Globaal ontwerp

**Project: IPBDAM5**

**Studiejaar: 2020/2021**

**Periode: 4**

**Groep: 7**

**Projectbegeleider/Scrummaster: Roland Westveer**

**Product Owner: Rudolf Brugman**

**Projectleden:**

* **Jelle van den Bosch, 1121244**
* **Brain Walters, 1121120**
* **Edwin van Leeuwen, 1135325**
* **Damian Alkemade, 1121187**

# Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Versie | Datum | Verandering | Auteur |
| 0.1 | 27-2-2021 | Document aangemaakt en koppen gemaakt | Jelle van den Bosch |
| 0.2 | 1-3-2021 | Kleine foutjes eruit gehaald | Jelle van den Bosch |
| 0.3 | 19-3-2021 | Requirements en data quality naar apart document overgezet | Jelle van den Bosch |
| 0.4 | 26-3-2021 | Requirements terug geplaatst, inhoudsopgave (volgorde en benaming) verbeterd | Jelle van den Bosch |
| 0.5 | 28-3-2021 | Datastructuur toegevoegd | Jelle van den Bosch |
| 0.6 | 5-4-2021 | Beheer, Systeem architectuur, voorwoord, samenvatting | Jelle van den Bosch |
| 0.7 | 7-4-2021 | Dingen | Jelle van den Bosch |
| 1.0 | 9-4-2021 | Final versie | Team |
| 1.1 | 14-5-2021 | Aanmaken nieuw bestand IPBDAM5 | Jelle van den Bosch |
| 1.2 | 18-5-2021 | Details aangepast | Jelle van den Bosch |
| 1.3 | 24-5-2021 | Kopjes aangevuld en aangepast | Jelle van den Bosch |
| 2.0 | 25-6-2021 | Laatste wijzigingen | Jelle van den Bosch |

# Samenvatting

In dit globale ontwerp zal de basis worden gelegd voor het ontwikkelen van de BI Solution voor Rudolf Brugman. Naar aanleiding van vergaderingen en interviews zullen als eerst de Business Intelligence requirements worden vastgesteld en geanalyseerd. Alle functionele en niet-functionele eisen zullen gestructureerd worden opgeschreven en gecategoriseerd doormiddel van de MoSCoW (Must have, Should have, Could have, Won’t have) methodiek. Op basis van deze eisen wordt de rest van het document opgesteld en zijn daarmee de beginselen van het BI solution. Vervolgens wordt opgesteld wat de volledige architectuur van de BI solution zal zijn. Zo wordt gekeken naar de ‘data pipeline’, naar de benodigde software, hardware, netwerkeisen en tot slot wordt omschreven hoe het datawarehouse zal worden ontworpen en ontwikkeld. Daarna wordt de BI solution zelf beschreven door te kijken naar hoe de schermen er ongeveer uit komen te zien en hoe de dagelijkse en andere benodigde processen verlopen. Tot slot wordt de data die zal worden gebruikt volledig beschreven van het moment wordt geëxtraheerd tot het wordt opgeslagen in de database.

# Inhoudsopgave

Inhoud

[Versiebeheer 2](#_Toc75544964)

[Samenvatting 3](#_Toc75544965)

[Inhoudsopgave 4](#_Toc75544966)

[1.0. Business Intelligence requirements 6](#_Toc75544967)

[1.1. User stories 7](#_Toc75544968)

[1.2. Requirements 9](#_Toc75544969)

[1.2.1. Functionele requirements 9](#_Toc75544970)

[1.2.2. Niet-functionele requirements 10](#_Toc75544971)

[2.0. Business Intelligence architectuur 12](#_Toc75544972)

[2.1. Input en output 12](#_Toc75544973)

[2.1.1. Huidige situatie 12](#_Toc75544974)

[2.1.2. Gewenste situatie – data pipeline 12](#_Toc75544975)

[2.2. Technology Stack 13](#_Toc75544976)

[2.2.1. Ontwikkeling 13](#_Toc75544977)

[2.2.2. Productie 14](#_Toc75544978)

