

Architectuur Document

Pedro Marques | s31 | fhict

# Documenthistorie

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Versie |  |  | Wijzigingen | Auteur | Datum | Verzendlijst | Verzoek |
| 0.1 |  |  | Opzet document, Inleiding, Domeinmodel | Pedro  Marques | 27-11 |  |  |

Inhoud

[Documenthistorie 1](#_Toc499678052)

[Inleiding 5](#_Toc499678053)

[Context van het project 5](#_Toc499678054)

[Applicatie 6](#_Toc499678055)

[Niet-functionele eisen 7](#_Toc499678056)

[Performance 7](#_Toc499678057)

[Betrouwbaarheid 7](#_Toc499678058)

[Compatibiliteit 7](#_Toc499678059)

[Beveiligbaarheid 7](#_Toc499678060)

[Onderhoudbaarheid 7](#_Toc499678061)

[Doel van dit document 7](#_Toc499678062)

[Domeinmodel 8](#_Toc499678063)

[Klassendiagram 8](#_Toc499678064)

[Klassenuitleg 8](#_Toc499678065)

[H3 Opdeling in Componenten 10](#_Toc499678066)

[Component diagram 11](#_Toc499678067)

[Koppeling tussen componenten 12](#_Toc499678068)

[Synchronisatie tussen componenten 12](#_Toc499678069)

[Services per component 12](#_Toc499678070)

[Allocatie van objecten 13](#_Toc499678071)

[Remote objecten 13](#_Toc499678072)

[Packagestructuur 13](#_Toc499678073)

[H4 Communicatie tussen componenten 14](#_Toc499678074)

[Persistentie per component 15](#_Toc499678075)

[Persistentie binnen CryptoHelper 15](#_Toc499678076)

[Data opslag tijdens gebruik van CryptoHelper 15](#_Toc499678077)

[H6 Detailed design per component 16](#_Toc499678078)

[H7 Realisatie niet-functionele eisen 17](#_Toc499678079)

[H8 Deployment 18](#_Toc499678080)

[H9 Specificatie van interfaces 19](#_Toc499678081)

# Inleiding

## Context van het project

Om de waarde van mijn project duidelijk uit te leggen het is belangrijk om te snappen wat blockchain technologie is.

**Blockchain technologie** is een soort van databasesysteem (heel primitief gezegd) dat gebruikt kan worden om waarde (geld of digitale items) te sturen naar een derde partij op een veilige manier, zonder een middenman (iemand dat de transactie voor je uitvoert – denk aan banken of services zoals PayPal).

Tegenwoordig gebruiken mensen middenmannen omdat ze de derde partij aan wie het sturen van goederen is bedoeld niet vertrouwen. Middenmannen geven de zekerheid dat een transactie werd uitgevoerd, zodat de derde partij deze informatie niet kan falsificeren - bijvoorbeeld door het communiceren dat het overgeschreven bedrag niet genoeg of helemaal niet binnen gekomen is. Het klinkt allemaal goed en nuttig, maar dit betekent ook dat banken en zulke instituties ook controle hebben over het uitvoeren van transacties en kunnen dus servicekosten invoeren of zelfs transacties blokkeren.

Blockchain technologie betekent dus een “bypass” op de middenman bij het uitvoeren van een transactie, onder andere voordelen, zoals:

* Het publiek en transparant stellen van alle transacties die worden gemaakt door een gebruiker. Bijvoorbeeld: In het geval van een overheidsinstitutie is dit nuttig, omdat dan kan iedereen zien waar er geld wordt besteden. (Blockchain records zijn voor iedereen beschikbaar).
* Het makkelijk kunnen verifiëren van eigendom.
* Toepassing op letterlijk alle services die tegenwoordig bestaan. (Er bestaat tegenwoordig een blockchain project voor bijna alles – verkoop van games (Enjin), het maken van andere blockchain projecten (ARK), anoniem dingen kopen (Monero), etc.

**Cryptocurrency** is een elektronische eenheid dat wordt “gecreëerd” op het moment dat een miner (iemand die zijn computer gebruikt om te contribueren voor de werking van een blockchain network) een transactie confirmeert.

Een miner confirmeert een transactie door een block (een series wiskundige problemen) op te lossen met behulp van een krachtige computer. Een transactie is altijd geassocieerd aan een block, en het aantal confirmaties van een transactie is de hoeveelheid aangemaakte blocks sinds de block die geassocieerd is aan de transactie. Een transactie moet tenminste 6 keer worden geconfirmeerd om als geldig te worden beschouwd.

