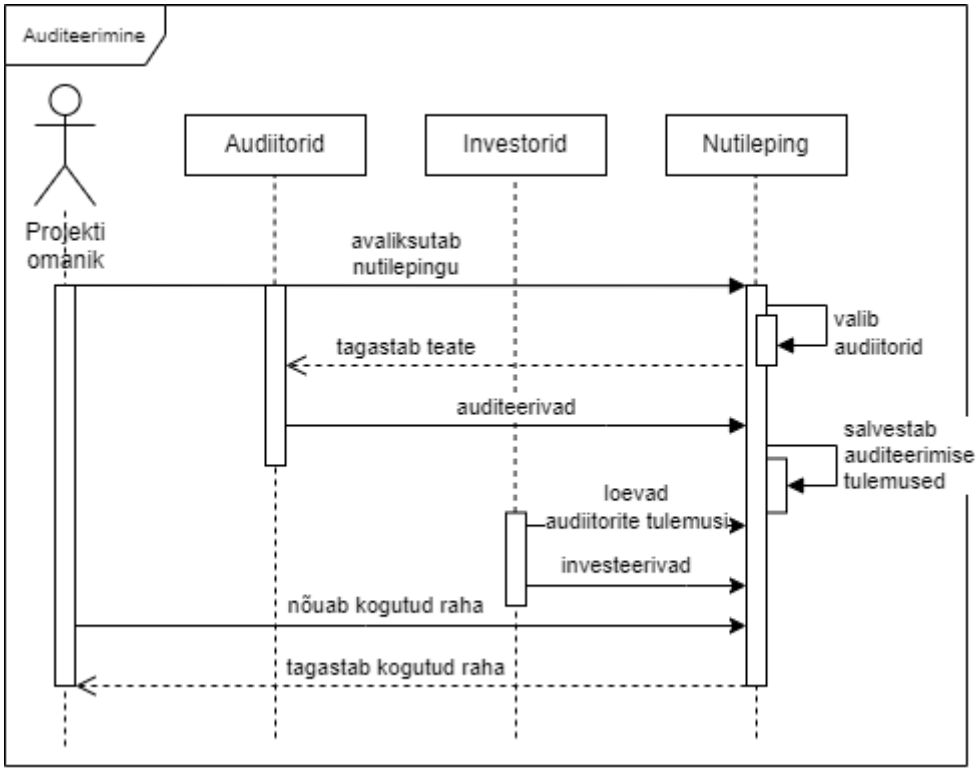
5.4 Plokiahela tehnoloogia ühisrahastuses

Traditsioonilised ühisrahastusplatvormid nõuavad kolme osapoolt: projekti omanikud, ühisrahastusplatvorm ja investorid. Ühisrahastuse suurimaks murekohaks on usaldusväärsus ning selle puudumisel tekkivad pettused. Mõned platvormid vastutavad pakutavate projektide usaldusväärsuse eest ise, kuid teised ainult vahendavad projekte.

Plokiahela tehnoloogia abil on võimalik ühisrahastusplatvormi risk maandada kasutades nutilepinguid, mis pakuvad usaldust kolmanda osapooleta. Investoritel on võimalik tutvuda nutilepingu lähtekoodiga veendumaks, et kogu funktsionaalsus töötab plaanipäraselt. Investeeritud raha hoitakse nutilepingus ning jagatakse laiali vastavalt implementeeritud loogikale. Nutilepingud lahendavad ühisrahastuse protsessi

töökindluse aspekti, kuid probleemiks võib osutuda andmete õigsus ning kasutajate anonüümsus, mis soosib kurjategijaid. 2018. aastal läbiviidud uuringu tulemused näitavad, et 78% plokiahelal põhinevatest ühisrahastusplatvormidest on pettused ning investoritelt on välja petetud üle 1,3 miljardi dollari [32].

Kasutajate anonüümsuse vastu võitlemine on raske, sest see tuleneb krüptorahakottide kasutamisest, mille ainsaks identifikaatoriks on selle aadress. Andmete õigsuse probleemi on võimalik lahendada kasutades audiitoreid. Nutilepinguga juurutatakse ka spetsiaalne audiitorite valimiseks loodud algoritm. Juurutamisel valib algoritm konkreetse nutilepingu jaoks sobivad audiitorid, kelle ülesandeks on kontrollida nutilepingut ning veenduda projekti õigsuses. Audiitorid esitavad oma tulemused plokiahelasse, mida on investoritel võimalik näha ning arvesse võtta oma investeerimisotsuste tegemisel (Joonis 11) [33]. Auditeerimise lahendus sarnaneb traditsioonilise ühisrahastuse puhul riigi poolt pakutavatele seadustele ning regulatsioonidele. Selle põhjal järeldab töö autor, et plokiahela poolt pakutav usaldus kolmanda osapooleta on pigem osaline, kuid mitte täielik.