[2.3. Pakketselectie 14](#_Toc75544979)

[2.3.1. Mogelijkheden 14](#_Toc75544980)

[2.3.2. Advies 16](#_Toc75544981)

[2.3.3. Besluit 16](#_Toc75544982)

[2.4. Systeemarchitectuur 17](#_Toc75544983)

[2.4.1. Hardware 17](#_Toc75544984)

[2.4.2. Netwerk 17](#_Toc75544985)

[2.4.3. Software 17](#_Toc75544986)

[2.5. Datawarehouse architectuur 17](#_Toc75544987)

[2.5.1 Architectuur 18](#_Toc75544988)

[3.0. Processen en ontwerp 19](#_Toc75544989)

[3.1. Wireframes 19](#_Toc75544990)

[3.2. Processen 19](#_Toc75544991)

[3.2.1. Dagelijks 20](#_Toc75544992)

[3.2.2. Controles 21](#_Toc75544993)

[4.0. Beheer 23](#_Toc75544994)

[4.1. Dagelijks 23](#_Toc75544995)

[4.2. Backup 23](#_Toc75544996)

[4.3. In gebruik neming van nieuwe versie 23](#_Toc75544997)

[5.0. Gegevensmodel 24](#_Toc75544998)

[5.1. Bronbeschrijving – Profiling 24](#_Toc75544999)

[5.2. Databeschrijving 25](#_Toc75545000)

[5.3. Conceptueel gegevensmodel 25](#_Toc75545001)

[5.4. OLTP database ontwerp – ERD 27](#_Toc75545002)

[5.5. OLTP database omschrijving 27](#_Toc75545003)

[5.5.1. Candlestick tabel 27](#_Toc75545004)

[5.5.2. Currency tabel 29](#_Toc75545005)

[5.5.3. Exchange tabel 30](#_Toc75545006)

[5.5.4. Wallet tabel 30](#_Toc75545007)

[5.5.4. Trade tabel 31](#_Toc75545008)

[5.6. Data warehouse model 33](#_Toc75545009)

# 1.0. Business Intelligence requirements

Business Intelligence requirements zijn eisen waaraan het systeem moet voldoen. Door vast te leggen wat de gebruiker moet kunnen doen, wat het systeem hoort te doen, tot welke klassen bepaalde eisen behoren en wat er altijd óf nooit moet gebeuren kan je bepalen wat er tijdens het project allemaal moet worden gedaan en wat er kan worden gedaan. De BI requirements worden opgesteld aan de hand van interviews met stakeholders en moeten ook de goedkeuring van stakeholders krijgen. Hierdoor kan je dit document gebruiken als een soort van contract of bewijs dat wat je oplevert voldoet aan de gestelde eisen en wensen waar de betrokkenen mee akkoord zijn gegaan.

De Business Intelligence requirements leggen wij vast door gebruik te maken van het 3-delen model. In dit model stel je ten eerste user stories op. User stories zijn handelingen van de gebruiker in het systeem/programma. Vervolgens op basis van die user stories stel je eerst bijbehorende (niet-)functionele requirements op en vervolgens requirements die los staan van de user stories. Functionele requirements zijn vereiste die het systeem/BI oplossing moet kunnen uitvoeren. Deze functionele requirements worden gecategoriseerd met het ‘MoSCoW’ model. Niet-functionele requirements worden gemaakt aan de hand van ISO25010. ISO25010 is een checklist die je langs kan gaan om te kijken waar het systeem aan moet voldoen, maar die niet door de opdrachtgever direct zullen worden gesteld. Denk bijvoorbeeld aan beveiliging, onder houdbaarheid en prestatie. Het nut van dit model is om iteratief te werk te gaan en bij elke user story gelijk de bijbehorende requirements op te stellen. Na het één keer langsgaan herhaal je het proces, totdat je alle mogelijke user stories en requirements hebt opgesteld.