## Applicatie

Tegenwoordig ervaart de cryptocurrency markt een explosieve groei. Bedrijven beseffen steeds meer de potentieel dat blockchain technologie te bieden heeft. Dit betekent dat er steeds meer significante investeringen in deze technologieën worden gemaakt, wat ook betekent dat er prijsfluctuaties ontstaan in de cryptocurrency markt.

Crytophelper is een project met als doel het maken van een gedistribueerde app dat gebruikt kan worden om:

* **Geïnformeerde investeringen binnen het cryptocurrency wereld te maken;**
  + Cryptohelper beschikt van een overzicht scherm die alleen maar **relevante** cryptocurrencies op de markt aan de gebruiker toont. Door informatie (prijzen, trends, etc.) over de cryptocurrencies aan de gebruiker te presenteren en alles bij elkaar in een fatsoenlijke GUI te presenteren wordt het maken van een slimme investering makkelijker. Daardoor wordt de kans om winst te maken ook groter.
* **De gebruikers portfolio te controleren en een overzicht daarvan te geven;**
  + Een overzicht over welke investeringen die al gemaakt zijn is noodzakelijk om goede beslissingen te maken. Cryptohelper helpt hierbij door dit makkelijk zichtbaar aan de user te stellen, zodat hij/zij makkelijk een accuraat analyse van de situatie van zijn portfolio kan maken.
* **Up to date te blijven met de laatste nieuws over de crypto wereld.**
  + Cryptohelper beschikt van een News tab waar relevante nieuws over cryptocurrency aan de user worden getoond. Vaak hebben gebeurtenissen in de cryptowereld een heel opmerkelijk invloed op de prijs van cryptocurrencies en hun respectieve market cap.

Cryptohelper is een driedelige product: Het houdt in een server die informatie vanuit API rest calls haalt voor gebruik in de client, de client zelf en een database component die verbinding maakt met de database server (DAL).

## Niet-functionele eisen

De volgende niet-functionele eisen zullen worden besproken in deze document:

### Performance

Alle netwerk-afhankelijk handelingen worden met een vertraging van maximaal 1 seconden uitgevoerd (URS. Q.1). Daarnaast worden alle GUI handelingen uitgevoerd met een vertraging van maximaal 500ms (URS. Q.2).

### Betrouwbaarheid

Het systeem moet 99% van de tijd beschikbaar zijn (URS Q.6). Per 24 uur moet het systeem 23 uur functioneel blijven (URS Q.7).

### Compatibiliteit

Documentatie van zowel interfaces als klassen is beschikbaar (URS Q.4)

### Beveiligbaarheid

Het systeem maakt gebruik van beveiligde verbindingen. (Q.8) Bovendien delen de componenten van het systeem geen niet-noodzakelijke klassen/gegevens met elkaar. (Q.9)

Database query’s worden vooraf bereid in het systeem om injectie problemen te voorkomen. (Q.10)

### Onderhoudbaarheid

Het systeem is modulair opgebouwd.(Q.11)

## Doel van dit document

Het doel van deze document is het verantwoorden van de software/hardware architectuur van de CryptoHelper applicatie.

De volgende aspecten van de architectuur van CryptoHelper zullen worden beschreven:

* Domeinmodel
* Persistentie
* Communicatie
* Realisatie van niet-functionele eisen
* Componenten
* Deployment
* Specificatie van Interfaces

# Domeinmodel

## Klassendiagram

Figuur 1: Klassendiagram van het domein.

### Klassenuitleg

**Coin:**

Representeert een cryptocurrency eenheid. Wordt vanuit een API call gevuld. Een coin heeft de volgende attributen:

* MarketName: de naam van de coin op de markt;
* High: de hoogste prijs van de coin op de uur waar de informatie werd aangevraagd;
* Low: de laagste prijs van de coin op de uur waar de informatie werd aangevraagd;
* Volume: de waarde van alle transacties van de blockchain waar de coin bij hoort in de laatste 24h;
* BaseVolume: de volume van de basis currency (BTC) in de laatste 24h;
* Last: de hoogste prijs van de coin in de laatste uur;
* TimeStamp: de tijdstip waarop de Coin informatie werd aangevraagd
* Bid: de hoogste koopprijs op de markt;
* Ask: de laagste verkooprijs op de markt
* OpenBuyOrders: hoeveel koop aanvragen actief zijn voor de coin;
* OpenSellOrders: hoeveel verkoop aanvragen actief zijn voor de coin;
* PrevDay: de hoogste prijs van de coin in de laatste dag;
* Created: de datum van wanneer de coin op de markt was gezet;

**Portfolio:**

Representeert de totale opsomming van alle investeringen dat de gebruiker heeft gemaakt. Bestaat uit een map van coins, een id, een double (totalValue) om de totale waarde van de portfolio op te slaan en een lijst van transacties.