Joonis 11. Jadadiagramm ühisrahastuse toimimisest audiitorite olemasolul

5.4.1 Tokeniseerimine

Ühisrahastust on võimalik kasutada nii heategevuse kui ka investeeringu eesmärgil. Heategevuse puhul ei oota annetajad tagasi midagi, kuid investeerides ootavad investorid vastutasuks väärtust. Selleks väärtuseks võib olla näiteks dividend, intress või millegi osalus. Kuna antud töö raames arendatava prototüübi kampaaniateks on roheprojektid, tundub töö autori jaoks mõistlik analüüsida just osaluse võimaldamist investoritele.

Selleks, et plokiahelas oleks võimalik varasid jälgida ning nendega kaubelda, tuleb need tokeniseerida. Tokeniseerimisega luuakse iga vara kohta münt (*token*), mis teda esindab ning lubab varade ülekandmist vastavalt standardsetele protokollidele. Ühisrahastuse näitel tuleks iga loodava kampaania kohta luua piiratud koguses uus münt, mis jaotatakse investorite vahel vastavalt nende investeeringute osakaaludele. Sellega tekib investorite osalustele väärtus, mida saab seostada näiteks aktsiatega. Müntide olemasolu pakub hulganisti uusi lahendusi. Mõned võimalused oleks näiteks:

- roheenergia projektist tuleneva tulu jaotamine investoritele andes neile rohkem münte;
- järelturu kauplemise võimalus, kus investorid saavad osta ja müüa osalusi kasutades münte;
- võimalus müntide tagasiostmiseks projekti omaniku poolt, millega on võimalik reguleerida väljastatud müntide kogust ning sellega ka nende väärtust [34].

5.5 Edasiarenduse võimalused

Prototüübi edasisel arendamisel tuleks esmalt lahendada ärilised küsimused, mis puudutavad investoritele investeeringutest tuleneva tootluse jaotamist. Töö autori hinnangul peaks detailsemalt analüüsima roheprojektide tokeniseerimise võimalust. Selleks, et väljastavatel müntidel oleks väärtus ka maailmas kasutusel oleva valuutaga, tuleks luua seos mõne teise krüptovaluutaga, näiteks *ethereum*'iga. Krüptovaluutad on maailmas endiselt populaarust koguv valdkond ning nende hinnad üsna volatiilsed. Loodava roheprojekti hind krüptovaluutas võib kampaania alguse ja lõpu vahel suurelt kõikuda, mistõttu peaks müntide väärtuse stabiliseerimiseks analüüsima *stablecoin*'ide tehnoloogia kasutamist. *Stablecoin*'id on digitaalsed väärtusühikud, mis ei ole ühegi

konkreetse valuuta vorm, vaid püüavad minimeerida hinnakõikumisi tuginedes stabiliseerimisvahendite komplektile [35].

5.5.1 Goerli plokiahela kasutamine

Seni on prototüübi arendamisel kasutatud lokaalset plokiahelat. Töö autor koos arendusmeeskonnaga proovis läbitud praktika käigus kasutada ka *Goerli* plokiahelat, kuid probleemiks kujunes vähene *Goerli* plokiahelas kasutatava *ethereum*'i saadavus. Siiski on *Goerli* plokiahela kasutamine kasulik, sest see matkib *Ethereum*'i põhivõrku, kuid teeb seda testimiseks mõeldud valuutaga. *Goerli* plokiahela kasutamine võimaldaks paremini kasutada ka populaarseid plokiahelaid jälgivaid tehnoloogiaid, näiteks *Etherscan*. Selliste tehnoloogiate kasutamine muudab plokiahela tehnoloogia kasutamise efektiivsemaks, sest võimalike probleemide korral saab näha kõikide plokiahela transaktsioonide ajalugu ning nende sisu. *Goerli* plokiahela kasutamine nõuab küll täiendavaid konfiguratsioone, kuid prototüübi arendamise lõppfaasis muudab see *Ethereum*'i põhilisele plokiahelale vahetamise kergemaks võrreldes lokaalse plokiahelaga. Samuti tuleks analüüsida plokiahelas kasutatava valuuta erinevaid võimalusi, et lahendada vähese *Goerli ethereum*'i kättesaadavuse probleem.

