

Automatiseren Claim Process

*Blockchain en Smart contract technologie gebruiken
om een gedistribueerd claim systeem te ontwikkelen*

Door: Calum Iain Munro



Software Development, ICA, VT

HBO bachelorscriptie:

Versie: 1 (Draft)

Datum: 18 mei 2018

Gegevens opdrachtgever:

Bedrijf: HeadForward B.V.

Contactpersonen: Daniël Siahaya

Gegevens opleiding:

Opleiding: HBO bachelor Informatica

School: Hogeschool van Arnhem en Nijmegen

Begeleider: Misja Nabben

Assessor: Rein Harle

Gegevens opdrachtnemer:

Teamlid: Calum Iain Munro (549288)

INHOUDSOPGAVE

1 Inleiding	3
1.1 Aanleiding	3
1.2 Relevantie	3
1.3 Probleemstelling	4
2 De blockchain technologie	5
2.1 Het algemene concept achter de blockchain	5
2.2 Type Blockchain	8
2.2.1 Privé Blockchain	8
2.2.2 Consortium Blockchain	8
2.2.3 Openbare Blockchain	9
2.2.4 Conclusie	9
3 De architectuur van de blockchain technologie	10
3.1 Blok (block)	10
3.2 Gedecentraliseerd netwerk	10
3.3 Consensus (overeenstemming) algoritmes	11
3.4 Smart contract	11
4 De architectuur van het proof of concept	13
4.1 Functionele en non-functionele requirements	14
4.1.1 Overige Requirements	15
4.1.2 Context - Actors en hun High-level Use Cases	16
4.2 Decision forces view	17
4.3 Container diagram	18
5 Bevindingen van het proof of concept	19
5.1 The good	19
5.2 The bad	19
5.3 Het resultaat	20
6 Conclusie	21
Bibliografie	21
7 Bijlagen	24
A Decision Forces View	25
B Smart contract: Claims	26

HOOFDSTUK 1

INLEIDING

Allereerst wordt in dit hoofdstuk het onderwerp van deze scriptie behandeld. Dit wordt gedaan door in paragraaf 1.1 de aanleiding van het onderzoek te bespreken. Waarna de relevantie in paragraaf 1.2 wordt besproken en aansluitend in paragraaf 1.3 de doel- en vraagstellingen worden geformuleerd.

Het verdere verslag bestaat uit de resultaten van het onderzoek. Er is in het begin algemeen onderzoek gedaan naar de blockchain-technologie en gerelateerde onderwerpen zoals smart contracts. Waarna er daarna verslag wordt gedaan over de implementatie en architectuur van beide de blockchain technologie en het proof of concept dat is ontwikkeld. Waarmee in paragraaf 6 de hoofdvraag wordt beantwoord. Hierna volgt een hoofdstuk voor verdere discussie en onderzoek.

1.1 Aanleiding

Zoals al aangegeven in het plan van aanpak [8] verzekeren verzekерingsmaatschappijen zoals Allianz panden voor miljoenen. Dit type verzekeringen worden gedeeld met meerdere verzekeraars, om zo het risico te verspreiden. Dit principe heet co-insurance en het probleem hiermee en ook gelijk de aanleiding voor dit onderzoek is dat het claimproces te veel tijd kost voordat deze wordt uitgekeerd naar de klant. Waardoor klanten van Allianz ontevreden zijn. Dit komt omdat dit proces door de verschillende instanties op verschillende handmatige manier worden uitgevoerd. Het proces wordt bijvoorbeeld bij Allianz gedaan met Excel bestanden, maar dit verschilt per verzekeringmaatschappij. Een claim kan dus vaak meer dan 3 maanden duren voordat deze werkelijk wordt uitbetaald.

1.2 Relevantie

De relevantie van dit onderzoek is om de laatste technologie op softwaregebied te onderzoeken en hiermee een proof of concept te ontwikkelen. In dit geval heeft de opdrachtgever aangegeven om in dit onderzoek naar de blockchain en smart contracts te willen kijken.

1.3 Probleemstelling

Het doel van deze scriptie is om aan te tonen hoe blockchaintechnologie en smart contracts gebruikt kunnen worden om informatie over claims van verzekeringen veilig te delen en te controleren tussen partijen die elkaar niet noodzakelijk vertrouwen.

Dit wordt bewezen door een proof-of-concept softwareapplicatie voor de use case van elektronische verzekeringsgegevens te maken. Andere use-cases liggen buiten de scope van dit onderzoek.

De hoofdvraag (**MRQ**) van dit onderzoek is:

“Hoe is de blockchain technologie in te zetten om het claimproces van Allianz te automatiseren?”

Deze vraag is onderverdeeld in verschillende deelvragen (**SRQ**):

- **SRQ1:** “*Wat is de blockchain en hoe werkt het?*” om deze vraag te beantwoorden wordt er een literatuuronderzoek uitgevoerd naar een technische basis termen rondom de blockchain-technologie en gerelateerde onderwerpen.
- **SRQ2:** “*Uit welke technologieën, use cases, requirements en concerns bestaat het te ontwikkelen proof of concept?*” om deze vraag te beantwoorden wordt er een software architectuur opgebouwd op basis van de wensen van de opdrachtgever, waaruit een literatuur onderzoek en vergelijking wordt uitgevoerd op verschillende oplossingsrichtingen en technologieën.
- **SRQ3:** “*Welke kansen & knelpunten bestaan bij het toepassen van de blockchain?*” om deze vraag te beantwoorden wordt er een experiment uitgevoerd op basis van de kennis die is opgedaan in de vorige SRQ1 en SRQ2 deelvragen.

De conclusie van deze deelvragen beantwoorden de hoofdvraag van dit onderzoek in paragraaf 6. Het resultaat van het experiment dat in SRQ3 wordt uitgevoerd zal in dit verslag alleen bestaan uit de smart contracts code. De verdere backend en front-end code die is ontwikkeld wordt naast dit document meegeleverd maar is te groot om opgenomen te worden in de bijlagen.