Een portfolio kan zijn totale waarde berekenen, de algemene verandering in waarde berekenen en coins toevoegen of verwijderen door middel van het aanmaken van transacties.

**Transaction:**

Representeert een verandering in de hoeveelheid coins in het portfolio. Bestaat uit een type (E.TransactionType) die aangeeft of het een verkoop of een koop transactie is, een id en het bedrag coins waar de transactie over gaat. Transacties worden opgeslagen in een Portfolio.

**NewsPiece**:

Representeert een nieuws artikel. Bevat een titel en inhoud, allebei als string.

**MainViewController**:

Handelt alle transities tussen schermen.

**PortfolioViewController**:

Beheert alle GUI logica wat met portfolio informatie te maken heeft. Dit houdt in informatie tonen, functies van Portfolio roepen, transacties initialiseren en transacties bekijken.

**NewsViewController**:

Beheert alle GUI logica wat met de opgehaalde nieuws te maken heeft; Dit houdt in alle nieuws organiseren in een lijst en focus geven aan een geklikte nieuwsstuk. Tijdens focus modus wordt er een completere nieuws aan de gebruiker getoond.

**CoinBrowserController:**

Beheert alle GUI logica wat met de opgehaalde coins te maken heeft. Dit houdt in alle coins in een lijst weergeven en focus geven aan een geklikte coin. Tijdens focus modus wordt er een completere weergave van informatie over een coin aan de user getoond.

# H3 Opdeling in Componenten

*In dit hoofdstuk wordt de opdeling in software componenten besproken aan de hand van een Componentendiagram met toelichting. Daarnaast wordt koppeling en synchronisatie tussen de componenten besproken en de opdeling in software packages.*

*Een component is een modulair deel van het systeem dat beschreven wordt in termen van aangeboden en gevraagde interfaces. In principe kan een component vervangen worden door een andere component, mits die andere component aan dezelfde interfaces voldoet. Er wordt onderscheid gemaakt tussen componenten en subsystemen: een component is altijd een executeerbare eenheid, maar een subsysteem is geen executeerbare eenheid. Tip: kies voor iedere component een naam die duidelijk verschilt van de namen van de klassen die in die component zitten.*

*De volgende onderwerpen worden in dit hoofdstuk besproken:*

* *Componentendiagram (opdeling componenten)*
* *Soort koppeling (RMI, HTTP, SQL, …)*
* *Synchronisatie (zie ook persistentie/communicatie)*
* *Service(s) per component (voor welke component interessant; aanduiding van behoeften (naam methode) per interface)*
* *Allocatie objecten (binnen welke component)*
* *Remote objecten (welke objecten worden op afstand aangesproken)*
* *Packagestructuur*

*Er zijn verschillende manieren om componentdiagrammen te tekenen, zelfs binnen UML. Hier is gekozen voor componenten opgebouwd uit klassen en interfaces. Klassen kunnen interfaces realiseren of hebben behoefte aan interfaces. Deze interfaces kunnen binnen een component met elkaar verbonden worden en/of via een zogenaamde poort beschikbaar worden gesteld aan andere componenten. Een poort (engels: port) wordt weergegeven door middel van een vierkantje op de rand van de component. Een poort kan een interface aanbieden (aangeboden poort) of behoefte hebben aan een interface (benodigde poort). Een poort die zowel een interface aanbiedt als een interface nodig heeft noemen we een complexe poort. Meestal wordt een poort gerealiseerd door middel van een object dat de aanvraag van services doorgeeft naar een ander object binnen de component. Tijdens uitvoering van het programma zal een benodigde poort een verbinding leggen met een aangeboden poort van een andere component. Voor meer informatie over componentendiagrammen, zie Hoofdstuk 14 van Praktisch UML van Jos Warmer en Anneke Kleppe (vijfde editie, uitgever Pearson).*

## Component diagramC:\Users\Pedro\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Component Diagram.jpg

Figuur 2: Componentendiagram.