5.5.2 Andmebaasi integreerimine

Ettevõtte sooviks oli arendada detsentraliseeritud ühisrahastusplatvormi prototüüp kasutades plokiahela tehnoloogiat andmete hoiustamiseks. Traditsioonilistel andmebaasidel ja plokiahelal põhinevatel lahendustel on omad eelised ja puudused. Andmebaaside tugevusteks on kättesaadavus, rikkalikud päringud, reaalajas analüüs ning madal latentsus. Plokiahelal põhinevate lahenduste põhilisteks eelisteks on detsentraliseeritus, järjepidevus ning skaleeritavus. Plokiahel saab kasu andmebaaside tugevustest, kui integreerida andmebaas plokiahelaga või luua plokiahela tehnoloogiale orienteeritud andmebaas [36]. Antud prototüübis kasutatakse plokiahelat kogu informatsiooni hoiustamiseks. Töö autori hinnangul ei ole tegemist mõistliku lahendusega, sest nutilepingute transaktsioonide tehingutasud on positiivses korrelatsioonis hoitavate andmete mahtudega.

Andmebaasi olemasolul on võimalik filtreerida salvestamist vajavaid andmeid ning nutilepingusse sisestada ainult funktsionaalsuse toimimiseks olulised andmed. Andmebaasi kasutamisega tekib ka võimalus siduda rakendus väliste teenusepakkujatega,

kes võimaldavad reaalajas jälgida plokiahelas toimuvaid muudatusi, lugeda nende logisid ning salvestada neid andmebaasi. Sellise lahendusega kulutatakse vähem ressursse plokiahela peale ning rakenduse töövõime peaks muutuma efektiivsemaks.

6 Kokkuvõte

Ühisrahastus on hea viis muuta taastuvenergia populaarseks ja kättesaadavaks. Eesti Energias alustati varasemalt detsentraliseeritud ühisrahastusplatvormi prototüübi arendamist, kuid põhjalikum plokiahela tehnoloogia analüüs ja oluline funktsionaalsus oli puudulik. Rakenduse keskseks elemendiks on plokiahel, mis vastutab andmete hoidmise eest ning nutileping, mis vastutab rakenduse toimimise eest. Töö eesmärgiks oli analüüsida prototüübi plokiahela tehnoloogilisi lahendusi, arendada puuduolev raha tagastamise ning välja jagamise funktsionaalsus ning kasutada levinud praktikaid lahenduste tegemisel.

Töö autor muutis nutilepingute arhitektuuri rakendades nii tehase kui ka kloonimise mustrit. Selle tulemusel vähenesid uute nutilepingute loomisel tekkivad tehingutasud ning andmete lugemine muutus kiiremaks. Samuti lisas töö autor puuduoleva funktsionaalsuse, mis muudab rakenduse töövõime terviklikuks.

Plokiahela kasutamine ühisrahastuses kaotab kolmanda osapoole vajaduse ning pakub tehingute läbipaistvust, kuid loob uued murekohad. Nendeks on kasutajate anonüümsus, andmete õigsuse kontrollimine, andmete hoidmine plokiahelas ning tekkiva tulu jaotamine. Probleemide lahendamiseks on võimalik kasutada kolmandate osapoolte loodud lahendusi, näiteks audiitori vahelüli lisamist, mis lahendaks andmete õigsuse probleemi. Rakenduse edasiarendamisel peaks põhjalikumalt analüüsima andmebaasi integreerimise ning projektide tokeniseerimise võimalust.

Kasutatud kirjandus

[1] L. Deng, Q. Ye, D. P. Xu, W. Sun, and G. Jiang, "A literature review and integrated framework for the determinants of crowdfunding success," *Financial Innovation*, vol. 8, no. 1, pp. 1–70, Dec. 2022, doi: 10.1186/S40854-022-00345-6/TABLES/11.

[2] "Alustage investeerimist kinnisvaraga tagatud ärilaenudesse - Estateguru." https://estateguru.co/et/ (accessed May 03, 2023).

[3] S. S. Sarmah, "Understanding Blockchain Technology," *Computer Science and Engineering*, vol. 8, no. 2, pp. 23–29, 2018, doi: 10.5923/j.computer.20180802.02.

[4] R. Chatterjee and R. Chatterjee, "An Overview of the Emerging Technology: Blockchain," 2017, doi: 10.1109/CINE.2017.33.

