

Architectuur Document

Pedro Marques | s31 | fhict

# Documenthistorie

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Versie |  |  | Wijzigingen | Auteur | Datum | Verzendlijst | Verzoek |
|  |  |  |  |  |  |  |  |
| 0.1 |  |  | Opzet document, Inleiding, Domeinmodel | Pedro  Marques | 27-11 |  |  |
| 0.2 |  |  | Componenten, Persistentie | Pedro Marques | 28-11 |  |  |
|  |  |  |  |  |  |  |  |

Inhoud

[Documenthistorie 1](#_Toc499999613)

[Inleiding 5](#_Toc499999614)

[Context van het project 5](#_Toc499999615)

[Applicatie 6](#_Toc499999616)

[Niet-functionele eisen 7](#_Toc499999617)

[Performance 7](#_Toc499999618)

[Betrouwbaarheid 7](#_Toc499999619)

[Compatibiliteit 7](#_Toc499999620)

[Beveiligbaarheid 7](#_Toc499999621)

[Onderhoudbaarheid 7](#_Toc499999622)

[Doel van dit document 7](#_Toc499999623)

[Domeinmodel 8](#_Toc499999624)

[Klassenuitleg 9](#_Toc499999625)

[Coin: 9](#_Toc499999626)

[IDatabaseHelper 9](#_Toc499999627)

[Transaction 9](#_Toc499999628)

[NewsPiece: 9](#_Toc499999629)

[ClientContainer: 9](#_Toc499999630)

[Opdeling in Componenten 10](#_Toc499999631)

[Component diagram 11](#_Toc499999632)

[Koppeling tussen componenten 12](#_Toc499999634)

[Synchronisatie tussen componenten 12](#_Toc499999635)

[Services per component 13](#_Toc499999636)

[Allocatie van objecten 13](#_Toc499999637)

[Remote objecten 14](#_Toc499999638)

[Packagestructuur 14](#_Toc499999639)

[Communicatie tussen componenten 15](#_Toc499999640)

[Communicatie Library 15](#_Toc499999641)

[Communicatie tussen componenten 15](#_Toc499999642)

[Communicatie tussen Client en Database Server 16](#_Toc499999643)

[Communicatie tussen Coin Client en Coin Server 18](#_Toc499999644)

[Communicatie tussen News Client en News Server 19](#_Toc499999645)

[Persistentie per component 20](#_Toc499999646)

[Persistentie binnen CryptoHelper 20](#_Toc499999647)

[Data opslag tijdens gebruik van CryptoHelper 20](#_Toc499999648)

[Detailed Design per Component 21](#_Toc499999649)

[Coin Server 21](#_Toc499999650)

[Database Server 22](#_Toc499999651)

[News Server 23](#_Toc499999652)

[CryptoHelper Client 24](#_Toc499999653)

[Realisatie Niet-Functionele Eisen 25](#_Toc499999654)

[Betrouwbaarheid 25](#_Toc499999655)

[Performance 25](#_Toc499999656)

[Beveiliging 25](#_Toc499999657)

[Schaalbaarheid 25](#_Toc499999658)

[Deployment 26](#_Toc499999659)

[Deploymentdiagram 26](#_Toc499999660)

[H9 Specificatie van interfaces 27](#_Toc499999661)

# 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 IDatabaseHelper 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 IDatabaseHelper 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)