HOOFDSTUK 2

DE BLOCKCHAIN TECHNOLOGIE

In dit hoofdstuk wordt er een korte uitleg gegeven over een aantal basis technische termen in cryptografie, blockchain-technologie en gerelateerde onderwerpen zoals smart contracts. Dit zodat we weten waar we het over hebben wanneer we het proof of concept verder in dit verslag behandelen.

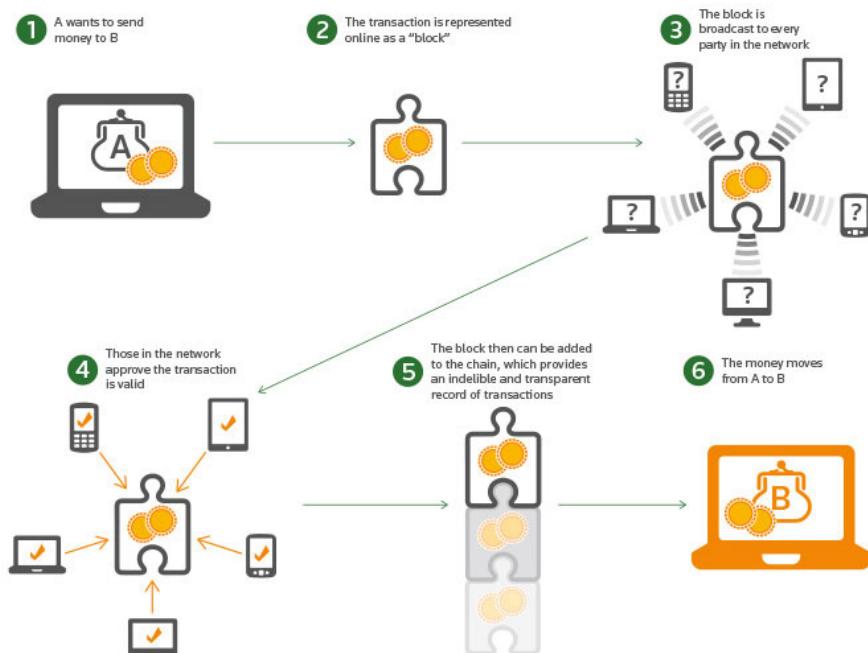
2.1 Het algemene concept achter de blockchain

De Blockchain technologie is wereldwijd bekend geworden door de introductie van de digitale valuta genaamd Bitcoin. De Bitcoin werd geïntroduceerd in 2008 door Satoshi Nakamoto in een white paper "Bitcoin: A Peer-To-Peer Electronic Cash System" [15]. Hierin legt hij uit hoe in een gedecentraliseerde softwareomgeving, geld veilig overgemaakt kan worden. Denk hierbij aan een online betaling, waar de een partij rechtstreeks naar een andere partij geld kan overmaken zonder de verschillende financiële instellingen die de transactie faciliteert.

In dit onderzoek gebruiken we de beschrijving het ICTU¹, die de blockchain beschrijft als: een specifieke databasetechnologie die leidt tot een gedistribueerd autonoom grootboekssysteem [6]. De integriteit van dit gedistribueerd autonoom grootboekssysteem wordt gewaarborgd doordat iedere partij zeggenschap heeft bij de validatie van een transactie. Dit versnelt het proces doordat beheerders en tussenpersonen worden uitgeschakeld. Meningsverschillen worden opgelost door een cryptografisch consensus algoritme.

¹<https://www.ictu.nl/>

Databasetransacties worden gegroepeerd en opgeslagen in blokken van data die vervolgens achter elkaar in een reeks wordt vastgelegd. Iedere deelnemer van het netwerk beschikt tot deze blokken en een nieuwe deelnemer download altijd eerst de gehele en recente historie van het netwerk. Hieruit krijgt de technologie de naam Blockchain. De koppeling tussen blokken en hun inhoud wordt beschermd door cryptografie en kan niet worden vervalst. Daarom kan informatie die eenmaal in een blockchain is ingevoerd niet worden gewist; In essentie bevat een blockchain een accuraat, tijd gestempeld en verifieerbaar archief van elke transactie die ooit is gemaakt. Figuur 2.1 geeft het algemene idee weer van hoe deze technologie werkt met als bekende use case de bitcoin.



Figuur 2.1: Illustrert hoe de blockchain werkt [9]

De blockchain technologie lost verschillende problemen op die voorkomen bij het gebruik van traditionele gecentraliseerde database technologieën die in handen zijn van één instantie. Dit soort technologieën vereisen vertrouwen dat de beheerder zorgvuldig omgaat met de toegang of bewerkingen van de data. Verder dat de database toegankelijk is voor de belanghebbenden en dat de instantie er de volgende dag nog is. Deze problemen komen niet voor in een gedecentraliseerde blockchain database. Dit komt omdat een nieuwe dienst, softwarebedrijf of markten op de blockchain de volgende zes design principes [10] hanteren:

1. Netwerk integriteit

Het systeem bewaakt de data integriteit doordat ieder lid in het netwerk alle transacties kan nalopen en kan controleren. In plaats dat er maar een lid is dit proces uitvoert. Gebruikers op het netwerk kunnen rechtstreeks waarde met elkaar uitwisselen door dit te registeren op een blok. Elk blok heeft een verwijzing naar een voorgaand blok, waardoor niemand een transactie kan verbergen of kan vervalsen. Dit omdat er meer andere gebruikers zijn met

de juiste realiteit.

2. Gedistribueerd

Het systeem is volledig Gedistribueerd. Dit houdt in dat er geen één punt is van controle of falen. Er is niet een gebruiker of organisatie die het systeem uit kan zetten.