Het componentendiagram in Figuur 2 laat de 4 componenten van CryptoHelper zien met de klassen en interfaces die ze aanbieden/ontvangen. De componenten zijn:

* Database Server
  + Representeert de software applicatie die alle communicatie met de database behandelt. Bevat de class Portfolio.
* Coin Server
  + Representeert de software applicatie die de cryptocurrency markt informatie ophaalt en vervolgens beschikbaar maakt. Bevat de class Wallet.
* News Server
  + Representeert de software applicatie die e nieuws ophaalt en vervolgens beschikbaar maakt. Bevat de class NewsFeed.
* CryptoHelper Client
  + Representeert de software applicatie voor de gebruiker. Bevat de classes Database Client, Coin Client en News Client.

## Koppeling tussen componenten

Om real-time communicatie uit te voeren worden er twee methodes toegepast: het sturen van TCP pakketten met requests en responses (meestal tussen Client en Database om in te loggen, sessies te refreshen, etc.) , en Remote Method Invocation (RMI). De koppelingen tussen componenten worden gerealiseerd door gebruik te maken van drie interfaces: IPortfolio, IWallet en INewsFeed.

Hier zal er voornamelijk over RMI gesproken worden, omdat de consensus op de FHICT is dat componentendiagrammen alleen componenten, interfaces en klassen toont. Voor meer informatie over niet-RMI communicatie zie hoofdstuk 4.

Alle servers worden gekoppeld aan de bijbehorende Client class in CryptoHelper Client. Ik heb hiervoor gekozen om aan de Single Responsability Principle te houden. Zo blijft het programma modulair en worden de data streams (portfolio info, coin info en news info) in aparte klassen gehouden. De classes in elke Client worden dan waar nodig in het programma gebruikt. Dit wordt niet in het diagram opgenomen om het schema overzichtelijk te houden.

## Synchronisatie tussen componenten

Binnen de context van CryptoHelper is synchronisatie niet van belang, omdat alle servers stateless zijn. Dat wilt zeggen dat ze reageren alleen naar requests of responses, en zijn niet afhankelijk van de status van elkaar om te functioneren.

## Services per component

De volgende services worden gerealiseerd door middel van de drie interfaces in het diagram:

* IWallet
  + Vraag referentie naar lijst van Coin
  + Vraag update van lijst van Coin
* INews
  + Vraag referentie naar lijst van News
  + Vraag update van lijst van News
* IPortfolio
  + Vraag referentie naar lijst van Coin van user
  + Vraag referentie naar lijst van Transaction van user

De volgende services worden per component aangeboden:

* Coin Server: IWallet
* News Server: INewsFeed
* Database Server: IPortfolio

**In CryptoHelper de enige component die services aanvraagt is de CryptoHelper Client.** Die vraagt namelijk voor alle aangeboden services door de andere componenten.

Verder vindt log in communicatie plaats tussen de Database Server en de Crypto Client in de vorm van het uitwisselen van TCP pakketten. Voor meer informatie hierover zie de hoofdstuk Communicatie.

### Allocatie van objecten

Hier worden beschreven waar welke instanties van welke objecten zijn gealloceerd.

* Coin Server: Instantie van Wallet.
* News Server: Instantie van News Feed.
* Database Server: Instantie van Portfolio.
* CryptoHelper Client: Instanties van Database, Coin en News Client;

### Remote objecten

Alle remote objecten (Portfolio, Wallet en NewsFeed) worden in CryptoHelper Client geïnstantieerd.

### Packagestructuur

Elke component heeft zijn eigen package. Daarnaast is er een Shared package voor klassen die in meer dan een component worden gebruikt, en een Network Package voor de requests en responses tussen twee connections.