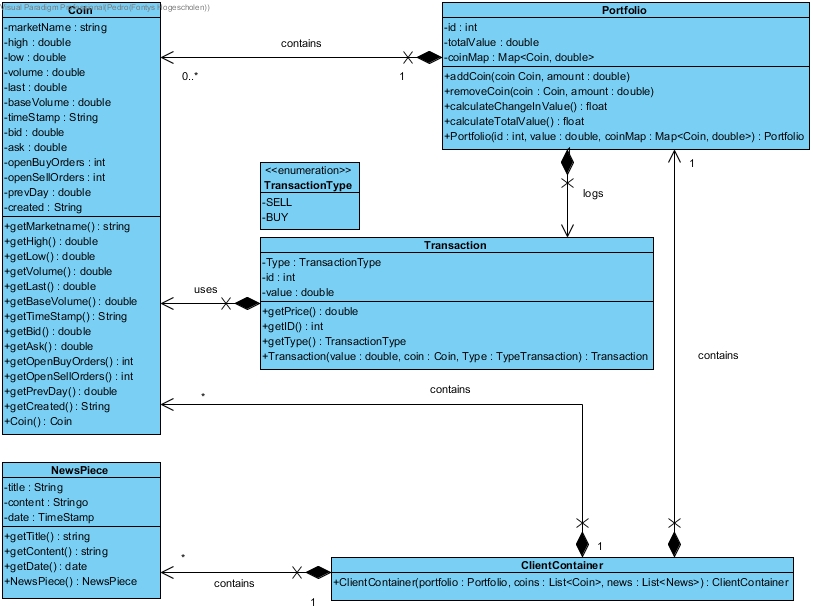
## Doel van dit document

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



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;

#### IDatabaseHelper:

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 IDatabaseHelper op te slaan en een lijst van transacties.

Een IDatabaseHelper 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 IDatabaseHelper. 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 IDatabaseHelper.

### NewsPiece:

Representeert een nieuws artikel. Bevat een titel, inhoud en een datum.

### ClientContainer:

Representeert een container voor een IDatabaseHelper, een lijst van coins en een lijst van nieuws.

# 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 diagram

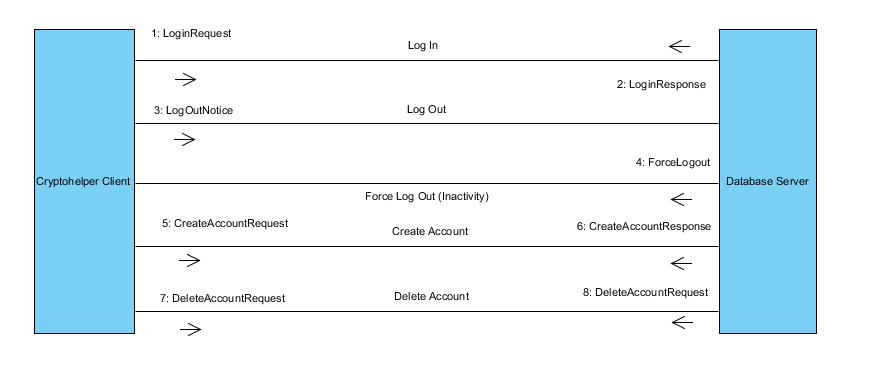
## C:\Users\Pedro\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Component Diagram.jpg

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. Dat houdt in het maken en verwijderen van accounts, het inloggen en uitloggen en het opslaan en laden van portfolio informatie. Bevat de class DatabaseHelper.
* 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 tussen Client en DatabaseServer (Figuur 3), en Remote Method Invocation (Figuur 2). De RMI koppelingen tussen componenten worden gerealiseerd door gebruik te maken van twee interfaces: ICoinRetriever en INewsFeed.

Figuur 3: Versimpelde TCP communicatiediagram tussen Database Client en Database Server.

Alle servers worden gekoppeld aan de bijbehorende Client class in CryptoHelper Client. Ik heb hiervoor gekozen om aan de Single Responsibility Principle te houden. Zo blijft het programma modulair en worden de data streams 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 tussen de Coin Server en de Client belangrijk bij het berekenen van de waarde van een portfolio. Elke keer dat een berekening van portfolio waarde plaats vindt wordt er nieuwe informatie opgehaald zodat het prijs actueel is. Bovendien vindt er een automatisch update plaats elke 5 minuten.