3. Security

Satoshi's white paper [15] vereist dat het systeem beveildigd is door een public key infrastructure (PKI)². De PKI is een geavanceerde vorm van asymmetrische cryptografie, waar de gebruiker beide een publiek en privé sleutel ontvangt om zichzelf binnen het netwerk te identificeren en berichten kan versleutelen en ontsleutelen.

4. Eigendomsrechten

Eigendomsrechten over valuta en andere data zijn transparant in het netwerk en dus beschikbaar voor iedere gebruiker van het netwerk. Hierdoor dient een blockchain als een publiek register. Door een tool die Proof of Existence (PoE) ³ heet. Deze tool creert en registreert de cryptografische overzichten van akten, licenties en andere rechten van gebruikers.

5. Privacy

Gebruikers beheren hun eigen data. Er is geen centrale partij en gebruikers op het netwerk geven zelf aan wat ze aan informatie vrijgeven. Dit is echter allemaal optioneel en een gebruiker op de blockchain hoeft in de meeste implementaties alleen een publiek en private key te hebben. De gehele identificatie en verificatie laag zijn los van elkaar waardoor gebruikers op de blockchain de mogelijkheid hebben om anoniem te zijn.

6. Valuta als motivatie

Het systeem motiveert deelnemers van het netwerk door ze valuta te geven voor bepaalde acties op het netwerk. In het geval van de bitcoin, krijgen miners bitcoin geld voor het eerstvolgende blok te koppelen aan het vorige blok aan data. Dit wordt gedaan door een cryptografische puzzel op te lossen die geleidelijk lastiger word.

²https://en.wikipedia.org/wiki/Public_key_infrastructure

³https://en.wikipedia.org/wiki/Proof_of_Existence

2.2 Type Blockchain

Om een geïnformeerd besluit te maken welk type blockchain juist is voor het proof of concept worden de verschillende type blockchain in dit paragraaf behandeld. Dit wordt gedaan vanaf een hoog technisch niveau. In essentie zijn er drie types blockchain: privé, consortium en openbaar. Deze types kunnen daarna weer onderverdeeld worden in de open en gesloten categorieën blockchain. Per type blockchain wordt er ook gelijk gekeken naar de grote projecten die relevant zijn. Zodat in een vervolg hoofdstuk hierover een vergelijking kan worden uitgevoerd.

De categorie gesloten, waarin de types privé en consortium zitten zijn bedoeld voor een gelimiteerde omgeving zoals één of meerdere bedrijven en organisaties. Terwijl een openbare blockchain volledig open is en geen permissies zijn die mensen of systemen erbuiten houden.

2.2.1 Privé Blockchain

Voor een volledige private blockchain, moeten de schrijf rechten op een centrale plek staan die beheerd word vaak door een organisatie. De lees rechten kunnen beide publiekelijk of ook net zo beperkt zijn als de schrijf rechten. Applicaties die gebruik maken van een privé Blockchain zijn interne applicaties die alleen gebruikt worden binnen een bedrijf of organisatie. Want in andere gevallen wordt publieke lees rechten en controleerbaarheid vereist [7]. Voorbeelden van privé Blockchains zijn MultiChain en Hyperledger:

MultiChain - MultiChain is een platform voor het ontwikkelen en publiceren van privé blockchains. Het lost een aantal schaalbaarheid problemen [13] van de blockchain op met een geïntegreerde gebruiker permissie systeem. Verder biedt het bedrijven de mogelijkheid om zonder software ontwikkelaars een blockchain op te richten [3].

Hyperledger - Hyperledger is een open source project die ontwikkeld word met het doel om een geavanceerd bedrijfstakoverkoepelende blockchain implementatie te ontwikkelen. Het word gehost door de Linux Foundation [2] en wordt gezamelijk ontwikkeld door grote organisatie in financiën, banken, IoT, productie en technologie [1].

2.2.2 Consortium Blockchain

Consortium blockchain is gedeeltelijk Privé. Het Overeenstemmingproces ookwel consensusproces over de integriteit van de data op de blockchain wordt door een aantal voorafgeselecteerde nodes (gebruikers) uitgevoerd. Deze nodes zijn bijvoorbeeld 10 grote financiële instellingen die bij de aanmaak van een nieuwe block aan de blockchain zeggenschap hebben. Andere deelnemers van de blockchain hebben nogsteeds het recht om de besluiten van de 10 nodes te controlleren, maar ze hebben verder geen stem in het feit of de volgende block valide is.

Het voordeel van de consortium variant is deze efficiënter zijn en toch voldoende transactie transparantie geven. Ook is het niet een bedrijf die alleen oordeelt over de data. Voorbeelden van dit type zijn Ethereum en R3:

Ethereum - Het Ethereum project beschrijft zich als een gedecentraliseerd platform voor applicaties die precies gedraait worden zoals ze geprogrammeerd worden zonder enige kans van fraude, censuur of veranderingen van derden⁴. Applicaties, smart contracts, worden geprogrammeerd in de Solidity taal die voor het Ethereum project is ontwikkeld. Het wordt open-source ontwikkeld en er is een bondgenootschap, de Enterprise Ethereum Alliance⁵ dat bestaat uit 315 fortune 500 bedrijven en organisaties, die gezamenlijk werken aan het enige platform die smart contracts ondersteund op de blockchain.[11]

Het project kan gezien worden als verder uitgewerkte versie van Bitcoin die meer functionaliteiten toevoegt. Zo bestaat de status van Ethereum netwerk net zoals de Bitcoin uit meerdere objecten die 'accounts' worden genoemd, waarbij elke account een adres van 20 bytes en statusovergangen heeft. De staat van deze objecten worden opgeslagen in de blockchain waaruit gelijk afgeleiden kan worden waar valuta naartoe gaat.[20]

R3 - Dit is een gedistribueerd database-technologiebedrijf in New York. Het is verbonden met veel van de werelds grootste financiële instellingen, met als missie om de voordelen van de blockchain te realiseren [17].