[5] "What was the DAO Hack? - GeeksforGeeks." https://www.geeksforgeeks.org/what-was-the-dao-hack/ (accessed Apr. 22, 2023).

[6] "Kuidas plokiahel töötab? | Binance Academy." https://academy.binance.com/et/articles/how-does-blockchain-work (accessed May 03, 2023).

[7] S. Meunier, "Blockchain 101: What is Blockchain and How Does This Revolutionary Technology Work?," *Transforming Climate Finance and Green Investment with Blockchains*, pp. 23–34, Jan. 2018, doi: 10.1016/B978-0-12-814447-3.00003-3.

[8] "Proof-of-stake (PoS) | ethereum.org." https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed May 03, 2023).

[9] "Smart Contracts." https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html (accessed May 03, 2023).

[10] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016, doi: 10.1109/ACCESS.2016.2566339.

[11] P. Catchlove, "Smart Contracts: A New Era of Contract Use," *SSRN Electronic Journal*, Dec. 2017, doi: 10.2139/SSRN.3090226.

[12] "Moralis Web3 Documentation - Step-by-Step Web3 API Tutorials." https://docs.moralis.io/ (accessed May 01, 2023).

[13] "Documentation." https://docs.ethers.org/v5/ (accessed May 01, 2023).

[14] "Documentation | Ethereum development environment for professionals by Nomic Foundation." https://hardhat.org/docs (accessed May 01, 2023).

[15] "What is Scrum? | Scrum.org." https://www.scrum.org/resources/what-scrum-module (accessed May 02, 2023).

[16] “Mis on funktsionaalsed nõuded: näited, määratlus, täielik juhend.” https://visuresolutions.com/et/blog/functional-requirements/ (accessed May 01, 2023).
[17] “Crowd Fund | Solidity 0.8 - YouTube.” https://www.youtube.com/watch?v=P-4ucHdjGpU (accessed May 01, 2023).
[18] V. Sarcar, “Simple Factory Pattern,” *Design Patterns in C#*, pp. 465–475, 2020, doi: 10.1007/978-1-4842-6062-3_24.
[19] “Learn Solidity: The Factory Pattern | by wissal haji | Better Programming.” https://betterprogramming.pub/learn-solidity-the-factory-pattern-75d11c3e7d29 (accessed May 01, 2023).
[20] “Workshop Recap: Cheap contract deployment through Clones - OpenZeppelin blog.” https://blog.openzeppelin.com/workshop-recap-cheap-contract-deployment-through-clones/ (accessed May 01, 2023).
[21] “openzeppelin-contracts/Clones.sol at master · OpenZeppelin/openzeppelin-contracts.” https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/Clones.sol (accessed May 01, 2023).
[22] “Costs Of Storing Data On The Blockchain | LinkedIn.” https://www.linkedin.com/pulse/costs-storing-data-blockchain-rohan-pinto/ (accessed May 01, 2023).
[23] “What is IPFS? | IPFS Docs.” https://docs.ipfs.tech/concepts/what-is-ipfs/ (accessed May 01, 2023).
[24] “Accessing Private Data in Smart contracts | by QuillAudits Team | Medium | Medium.” https://quillaudits.medium.com/accessing-private-data-in-smart-contracts-quillaudits-fe847581ce6d (accessed Apr. 28, 2023).
[25] “Goerli Faucet.” https://goerlifaucet.com/ (accessed Apr. 28, 2023).
[26] P. Murray, N. Welch, and J. Messerman, “ERC-1167: Minimal Proxy Contract,” *Ethereum Improvement Proposals, no. 1167*, Jun. 2018. https://eips.ethereum.org/EIPS/eip-1167 (accessed May 01, 2023).
[27] “Cloning Solidity smart contracts using the factory pattern - LogRocket Blog.” https://blog.logrocket.com/cloning-solidity-smart-contracts-factory-pattern/ (accessed May 01, 2023).
[28] “Stack Too Deep.” https://soliditydeveloper.com/stacktoodeep (accessed May 01, 2023).
[29] “‘Stack Too Deep’- Error in Solidity | by Aventus Network | Coinmonks | Medium.” https://medium.com/coinmonks/stack-too-deep-error-in-solidity-608d1bd6a1ea (accessed May 01, 2023).
[30] S. Kumar, A. K. Bharti, and R. Amin, “ Decentralized secure storage of medical records using Blockchain and IPFS : A comparative analysis with future directions ,” *Security and Privacy*, vol. 4, no. 5, Sep. 2021, doi: 10.1002/SPY2.162.
[31] R. Yuan, Y. Bin Xia, H. B. Chen, B. Y. Zang, and J. Xie, “ShadowEth: Private Smart Contract on Public Blockchain,” *J Comput Sci Technol*, vol. 33, no. 3, pp. 542–556, May 2018, doi: 10.1007/S11390-018-1839-Y/METRICS.
[32] “SATIS Group Report: ‘78% of ICOs are Scams.’” https://cryptoslate.com/satis-group-report-78-of-icos-are-scams/ (accessed May 01, 2023).
[33] Y. Xu *et al.*, “A decentralized trust management mechanism for crowdfunding,” *Inf Sci (N Y)*, vol. 638, p. 118969, Aug. 2023, doi: 10.1016/J.INS.2023.118969.
[34] J. Roth, F. Schär, and A. Schöpfer, “The Tokenization of Assets: Using Blockchains for Equity Crowdfunding,” pp. 329–350, 2021, doi: 10.1007/978-3-030-52275-9_19/COVER.