## Services per component

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

* ICoinRetriever
  + Vraag referentie naar lijst van Coin
  + Vraag update van lijst van Coin
* INewsFeed
  + Vraag referentie naar lijst van News
  + Vraag update van lijst van News

De volgende services worden per component aangeboden:

* Coin Server: IWallet
* News Server: INewsFeed
* Database Server: IDatabaseHelper, LogInResponse, CreateAccountResponse, DeleteAccountResponse, ForceLogout.
* Database Client , LogInRequest, CreateAccountRequest, DeleteAccountRequest

**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 CoinRetriever.
* News Server: Instantie van NewsFeed.
* Database Server: Instantie van DatabaseHelper.
* CryptoHelper Client: Instanties van DatabaseClient, CoinClient en NewsClient;

### Remote objecten

Alle remote objecten (CoinRetriever 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, IDatabaseHelper, IWallet, INewsFeed. Network (Bevat alle TCP Requests en Responses)
* Client: CoinClient, NewsClient, DatabaseClient.
* CoinServer: CoinServer, Wallet.
* NewsServer: NewsServer, NewsFeed.
* DatabaseServer: DatabaseServer, IDatabaseHelper;

# Communicatie tussen componenten

## Communicatie Library

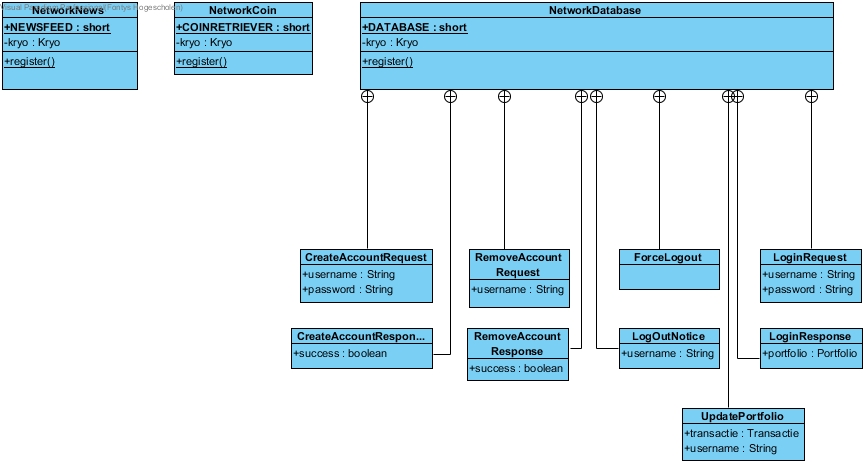
Communicatie binnen de componenten van CryptoHelper wordt gerealiseerd op twee manieren: Door middel van RMI en het gebruik van TCP communicatie.

Om dit te coderen er wordt gebruik gemaakt van de Kryonet networking library (<https://github.com/EsotericSoftware/kryonet>). Er werd hiervoor gekozen omdat Kryonet versimpelt het opstellen van RMI en TCP communicatie, en maakt gebruik van methoden die goed te combineren zijn met de observable pattern.

## Communicatie tussen componenten

Tijdens het gebruik van CryptoHelper staat de CryptoHelper Client verbonden aan de andere drie componenten (Database, News en CoinRetriever Servers). Deze drie comuniceren tussen elkaar niet. Alle servers zijn stateless.

Ik zal per component uitleggen welke eigenschappen van welke objecten worden gecommuniceerd**.**