2.2.3 Openbare Blockchain

Dit type blockchain is zoals de naam al aangeeft publiek beschikbaar tot iedereen in de wereld. Dit houdt in tegenstelling tot Consortium ook het consensusproces in. Iedere gebruiker op het netwerk heeft zeggenschap op de geldigheid van nieuwe data en kan nieuwe data schrijven.

Een volledig publieke blockchain is een open-source systeem door het gebruik van zogeheten cryptoeconomics gebruikers op economisch doeleinde motiveert om samen te werken. Hierdoor kunnen ontwikkelaars die gebruik maken van zo'n blockchain belangen zoals beschikbaarheid waarborgen. Bijvoorbeeld zorgt een hogere tarief in een transactie resulteren in snellere transacties of convergentie over de nieuwe blokken die toegevoegd worden aan de blockchain. Een voorbeeld van hiervan is het bekende Bitcoin project.

Bitcoin. Bitcoin is het bekendste voorbeeld van een blockchain project. Bitcoin staat vooral bekend als digitale valuta en online betalingssysteem die door gebruik van cryptografie om valuta-eenheden te geneert en reguleert. Verder gebruikt het cryptografie om de overdracht van fondsen te verifiëren zonder een centrale bank.

2.2.4 Conclusie

- Consortium Blockchain.

⁴<https://www.ethereum.org/>

⁵<https://entethalliance.org/>

HOOFDSTUK 3

DE ARCHITECTUUR VAN DE BLOCKCHAIN TECHNOLOGIE

Dit hoofdstuk gaat iets verder in op de basis architectuur van de blockchain. Dit zodat tijdens het ontwikkelen van de Proof of Concept we de basis begrippen van de architectuur begrijpen.

3.1 Blok (block)

De blockchain bied een gedistribueerd grootboeksysteem. Gegevens worden permanent opgeslagen in het netwerk via bestanden die blokken worden genoemd. Een blok is een document alle recente transacties die nog moeten worden vastgelegd. Het heeft daarom de naam blockchain, omdat het een reeks van blokken die naar de vorige verwijzen[14].

Een blok in het geval van de Bitcoin bestaat uit een header en een body [12]. De header bestaat uit drie stukken meta gegevens. De eerste is een verwijzing naar een vorige blokhash (Merkle-hash¹). Hierdoor verbindt het blok met de vorige uit de blockchain. De tweede set van meta gegevens is moeilijkheidsgraad, tijdstempel en nonce. Het laatste stuk metadata is de Merkle-tree root, een datastructuur die wordt gebruikt om alle transacties in het blok efficiënt samen te vatten [4].

3.2 Gedecentraliseerd netwerk

The interactions among user on blockchain principally use a decentralized network in which each user represents a node at which a blockchain client is installed. When a user performing a transaction with another user or when a node receives data from another node, it verifies the authenticity of the data. It then broadcasts the validated data to every other node connected to it [86]. Within such a mechanism, the data spreads across the whole network. The benefit of using this mechanism is the centralization of the human factor is minimized and trust shifts from the human agents of a central organization to an open source code [5].

¹https://en.wikipedia.org/wiki/Merkle_tree

3.3 Consensus (overeenstemming) algoritmes

Om de werking van de blockchain te begrijpen en te vertrouwen, moet het begrip van Consensus oftewel overeenstemmings algoritmes duidelijk zijn. Deze algoritmes worden gebruikt wanneer een (nieuw) blok aan informatie geverifieerd wordt. Het zorgt voor één historie van transacties waar de geschiedenis geen ongeldige of tegenstrijdige transacties bevat.

Dit is allemaal nodig omdat de blockchain draait in een zelf geregeerde, wantrouwende omgeving waar het nodig is om meningsverschillen over transacties binnen het netwerk op een lijn te krijgen. Het zorgt er bijvoorbeeld ook voor dat er niet één account is die meer uitgeeft dan dat het heeft, of waar hij of zij twee keer iets overmaakt, dit heet double-spending. De bekende consensus algoritmes zijn proof of work en proof of stake.

1. Proof of Work (PoW)

Het PoW consensus algoritme is het meest voorkomende algoritme in blockchain. Het werd geïntroduceerd door de Bitcoin en gaat ervan uit dat alle peers met rekenkracht mee stemmen door PoW-instanties, cryptografische puzzels op te lossen en hiermee het recht hebben om de volgende blok aan te maken in het netwerk. Zo maakt de Bitcoin gebruik van een hash-gebaseerde PoW, wat inhoudt dat de peers een nonce-waarde² proberen te vinden. Hierbij is wel de voorwaarden dat de vorige blokhash kleiner moet zijn dan de huidige doelwaarde die in de blokparameters staat van het vorige blok. Wanneer een dergelijke nonce wordt gevonden, maakt de miner het blok aan en stuurt hij het door naar zijn peers. Deze peers ontvangen dit dan en verifiëren of het klopt aan de hand van het vorige blok [5].

2. Proof-of-Stake (PoS)

Op het moment moet Proof-of-Stake zich nog bewijzen in de crypto valuta gemeenschap. Het is ontwikkeld om bestaande inefficiënte consensus algoritmes zoals PoW te vervangen. Het algemeen begrip van PoS is dat een peer (deelnemer van de blockchain), pas het stemrecht heeft op een nieuwe blok in de blockchain als de peer voldoende heeft ingezet in het netwerk. In het geval van PeerCoin³ worden nieuwe blokken gegeneerd door het netwerk op basis van niet gespendeerde valuta en hoe oud deze is [19].

Met deze methode wordt aangenomen dat mensen met meer valuta minder snel het netwerk zullen aanvallen [12]. Dit lost op het gebied van energiebesparing de problemen van PoW op, waar gebruikers miners aan zetten om valuta te ontvangen. Bij PoS wordt de valuta die niet beweegt steeds meer waard.