[35] D. Bullmann, J. Klemm, and A. Pinna, “In Search for Stability in Crypto-Assets: Are Stablecoins the Solution?,” *SSRN Electronic Journal*, Nov. 2019, doi: 10.2139/SSRN.3444847.
[36] M. Raikwar, D. Gligoroski, and G. Velinov, “Trends in Development of Databases and Blockchain,” *2020 7th International Conference on Software Defined Systems, SDS 2020*, pp. 177–182, Apr. 2020, doi: 10.1109/SDS49854.2020.9143893.

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

Mina, Kevin Pajula

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Plokiahela tehnoloogia analüüs ja arendamine detsentraliseeritud ühisrahastusplatvormi prototüübi näitel ettevõttes Eesti Energia AS", mille juhendaja on Liisa Jõgiste
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2023

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

Lisa 2 – Factory nutilepingu kood

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/proxy/Clones.sol";
import "./Crowdfund.sol";

contract Factory is Ownable {

  constructor() {
    Crowdfund masterCrowdfund = new Crowdfund();
    masterAddress = address(masterCrowdfund);
  }

  event Launch(
    uint256 id,
    address indexed creator,
    string name
  );
  event Pledge(uint256 id, address indexed caller, uint256 amount);
  event SetMasterAddress(address oldAddress, address newAddress);

  mapping(uint256 => Crowdfund) public contracts;
  uint256 public count;
  address public masterAddress;

  function launch(
    uint256 _goal,
    uint32 _startAt,
    uint32 _endAt,
    string memory _name,
    string memory _descriptions,
    string memory _propertyInfo,
    string memory _urls,
    string memory _creatorInfo
  ) external {
      count++;
      Crowdfund crowdfund = Crowdfund(Clones.clone(masterAddress));
      crowdfund.initialize(count, owner());
      crowdfund.launch(_goal, _startAt, _endAt, msg.sender, _name,
_descriptions, _propertyInfo, _urls, _creatorInfo);
      contracts[count] = crowdfund;
      emit Launch(
        count,
        msg.sender,
        _name
    );
  }
```

```
  function pledge(uint256 _id) external payable {
    contracts[_id].pledge{value: msg.value}(msg.sender);
    emit Pledge(_id, msg.sender, msg.value);
  }

  function setMasterAddress(address _newAddress) external onlyOwner {
    address oldAddress = masterAddress;
    masterAddress = _newAddress;
    emit SetMasterAddress(oldAddress, _newAddress);
  }

  // Functions to prevent Hardhat from throwing errors in console
  // * receive function
  receive() external payable {}