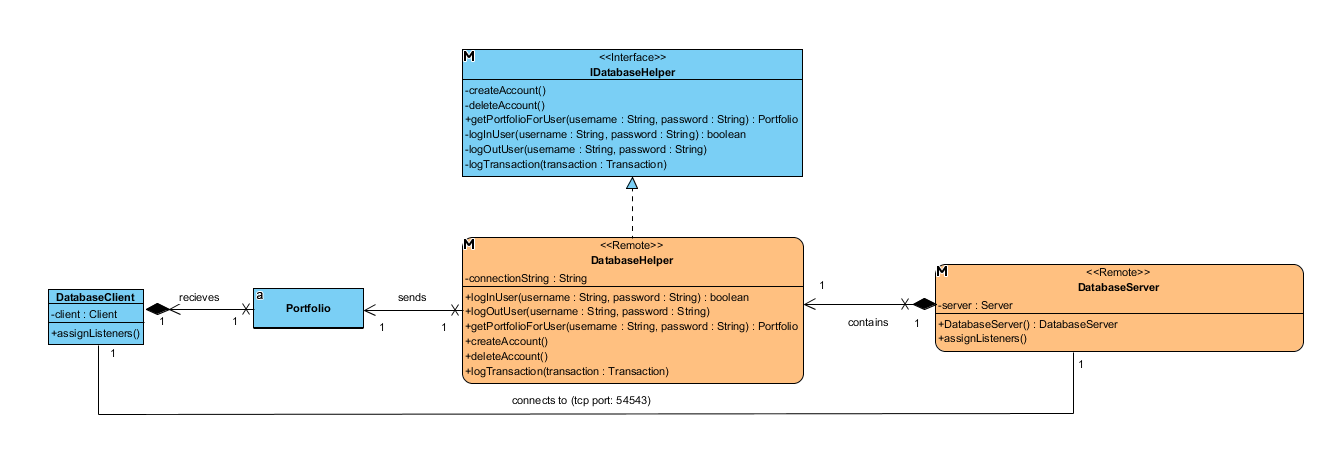


Figuur 4: Network klassen. Gebruikt om klasses/return types op de registry te registreren op twee endpoints van een verbinding, zodat ze op beide kanten van een verbinding beschikbaar zijn. Requests en responses worden als inner klassen gerepresenteerd, zoals beschreven en gesuggereerd in de Kryonet Github voorbeelden.

(<https://github.com/EsotericSoftware/kryonet/tree/master/examples/com/esotericsoftware/kryonet/examples>)

In de volgende diagrammen wordt er aangenomen dat de Client class leidend is in het realiseren van communicatie. Er zijn enkele uitzonderingen hiervoor.

## Communicatie tussen Client en Database Server



Figuur 5: Klassendiagram van communicatie tussen Client en Database Server

Communicatie tussen de Client en DatabaseServer wordt gerealiseerd door middel van uitwisseling van TCP requests en responses. De requests en responses bevatten attributen die over het netwerk verstuurd worden. In dit geval worden er strings, booleans, Transactie en Portfolio objecten verstuurd.

Het ontvangen van een request/response triggert een event in de connection endpoint waar het plaatsvindt. Deze events worden gebruikt om methoden uit te voeren binnen de klasse, om een response terug te sturen, of allebei. De Network klassen, requests en responses die hiervoor gebruikt worden staan uitgebeeld in Figuur 4.

Er vindt communicatie plaats tussen de Client en de Database Server op de volgende momenten:

Note: De uitwisseling van pakketten staat beschreven in detail in de hoofdstuk interface specificatie, in de DatabaseHandler sectie.

* Bij het inloggen van een user (Client Pull):
  + Er wordt een pakket verstuurd met een username en een password en een ontvangen met een boolean die aangeeft of de gegevens correct zijn of niet.
  + Als de login succesvol is dan krijgt de actor een bericht hierover en krijgt hij/zij toegang tot de rest van het programma. Er wordt ook een sessie token gegenereerd en een portfolio opgeladen.
  + Als de login niet succesvol is dan krijgt de actor een bericht met het fout tijdens het inloggen.
  + Bij het inloggen wordt er een token gemaakt en in de database opgeslagen. Zolang als deze token bestaat kan niemand anders zich inloggen met de net ingevoerde gegevens.
* Bij het uitloggen van een user (Client pull):
  + Er wordt een pakket verstuurd met een username bij het uitloggen in de app. Het ontvangen van het pakket triggert een event die de user uitlogt.
  + Als er geen handmatig logout wordt verstuurd dan wordt de disconnect van de client gedetecteerd door een listener op de server en wordt de client alsnog uitgelogd.
* Bij het aanmaken van een account(Client pull):
  + Net als bij het inloggen, er wordt een pakket verstuurd naar de database server met een username en een wachtword. Het ontvangen van het pakket triggert een event op de server die een methode van DatabaseHandler uitvoert om een account te maken met de gekregen gegevens.
  + Als het aanmaken van een account niet lukt (bijv. door al bestaande gegevens) krijgt de user een message met de foutmelding.
* Bij het verwijderen van een account (Client pull):
  + Er wordt een pakket verstuurd met een username. Het ontvangen van het pakket triggert een event op de server die een methode van DatabaseHandler uitvoert om een account te verwijderen met de gekregen username.
  + Als het verwijderen van een account niet lukt (geen verbinding) krijgt de user een message met de foutmelding.
* Bij het updaten van een portfolio (nieuwe transactie) (Client pull):
  + Er wordt een pakket verstuurd met een transactie en een username. Het ontvangen van het pakket triggert een event op de server die een methode van DatabaseHandler uitvoert om een transactie toe te voegen aan het portfolio van de gekregen username.
  + Als de update niet gelukt is krijgt de user een foutmelding.
* Bij het detecteren van client idling (geen activiteit) (SERVER PUSH):
  + Hierbij is de server leidend.
  + Als er geen client activiteit is gedetecteerd voor 20 minuten wordt een ForceLogOut pakket verstuurd naar de betreffende client. Het ontvangen van het pakket triggert een logout in de client.
  + De gebruiker wordt verstuurd naar het inlogscherm.