3.4 Smart contract

Het idee achter smart contracts is een "geautomatiseerd transactieprotocol dat de voorwaarden van een contract uitvoert" [16] en werd voor het eerst bedacht door cryptograaf Nick Szabo. Dit

²https://en.wikipedia.org/wiki/Cryptographic_nonce

³<https://peercoin.net/>

idee is door de opkomst van de blockchain populair geworden. Dit komt door dat de blockchain gedecentraliseerd is en daardoor de tussenpersonen bij een gecentraliseerde smart contracts applicatie eruithaald. Smart contracts is, in de context van blockchain, gewoon software die op een blockchain wordt gepubliceerd en die transacties kan ontvangen of uitvoeren. Iedere transactie heeft een adres en kan worden gevolgd.

HOOFDSTUK 4

DE ARCHITECTUUR VAN HET PROOF OF CONCEPT

Dit hoofdstuk onderzoekt naar de geschikte software architectuur voor het proof of concept en beantwoord hierdoor de vraag “*Uit welke technologieën, use cases, requirements en concerns bestaat het te ontwikkelen proof of concept?*” (**SRQ2** uit paragraaf 1.3). Om hier antwoord op te geven word de vraag in de volgende subvragen verder opgesplitst in:

1. Wie zijn stakeholders van het proof of concept en wat zijn hun algemene belang?
2. Wat zijn de requirements?
3. Welke technologieën beste geschikt voor de proof of concept use case?

De resultaten op deze vragen die worden gepresenteerd in dit hoofdstuk defineren gelijk het design van het proof of concept (PoC). Technologie keuzes worden in paragraaf 4.2 behandeld. De C4¹ software architectuur modeleer methodiek die is ontwikkeld door Simon Brown wordt gebruikt in dit proces.

¹<https://c4model.com/>

4.1 Functionele en non-functionele requirements

De scope van de implementatie beschreven beperkt zich tot de volgende gebruikers: verzekeraar, verzekerde, makelaar en administrator. Om zo compleet mogelijk alle functionele requirements te noteren zijn de user stories in de onderstaande tabel 4.1 vanuit de verschillende gebruikers perspectief geschreven. Hiermee beantwoorden we gelijk de vraag wie de stakeholders zijn in het systeem en wat hun belangen zijn.

Er is tijdens het opstellen van deze requirements gekozen om alleen het essentiële op te schrijven en deze zoveel mogelijk de versimpelen. Te weten dat, het quality attributes zoals het gebruiksgemak en security eisen van de PoC op een rendabel niveau moeten zijn.

Als een ...	Wil ik / moet ik ...	Zodat ...	Nr
Verzekerde	Kunnen zien welke ingediende claims ik heb	Ik de status hiervan kan controleren.	1.1
	Kunnen inloggen met een email en wachtwoord combinatie	Mijn informatie kan inlezen van de blockchain.	1.2
	Een claim aanvraag kunnen indienen	Ik het schadebedrijf uitbetaald krijg	1.3
	Kunnen registreren	Zodat ik kan inloggen	1.4
Verzekeraar	Kunnen zien welke ingediende claims er zijn	Ik de status hiervan kan controleren	2.1
	Mijn stem kunnen registreren op een openstaande claim	Zodat deze vervolgens goedkeuring of afkeuring ontvangt	2.2
Makelaar	Kunnen zien welke polissen er geregistreerd zijn	Zodat verzekerde claims kunnen indienen	3.1
	Een nieuwe polis kunnen registreren	Zodat verzekerde claims kunnen indienen	3.2
Admin	Makelaars registreren	Zodat deze kunnen inloggen op het systeem	4.1

Figuur 4.1: User stories die de functionele requirements defineren.

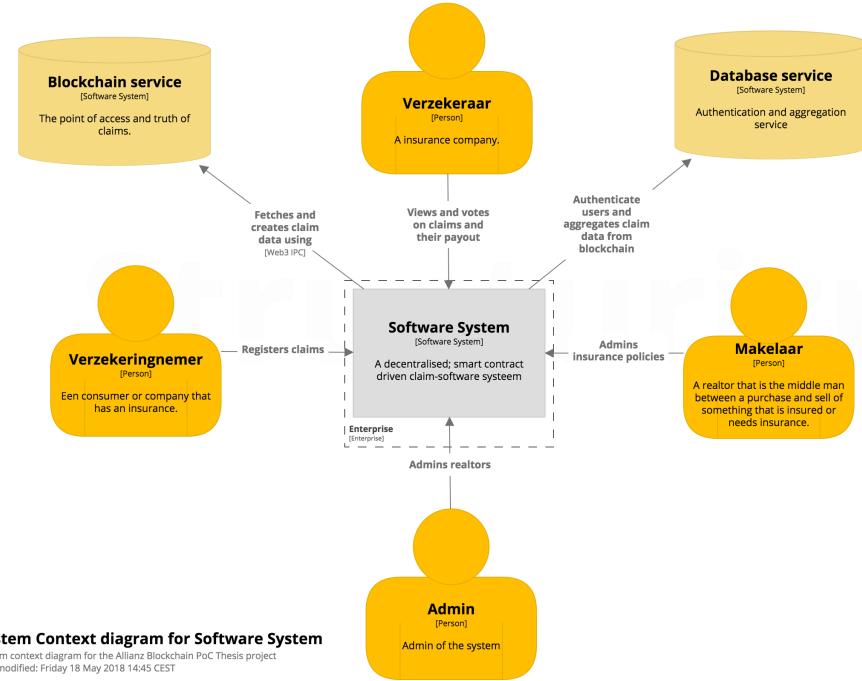
4.1.1 Overige Requirements

In de onderstaande opsomming staan overige requirements die ook aan het PoC worden gesteld.