Klassen/Interfaces per package:

* Shared: Coin, NewsPiece, IPortfolio, IWallet, INewsFeed.
* Client: CoinClient, NewsClient, DatabaseClient.
* CoinServer: CoinServer, Wallet.
* NewsServer: NewsServer, NewsFeed.
* DatabaseServer: DatabaseServer, Portfolio;

# H4 Communicatie tussen componenten

*In dit hoofdstuk wordt de communicatie van en tussen objecten van verschillende componenten beschreven. Beschrijf welke eigenschappen van welke objecten moeten worden gecommuniceerd. Beschrijf hoe communicatie wordt gerealiseerd (bijvoorbeeld m.b.v Remote Method Invocation, RMI). Dit kan verschillen per eigenschap. Beschrijf tot slot wanneer communicatie plaatsvindt (push-pull, welk object is leidend?, etc.) Ook dit kan verschillen per eigenschap.*

*In het geval RMI wordt gebruikt voor communicatie tussen componenten: stel een klassendiagram op en geef per klasse/interface aan of deze UnicastRemoteObject, Remote of Serializable is. In het geval gebruik gemaakt wordt van (SOAP) webservices: stel een klassendiagram op en geef aan welke services door welke klasse worden geleverd.*

**Klassendiagram tbv communicatie**

Figuur X: Klassendiagram ten behoeve van communicatie tussen componenten.

# Persistentie per component

*In dit hoofdstuk wordt de persistentie van de objecten beschreven voor iedere component. Beschrijf welke eigenschappen van welke objecten moeten worden bewaard. Beschrijf hoe opslag wordt gerealiseerd (database, serialiseren, XML, etc.). Dit kan verschillen per eigenschap. Voeg het ERD met beschrijving toe indien gegevens worden opgeslagen in een database. Beschrijf tot slot wanneer opslag plaatsvindt. Ook dit kan verschillen per eigenschap.*

## Persistentie binnen CryptoHelper

Er is maar een component waarbij er gebruik wordt gemaakt van persistentie, de DatabaseServer. Om de login gegevens, sessie en de portfolio van een klant op te slaan wordt er gebruik gemaakt van de Account data object. Een account bevat een ID, een naam, een wachtwoord, een portfolio en een token veld om de status van een sessie op te slaan.

## Data opslag tijdens gebruik van CryptoHelper

Data opslag gebeurt binnen CryptoHelper op de volgende momenten:

**Bij het aanmaken van een account:**

Als een account wordt aangemaakt wordt het meteen opgeslagen in de database. De portfolio is dan leeg.

**Bij het inloggen:**

Om log in sessies bij te houden en ongeoorloofde gebruik tegen te gaan, wordt een token gegenereerd en opgeslagen in de database. Als iemand probeert op een account in te loggen die al ingelogd is krijgt hij/zij een error message.

**Bij het uitvoeren van transacties:**

Tijdens gebruik van CryptoHelper worden alle updates aan de portfolio van de klant opgeslagen op het moment dat ze uitgevoerd zijn. Dus elke keer dat de klant een transactie uitvoert wordt de portfolio data entry op de database meteen geüpdatet.

**Bij het uitloggen:**

Op het moment dat een user zich uitlogt wordt de login token verwijdert van de database. **Dit kan ook gebeuren als de user 20 minuten inactief is.**

# H6 Detailed design per component

*In dit hoofdstuk wordt het detailed design (implementatiemodel) per component beschreven. Maak hiervoor gebruik van klassendiagrammen en sequence diagrammen. Voor specifieke doeleinden kun je gebruik maken van objectdiagrammen, statemachinediagrammen, activiteitendiagrammen, etc. Voor meer informatie over UML diagrammen, zie Praktisch UML van Jos Warmer en Anneke Kleppe (vijfde editie, uitgever Pearson).*

# H7 Realisatie niet-functionele eisen

*In dit hoofdstuk wordt de realisatie van (overige) niet-functionele eisen besproken. De volgende onderwerpen dienen tenminste te worden besproken: betrouwbaarheid, performance, beveiliging, schaalbaarheid. Eventueel kunnen nog extra niet-functionele eisen worden besproken.*

**Betrouwbaarheid**

**Performance**

**Beveiliging**

**Schaalbaarheid**

# H8 Deployment

*In dit hoofdstuk wordt de toekenning van software componenten aan hardware besproken aan de hand van een Deploymentdiagram met toelichting. Daarnaast wordt de koppeling (lokaal netwerk, internet, etc.) tussen de nodes besproken.*

**Deploymentdiagram**

Figuur X: Deploymentdiagram.

# H9 Specificatie van interfaces

*In dit hoofdstuk wordt de specificatie van interfaces besproken. Voor ieder interface wordt per methode gedefinieerd:*

* *Naam methode*
* *Naam en type argumenten*
* *Precondities*
* *Type returnwaarde*
* *Beschrijving*
* *Aanleiding voor excepties*