## Communicatie tussen Coin Client en Coin Server

Figuur 6: Klassendiagram van communicatie tussen CoinClient en CoinServer

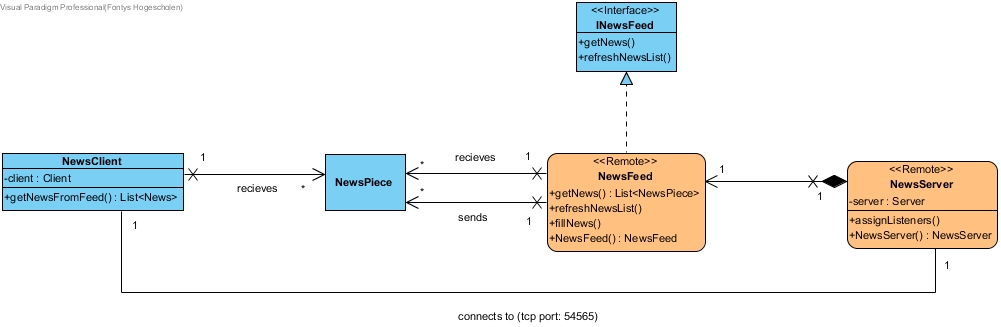
Het klassendiagram in figuur 6 representeert de relaties tussen betrokken klassen en interfaces gebruikt in het versturen van lijsten van Coin instanties. In het geval van het updaten van een lijst door de server, moeten de remote objecten beschouwd worden als lokale, en de lokale als remote.

Tijdens gebruik van CryptoHelper wordt communicatie tussen deze twee componenten gerealiseerd door middel van RMI. Markt informatie wordt opgehaald met een API call. Er worden lijsten van Coin gecommuniceerd.

Coins worden gestuurd als een client zich inlogt, een refresh aanvragt (client pull) of een automatisch server refresh actie (server push) die elk 5 minuten de informatie (JSON format) van de markt weer ophaalt via een API GET call.

Door gebruik te maken van een propertyChanged() event wordt de lijst van coins op de client automatisch geüpdatet elke keer dat de CoinRetriever zelf de methode refreshCoins() uitvoert.

## Communicatie tussen News Client en News Server



Figuur 7: Klassendiagram van communicatie tussen NewsClient en NewsServer

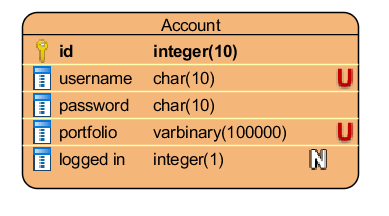
Het klassendiagram in figuur 6 representeert de relaties tussen betrokken klassen en interfaces gebruikt in het versturen van lijsten van NewsPiece instanties.

De communicatie tussen de News Client en de News Server componenten wordt gerealiseerd door middel van RMI. Hier worden alleen NewsPiece lijsten verstuurd.

Zoals bij de communicatie tussen de Coin Client en Coin Server wordt er gebruik gemaakt van de client pull principe voor als de gebruiker de newsfeed vernieuwt. Er wordt geen gebruik gemaakt van server push bij deze twee componenten.