- **R1.** Het systeem geeft gebruikers toegang op basis van hun email en wachtwoord.
- **R2.** Het systeem controleert bij een claim aanvraag van of het polis nummer valide is
- **R3.** Het systeem moet automatisch goedkeuring geven voor een claim als het claimbedrag onder 1000 euro zit en het type diefstal is.
- **R4.** Met gebruik van smart contracts en de blockchain wordt de data integriteit gewaarborgd.
- **R5.** Een verzekeraar kan bij het aanmaken van een nieuwe polis aangeven wat de verdeling is tussen de verzekeringsmaatschappijen.
- **R6.** Het systeem ondersteund Ruitschade, Brandschade, Stormscade en diefstal als types voor een claim aanvraag.
- **R7.** Een claim is via de blockchain te verifiëren.
- **R8.** Nadat alle verzekeraars een claim aanvraag hebben goed gekeurd word de claim automatisch op status goedgekeurd gezet.
- **R9.** Een claim kan alleen de open, in behandeling, goedgekeurd, afgewezen en automatisch goedgekeurd statussen hebben.
- **R10.** Het systeem geeft per claim aan op hoeveel goedkeuringen het wacht en nog nodig heeft.
- **R11.** Wanneer een verzekeraar een claim weigert word de claim automatisch op status afgewezen gezet.
- **R12.** A transaction should not take more than 5 minutes.
- **R13.** Het systeem slaat de claims gedecentraliseerd op. Er zijn dus geen centrale punten van controle.
- **R14.** Het systeem schaalt met de groei van gebruikers mee zonder dat het systeem langzamer word.
- **R15.** Gebruikers hebben interactie met het systeem via een web interface.

4.1.2 Context - Actors en hun High-level Use Cases

In het onderstaande diagram zijn de actors te zien. Het laat zien hoe deze gebruikers/systemen met het systeem communiceren.



Figuur 4.2: C4 - Context.

In de onderstaande tabel worden de actors beschreven.

Actor	Beschrijving
Verzekeringnemer	Een consument of bedrijf die een verzekering aanneemt.
Verzekeraar	Een verzekeringsmaatschappij of verzekeraar is een bedrijf dat verzekeringen aanbiedt.
Makelaar	Een makelaar bemiddelt bij de koop en verkoop of huur en verhuur van onroerend goed.
Admin	Beheerder van het systeem
Blockchain dienst	Opslaan van transacties tussen gebruikers van het systeem
Database dienst	Authenticatie van gebruikers en aggregatie van claim data

Figuur 4.3: Actors in het systeem

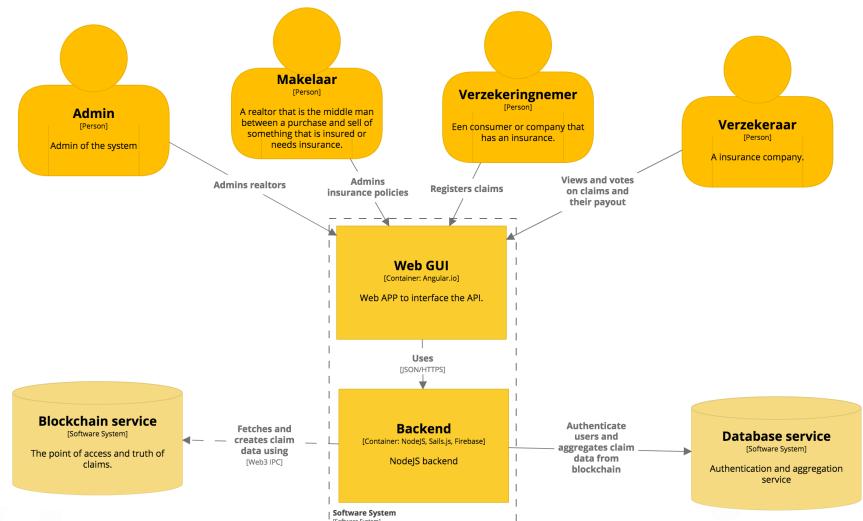
4.2 Decision forces view

Een decision force is een term die ik tijdens het Advanced Software Development semester heb geleerd. Een force is een beslissingsfactor van een architectureel probleem dat zich in het systeem of zijn omgeving. In de context van dit project is het de blockchain technologie keuze waarop het proof of concept op wordt gebouwd. Deze decision forces kunnen vanuit viewpoints komen, zoals: operationeel, ontwikkeling, zakelijk, organisatorisch, politiek, economisch, juridisch, regelgevend, ecologisch, sociaal [18]. Uiteindelijk word een decision force gekoppeld aan een system quality attribute die de verschillende concerns aantoot. Hiermee worden uiteindelijk de verschillende technologieën vergeleken. De volgende decision forces zijn in samenwerking met Allianz, de stakeholder van het proof of concept samengesteld naast de requirements die in paragraaf 4.1 zijn behandeld.

- **F1.**
- **F2.**
- **F3.**
- **F4.**
- **F5.**
- **F6.**
- **F7.**
- **F8.**

Zie de bijlagen [?] waar de decision forces view te zien is.

4.3 Container diagram



Container diagram for Software System
Container diagram for the Allianz Blockchain PoC Thesis project
Last modified: Friday 18 May 2018 15:31 CEST

Figuur 4.4: C4 - Containers.

HOOFDSTUK 5

BEVINDINGEN VAN HET PROOF OF CONCEPT

Dit hoofdstuk presenteert de bevindingen die zijn waargenomen tijdens het uitvoeren van het experiment waaruit de vraag “*Welke kansen & knelpunten bestaan bij het toepassen van de blockchain?*” (**SRQ3** uit paragraaf 1.3) word beantwoord.

De smart contracts maken het grootste deel uit van de core business logica en autorisatie. Aangezien de korte projectduur van 3 maanden en het overvloed aan bestaande blockchain en smart contract implementaties is er gekozen om geen blockchain te programmeren.