  // * fallback function
  fallback() external payable {}
}
```

Lisa 3 – Crowdfund nutilepingu kood

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Crowdfund {
    event Launch(
        uint256 id,
        address indexed creator,
        uint256 goal,
        string name,
        uint32 startAt,
        uint32 endAt,
        string descriptions,
        string propertyInfo,
        string urls,
        string creatorInfo
    );
    event Pledge(address indexed caller, uint256 amount);
    event Claim(uint256 id);
    event Approve(uint256 id);
    event Reject(uint256 id);
    event Refund(address indexed caller, uint256 amount);
    event SetOwner(address oldOwner, address newOwner);
    event Distribute(address investor, uint256 amount);

    enum Status {
        Pending,
        Approved,
        Rejected
    }

    struct Campaign {
        address creator;
        uint256 goal;
        string name;
        uint256 pledged;
        uint32 startAt;
        uint32 endAt;
        bool claimed;
        string descriptions;
        string propertyInfo;
        string urls;
        string creatorInfo;
        Status status;
    }

    address public owner;
    uint256 public id;
```

```
    Campaign public campaign;
    mapping(address => uint256) public pledgedAmount;
    bool private isLaunched;
    bool private isInitialized;

    function initialize(uint256 _count, address _owner) external {
        require(isInitialized == false, "Contract has already been
initialized");
        id = _count;
        isLaunched = false;
        owner = _owner;
        isInitialized = true;
    }

    function launch(
        uint256 _goal,
        uint32 _startAt,
        uint32 _endAt,
        address _creatorAddress,
        string memory _name,
        string memory _descriptions,
        string memory _propertyInfo,
        string memory _urls,
        string memory _creatorInfo
    ) external {
        require(
            isLaunched == false,
            "Campaign is already launched under this contract"
        );
        require(
            _startAt >= block.timestamp - 60,
            "Starting time can't be in the past"
        );
        require(
            _endAt >= _startAt,
            "Starting time can't be greater than ending time"
        );
        require(
            _endAt <= _startAt + 90 days,
            "Campaign duration can't exceed time limit of 90 days"
        );

        campaign = Campaign(
            _creatorAddress,
            _goal,
            _name,
            0,
            _startAt,
            _endAt,
            false,
            _descriptions,
```

```
            _propertyInfo,
            _urls,
            _creatorInfo,
            Status.Pending
        );

        emit Launch(
            id,
            _creatorAddress,
            _goal,
            _name,
            _startAt,
            _endAt,
            _descriptions,
            _propertyInfo,
            _urls,
            _creatorInfo
        );
        isLaunched = true;
    }

    function pledge(address caller) external payable {
        require(campaign.status == Status.Approved, "Campaign is not
approved");
        require(msg.value > 0, "Investment must be bigger than 0");
        require(isLaunched == true, "Campaign is not launched");
        require(
            msg.value + campaign.pledged <= campaign.goal,
            "Campaign can't be overfunded"
        );
        require(
            block.timestamp >= campaign.startAt,
            "Campaign has not started yet"
        );
        require(block.timestamp <= campaign.endAt, "Campaign has ended");

        campaign.pledged += msg.value;
        pledgedAmount[caller] += msg.value;

        emit Pledge(caller, msg.value);
    }

    function claim() external {
        require(msg.sender == owner, "Message sender is not the owner");
        require(campaign.status == Status.Approved, "Campaign is not
approved");
        require(
            block.timestamp >= campaign.endAt,
            "Campaign has to be over in order to claim funds"
        );
        require(
```

```
            campaign.pledged >= campaign.goal,
            "Campaign has not achieved the goal"
        );
        require(!campaign.claimed, "Funds are already claimed");

        campaign.claimed = true;
        payable(campaign.creator).transfer(campaign.pledged);

        emit Claim(id);
    }

    function refund(address _investor) external {
        require(msg.sender == owner,
"Message sender is not the owner");
        require(campaign.status == Status.Approved,
"Campaign is not approved");
        require(!campaign.claimed,
"Campaign has already finished");
        require(block.timestamp > campaign.endAt,
"Campaign isn't over");
        require(
            campaign.pledged < campaign.goal,
            "Can't refund if campaign goal was achieved"
        );

        uint256 balance = pledgedAmount[_investor];
        pledgedAmount[_investor] = 0;
        payable(_investor).transfer(balance);

        emit Refund(_investor, balance);
    }

    function distribute(address _investor) external payable {
        require(msg.sender == owner,"Message sender is not the owner");
        require(campaign.status == Status.Approved,
"Campaign is not approved");
        require(campaign.claimed == true,
"Campaign is has not been claimed");

        payable(_investor).transfer(msg.value);
        emit Distribute(_investor, msg.value);
    }

    function approve() external {
        require(msg.sender == owner, "Message sender is not the owner");
        campaign.status = Status.Approved;
        emit Approve(id);
    }

    function reject() external {
        require(msg.sender == owner, "Message sender is not the owner");
```

```
        campaign.status = Status.Rejected;
        emit Reject(id);
    }

    function setOwner(address _newOwner) external {
        require(msg.sender == owner, "Message sender is not the owner");
        address oldOwner = owner;
        owner = _newOwner;

        emit SetOwner(oldOwner, _newOwner);
    }
}
```