De NewsFeed klasse gebruikt een RSS feed (Really Simple Syndication, XML bestanden uit de web) als resource om de NewsPiece instanties op te bouwen.

# Persistentie per component



Figuur 8: Database Ontwerp Diagram van dataopslag in CryptoHelper

## Persistentie binnen CryptoHelper

Om de login gegevens, sessie en de Portfolio van een klant op te slaan wordt er gebruik gemaakt van de Account data object.

Figuur 8 beschrijft hoe de Account tabel in elkaar zit: Een account bevat een unieke ID, een unieke gebruikersnaam, een wachtwoord, een uniek geserialiseerde (bytecode) Portfolio en een boolean veld om aan te geven of een gebruiker ingelogd is.

## 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 worden de username en password aan een id nummer geassocieerd en opgeslagen in de database. Een Portfolio wordt gemaakt en is leeg.

**Bij het inloggen:**

Om log in sessies bij te houden en ongeoorloofde gebruik tegen te gaan, wordt het loggedin veld in de database naar 1 gezet. Als iemand probeert met gegevens van een al ingelogde account in te loggen krijgt hij/zij een error message en wordt de toegang geweigerd.

**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 van de user op de database meteen geüpdatet.

**Bij het uitloggen:**

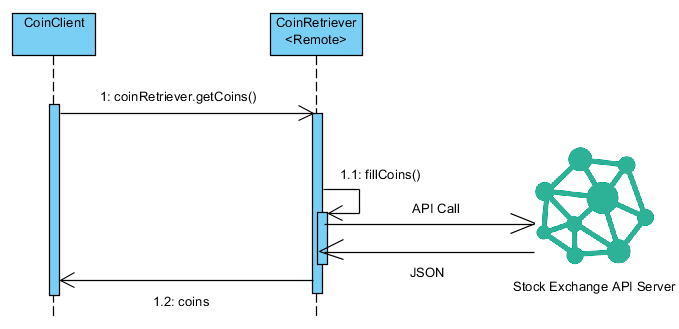
Op het moment dat een user zich uitlogt wordt de login token verwijdert van de database. **Dit kan ook gebeuren als de user 20 minuten inactief is.**

# Detailed Design per Component

Figuur 9: Klassendiagram van Coin Server component

## Coin Server

De klassendiagram in figuur 9 is de representatief voor de klassen van het Coin Server component. Coin Server heeft als doel het ophalen en versturen van een lijst van Coins naar de Client. Dit wordt gerealiseerd door gebruik te maken van de ICoinRetriever interface.

Zoals elke server component in CryoHelper, Coin Server bestaat uit een Kryonet Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, , etc.). 

Figuur 10: Sequence Diagram van het ophalen van coins bij de log in. (Client Pull)

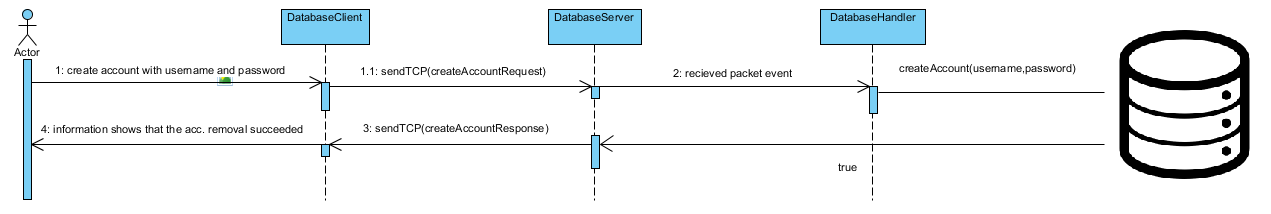
Naast de server bevat de CoinServer klasse een instantie van CoinRetriever, die een realizatie is van ICoinRetriever. Deze instantie wordt gegeven aan de client via RMI en gebruikt om lijsten van coins te sturen (Figuur 10) en vernieuwen. Bij een automatisch vernieuwing wordt de lijst coins gepusht naar de client. Vernieuwingen kunnen door de client aangevraagd worden.