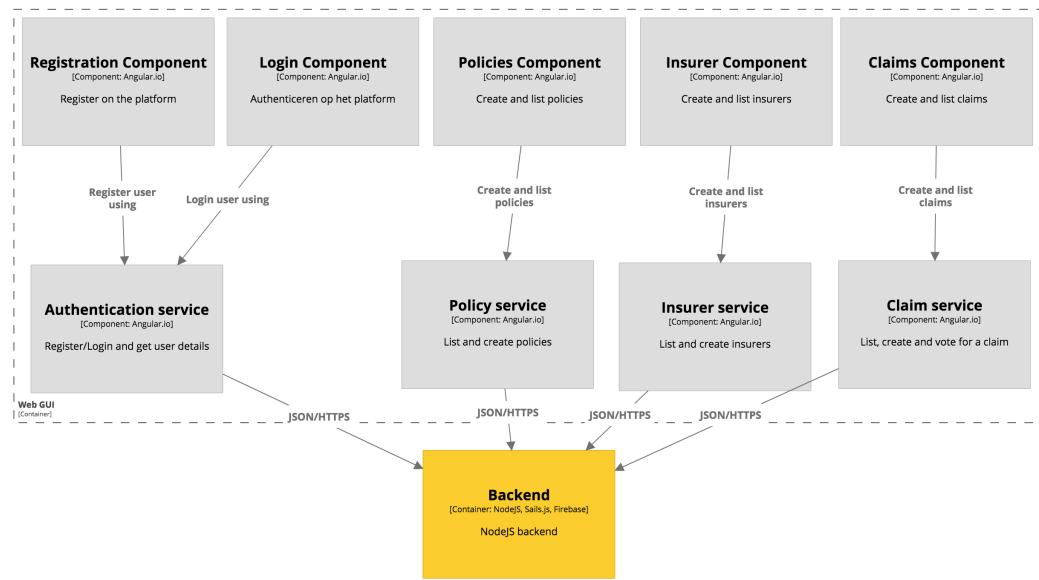
5.1 The good

- Reliability en speed is top.
- Traceability van transacties is top.

5.2 The bad

- Technologie is nog niet mature genoeg.
- Smart contracts nog te beperkt.

5.3 Het resultaat



Figuur 5.1: C4 - Components.

HOOFDSTUK 6

CONCLUSIE

BIBLIOGRAFIE

- [1] Blockchain technologies for business. URL <https://www.hyperledger.org/>.
- [2] The linux foundation. URL <https://www.linuxfoundation.org/>.
- [3] Multichain. URL <https://www.multichain.com/>.
- [4] Andreas M. *Mastering Bitcoin (Second Edition, Second Print): Programming the Open Blockchain*. June 2017. URL <https://github.com/bitcoinbook/bitcoinbook/blob/develop/book.asciidoc>.
- [5] Arthur Gervais. *On the security and performance of proof of work blockchains*. 2016.
- [6] Steven Gort Bas Kaptijn, Peter Bergman. *Blockchain*. 2016. URL https://www.ictu.nl/sites/default/files/documents/ICTU_Whitepaper_Blockchain.pdf.
- [7] Vitalik Buterin. On public and private blockchains, 2015. URL <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>.
- [8] Calum Iain Munro. *Plan van Aanpak - Afstudeerproject: Allianz*. 2018.
- [9] Tim Minshall Thas Nirmalathas Zoran Perunovic Ikhlaq Sidhu Dhrubes Biswas, Charlotta Johnsson. Applied innovation review, issue no. 2 june 2016. 2016.
- [10] Alex Tapscott Don Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Portfolio / Penguin, 2016.
- [11] DR. GAVIN WOOD. *ETHEREUM: A SECURE DE-CENTRALISED GENERALISED TRANSACTION LEDGER*. 2014. URL <https://bravenewcoin.com/assets/Whitepapers/Ethereum-A-Secure-Decentralised-Generalised-Transaction-Ledger-Yellow-Paper.pdf>.
- [12] Tzu-Chun Liao Iuon-Chang Lin. *A Survey of Blockchain Security Issues and Challenges*. Department of Photonics and Communication Engineering, Asia University, 2017. URL <http://ijns.jalaxy.com.tw/contents/ijns-v19-n5/ijns-2017-v19-n5-p653-659.pdf>.
- [13] Kieren James-Lubin. Blockchain scalability: A look at the stumbling blocks to blockchain scalability and some high-level technical solutions. URL <https://www.oreilly.com/ideas/blockchain-scalability>.
- [14] John Domingue Matthew English, Soren Auer. *Block Chain Technologies The Semantic Web: A Framework for Symbiotic Development*. Computer Science Conference for University of Bonn Students, J. Lehmann, H. Thakkar, L. Halilaj, and R. Asmat, Eds, 2016. URL <https://pdfs.semanticscholar.org/2fd3/7fed17e07c4ec04caef7dcacb16670fa2d8.pdf>.

- [15] Nakamoto, Satoshi. *Bitcoin: A peer-to-peer electronic cash system*. 2008.
- [16] Nick Szabo. Smart contracts, 1994. URL <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smарт.contracts.html>.
- [17] Stan Higgins. Inside r3cev's plot to bring distributed ledgers to wall street. URL <https://www.coindesk.com/r3cev-distributed-ledger-wall-street/>.
- [18] Rich Hilliard Uwe van Heesch, Paris Avgeriou. *Forces on Architecture Decisions*. 2012. URL <http://www.cs.rug.nl/paris/papers/WICSA12b.pdf>.
- [19] Pavel Vasin. *BlackCoin's Proof-of-Stake Protocol v2*. URL <http://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf>.
- [20] Vitalik Buterin. *A NEXT GENERATION SMART CONTRACT DECENTRALIZED APPLICATION PLATFORM*. URL http://www.the-blockchain.com/docs/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