## Database Server

Figuur 11: Klassendiagram van Database Server component

De klassendiagram in figuur 9 is de representatief voor de klassen van het Database Server component.

Het Database Server component heeft als doel de communicatie tussen de Client component en de database beheren. Dit houdt in het maken van parameterized queries naar een database server en het bijhouden van login sessies.

Zoals elke server component in CryptoHelper, de DatabaseServer klasse bestaat uit een Kryonet Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, etc.).

Figuur 12: Sequence diagram van het versturen en behandelen van pakketten door de Database Server in het geval van een succesvol query. removeAccount kan vervangen worden door elk andere Request van de NetworkDatabase klasse (Figuur 4, blz. 14).

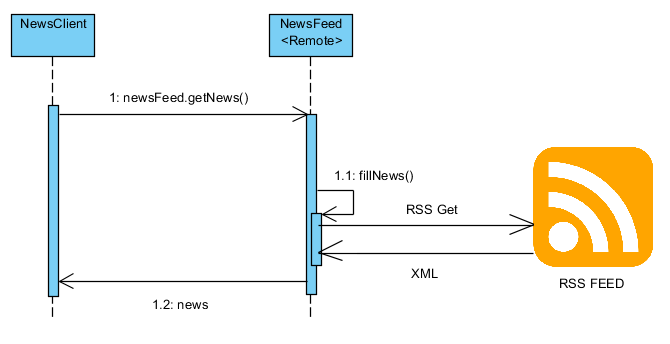
De DatabaseHelper klasse voert alle communicatie tussen database en de component in de vorm van SQL queries met parameters.. Het manier waarop de communicatie gerealiseerd wordt staat beschreven in de Communicatie hoofdstuk (blz. 15-17). De sequence diagram op Figuur 12 beschrijft hoe de event listener gebonden aan de kryonet. Server object binnen DatabaseServer reageert op het ontvangen van pakketten.

## News Server

Figuur 13: Klassendiagram van News Server component

De klassendiagram in figuur 13 is de representatief voor de klassen van het News Server component. De News Server component heeft als doel het versturen van een lijst van News naar de Client. Dit wordt gerealiseerd door gebruik te maken van de INewsFeed interface.

Zoals elke server component in CryptoHelper, de NewsServer klasse bestaat uit een Kryonet.Server object, die alle networking beheert door middel van een combinatie van het beschikbaar stellen van de server op een netwerk en event listening (events bij een nieuwe verbinding, disconnect, etc.).



Figuur 14: Sequence diagram van het ophalen van nieuws uit een RSS Feed.

Naast de server bevat de NewsServer klasse een instantie van NewsFeed, die een realisatie is van INewsFeed. Deze instantie wordt gegeven aan de client via RMI en gebruikt om lijsten van NewsPiece te sturen (Figuur 14). Vernieuwingen kunnen door de client aangevraagd worden.

## CryptoHelper Client

# Realisatie Niet-Functionele Eisen

### Betrouwbaarheid

1. *Het systeem is tenminste 99% van de tijd beschikbaar:*
   1. Door alle servercomponenten stateless te maken wordt de load van de server laag gehouden, omdat de server houdt geen instanties van client gerelateerde objecten (denk aan Portfolio per klant, etc). Het gevolg hiervan is dat alle communicatie een stuk sneller loopt. Dit leidt tot een kleinere kans tot crashes en connection overload, en dus naar een langere functioneel status.
2. *Het systeem is tenminste 1 uur per 24 uur niet beschikbaar: Zie boven*
3. *Het systeem maakt gebruik van actuele gegevens:*
   1. Door de refresh functionaliteit in de Server Componenten kunnen er continu actuele gegevens worden aangevraagd door de Klant. De Coin server bevat zelf een automatisch-refresh functionaliteit om elke 5 minuten de coin gegevens actueel te houden.
4. *Het systeem slaat veranderingen in een portfolio op:*
   1. Elke keer dat een transactie plaatsvindt wordt de betreffende portfolio opgeslagen in de database zodat er geen veranderingen kwijtgeraakt kunnen worden.

### Performance

1. *De netwerk responsetijd bedraagt minder dan 1s*
   1. Zie Betrouwbaarheid 1.a
2. *De UI wordt niet voor langer dan 500ms geblokkeerd tijdens het uitvoeren van taken*
   1. Dit wordt opgelost door threading te gebruiken tijdens het updaten van de View. Er zal altijd feedback ontstaan tijdens het uitvoeren van taken.

### Beveiliging

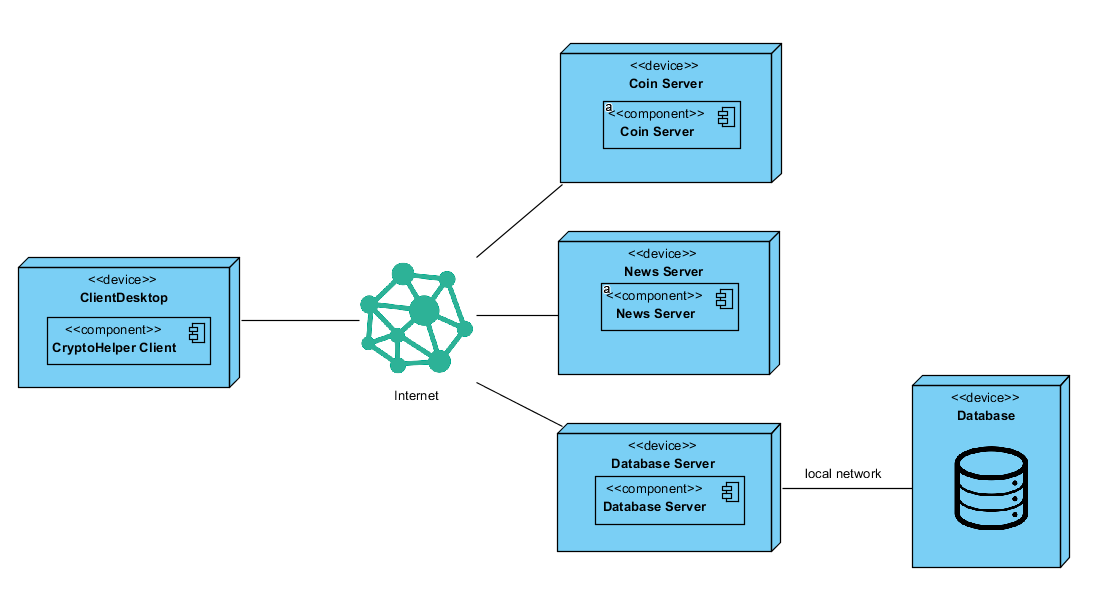
1. *Gegevens worden niet gedeeld tussen Servers:*
   1. Alle servers werken onafhankelijk van elkaar. Zo wordt de flow van informatie beperkt naar waar de informatie absoluut nodig is. Dit leidt naar minder kans op security lekken.
2. *Queries worden vooraf binnen het systeem gemaakt om SQL injectie te voorkomen:*
   1. Er wordt gebruik gemaakt van parameters tijdens het maken van database queries, zodat er geen SQL injectie mogelijk is.

### Schaalbaarheid

1. *Het systeem is modulair opgebouwd:*
   1. Omdat het systeem in verschillende componenten is verdeeld, alle componenten zijn te vervangen.
2. *Het systeem kan gebruikt worden door 20 gebruikers tegelijkertijd:*
   1. Zie Betrouwbaarheid 1.a . Door de stateless design kan de server een fatsoenlijk hoeveelheid gebruikers aan.

# Deployment

## Deploymentdiagram



Figuur 18: Deploymentdiagram.

Deployment van de gedistribueerde app is afgebeeld in Figuur 18. Het bestaat uit de volgende onderdelen:

* Een client desktop: Computer waar de CryptoHelperClient .jar executable is aanwezig.
* Coin Server: Server waar de CryptoHelperCoinServer .jar executable runt.
* News Server: Server waar de CryptoHelperNewsServer .jar executable runt.
* Database Server: Server waar de CryptoHelperDatabaseServer.jar executable runt. Deze server staat verbonden via local network met de database.

# 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*