HOOFDSTUK 7

BIJLAGEN

BIJLAGE A

DECISION FORCES VIEW

			<Approved>	<Rejected>	<Approved>	<Rejected>	<Rejected>	<Rejected>	<Approved>
	Description	Concerns	AngularJS 2	React	NodeJS (SailJS)	Java (Spring)	Bitcoin (Blockchain)	MySQL (Centralised)	Ethereum (Blockchain)
Requirements									
R1	Access through email/password	Authenticity							
R2	Validate policy number	Functional completeness							
R3	Automatic claim acceptance	Scalability							
R4	Data integrity	Operability							
R5	Ability to supply insurance ratio	Functional completeness							
R6	Ability to set type of claim	Functional completeness							
R7	Transaction traceability	Authenticity							
R8	System auto sets status if all insurers vote yes	Functional completeness							
R9	Claim can only have certain states	Operability							
R10	System tells user the status of numer of votes	Functional completeness							
R11	System auto sets status if one insurer votes no	Functional completeness							
R12	Transaction should take less than 300 seconds	Performance				-	++	+	
R13	Claims are saved decentralised	Usability					--	++	
R14	System doesn't get slow when it grows	Availability					--	--	
R15	Users use application through web interface	Usability							
Other forces									
F1-Experience	Development time	++							
F2-Smart contracts	Resource utilisation				--		-	++	
F3-Is it free to use?	Costs				+	-		+	
F4-Docummentation	Development time			++	-				
F6-OS-independent	Compatibility			++	++				
F7-Widely used library	Supportability		++		-				
F8-Blockchain API available	Compatibility		++	-					

Figuur A.1: Decision Forces View.

BIJLAGE B

SMART CONTRACT: CLAIMS

```
1 pragma solidity ^0.4.15;
2
3 contract Claims {
4
5     // Claim struct
6     struct Claim {
7         bytes32 id;
8         string policyId;
9         ClaimTypes claimType;
10        uint amount;
11        State state;
12        bool exists;
13        bytes32[] insurers;
14    }
15
16    struct Vote {
17        bool value;
18        bool hasVoted;
19        bool canVote;
20    }
21
22    enum State {New, Pending, Approved, Rejected, AutoApproved}
23    enum ClaimVote {Approved, Rejected}
24    enum ClaimTypes {NA, GlasDamage, FireDamage, StormDamage, Theft}
25
26    // Who owns this contract
27    address private authority;
28
29    // Claim[] claims;
30
31    mapping(bytes32 => Claim) claims;
32    mapping(bytes32 => mapping(bytes32 => Vote)) claimVotes;
33
34    // Map of claims for addresses
35    mapping(bytes32 => uint) claimsIndexMapping;
36    mapping(address => bytes32[]) addressClaimsMapping;
37
38    event newClaim(bytes32 id, string policyId, ClaimTypes claimType, uint
39                    amount);
40    event claimApproved(bytes32 id);
41    event claimRejected(bytes32 id);
42    event claimAutoApproved(bytes32 id);
43    event logVote(bool value, bool hasVotes, bool canVote);
```

```

43
44     // Construct me
45     function Claims() public {
46         authority = msg.sender;
47     }
48
49     function createClaim(bytes32 id, string policyId, ClaimTypes claimType, uint
50         amount, bytes32[] insurers) public {
51         require(!hasClaim(id));
52         var claim = Claim(
53             id,
54             policyId,
55             claimType,
56             amount,
57             State.New,
58             true,
59             insurers
60         );
61
61         for (uint i = 0; i < insurers.length; i++) {
62             claimVotes[id][insurers[i]] = Vote(false, false, true);
63         }
64
65         claims[id] = claim;
66         addressClaimsMapping[msg.sender].push(id);
67
68         newClaim(id, policyId, claimType, amount);
69         if (shouldAutoApprove(claim)) {
70             claim.state = State.AutoApproved;
71             claims[id] = claim;
72
73             claimAutoApproved(claim.id);
74         }
75     }
76
77     function getClaim(bytes32 id) public constant returns (
78         string policyId,
79         ClaimTypes claimType,
80         uint amount,
81         uint state
82     ) {
83         require(hasClaim(id));
84         var claim = claims[id];
85
86         policyId = claim.policyId;
87         claimType = claim.claimType;
88         amount = claim.amount;
89         state = uint(claim.state);
90     }
91
92     function vote(bytes32 claimId, bytes32 insurerId, bool vote) public {

```

```

93     require(hasClaim(claimId));
94
95     var claim = claims[claimId];
96     claim.state = State.Pending;
97     claims[claimId] = claim;
98
99     var voteObject = claimVotes[claimId][insurerId];
100    require(!voteObject.hasVoted);
101
102    voteObject.value = vote;
103    voteObject.hasVoted = true;
104    claimVotes[claimId][insurerId] = voteObject;
105
106    if (hasVoteFinished(claimId)) {
107        var approved = true;
108        for (uint i = 0; i < claim.insurers.length; i++) {
109            voteObject = claimVotes[claimId][claim.insurers[i]];
110
111            if (!voteObject.value) {
112                approved = false;
113                i = claim.insurers.length + 1;
114            }
115        }
116
117        if (approved) {
118            claim.state == State.Approved;
119            claimApproved(claimId);
120        } else {
121            claim.state == State.Rejected;
122            claimRejected(claimId);
123        }
124
125        claims[claimId] = claim;
126    }
127 }
128
129 function setClaimState(bytes32 id, State state) internal {
130     var claim = claims[id];
131
132     claim.state = state;
133     claims[id] = claim;
134 }
135
136 function shouldAutoApprove(Claim claim) internal constant returns (bool) {
137     return (
138         (claim.claimType == ClaimTypes.Theft && claim.amount < 1000)
139     );
140 }
141
142 function hasClaim(bytes32 id) internal constant returns (bool) {
143     Claim storage claim = claims[id];

```

```
144     return claim.exists == true;
145 }
146
147 function hasVoteFinished(bytes32 claimId) internal constant returns (bool) {
148     var claim = claims[claimId];
149     uint votes = 0;
150     for (uint i = 0; i < claim.insurers.length; i++) {
151         var vote = claimVotes[claimId][claim.insurers[i]];
152
153         if (vote.hasVoted) {
154             votes++;
155         }
156     }
157
158     return votes == claim.insurers.length;
159 }
160 }
```