## 1.1. User stories

User stories maakt een handeling van de gebruiker duidelijk. Deze user stories volgen de vorm van: *actor -> handeling -> doel.* Dit wordt geformuleerd als: “<actor> doet <handeling> met als beoogd <doel>”.

|  |  |  |
| --- | --- | --- |
| Actor | Handeling | Doel |
| Gebruiker | Bekijkt om het uur, van 09:00 tot 21:00, of er een koopsignaal is, doordat een stijgende trend wordt voorspeld doormiddel van het ‘Bullish Dragonfly Doji’ of het ‘Bullish White Marubozu’ patroon | Koopt Dogecoin ten waarde van €100,- in |
| Gebruiker | Bekijkt om het uur, van 09:00 tot 21:00, of er een verkoopsignaal is, doordat een dalende trend wordt voorspeld doormiddel van het ‘Bearish Long Black Candle’ of het ‘Bearish Gravestone Doji’ patroon | Gaat ‘short’ en verkoopt Dogecoin ten waarde van €100,- |
| Gebruiker | Een uur na het inkopen van Dogecoin ter waarde van €100,- verkoopt de gebruiker het aantal gekochte Dogecoin | Door na een uur meteen het aantal gekochte Dogecoin weer te verkopen, toen er vanuit werd gegaan dat er een stijgende trend zou zijn en ‘long’ ing, int de gebruiker zijn winst |
| Gebruiker | Een uur na het verkopen van Dogecoin ter waarde van €100,- koopt de gebruiker het aantal verkochte Dogecoin terug in | De gebruiker ging uit van een dalende trend en ging ‘short’. Na een uur koopt hij het aantal verkochte Dogecoin terug in voor een lagere prijs om zijn winst te innen |
| Gebruiker | Klikt op zijn balans | Ziet de status en een overzicht van zijn balans over de tijd heen |
| Gebruiker | Klikt op een logboek | Ziet een overzicht van de acties (koop, verkoop) die zijn gedaan over de tijd heen |
| Gebruiker | Klikt op een specifieke actie | Ziet de status van de koers van Dogecoin op het moment vóór de uitvoering van de actie. Dit is inclusief een candlestick grafiek en de status van Dogecoin |
| Gebruiker | Gaat naar het overzicht van de Dogecoin koers en selecteert een datum tot en met drie jaar terug | Ziet een overzicht van hoe de koers er op die periode uitzag. Is inclusief een candlestick grafiek, informatie over Dogecoin en de status van zijn balans |
|  |  |  |

## 1.2. Requirements

Alle requirements, functioneel en niet-functioneel worden geclassificeerd aan de hand van het ‘MoSCoW’ model. Oftewel er wordt per requirement gekeken of het moet worden gemaakt, of het zou moeten worden gemaakt, of het kan worden gemaakt of dat het niet mag worden gemaakt.

### 

### 1.2.1. Functionele requirements

Functionele requirements zijn vereiste die het systeem moet kunnen uitvoeren. De requirements volgen de vormen van: *voorwaarde -> functie -> resultaat* of *systeem -> functie -> resultaat.* Dit wordt geformuleerd als: “In het geval van <voorwaarde> dient het systeem <functie> uit te voeren met dit <resultaat>” en “Het <systeem> moet <functie> uit voeren met het <resultaat>”.

|  |  |
| --- | --- |
| Requirement | Categorie |
| In het geval van een verkoopsignaal dient het systeem ‘short’ te gaan en Dogecoin ten waarde van €100,- te verkopen | Must have |
| In het geval van een koopsignaal dient het systeem ‘short’ te gaan en Dogecoin ten waarde van €100,- te kopen | Must have |
| In het geval dat er ‘short’ is gegaan dient het systeem een uur na de verkoop dezelfde hoeveelheid Dogecoin terug in te kopen voor een lagere prijs om zo de winst te innen | Must have |
| In het geval dat er ‘long’ is gegaan dient het systeem een uur na de koop dezelfde hoeveelheid Dogecoin weer te verkopen voor een hogere prijs om zo winst te maken. | Must have |
| In het geval dat er wordt genavigeerd naar de balans dient het systeem een overzicht de geven van de status van de balans over de tijd heen. | Must have |
| In het geval dat er wordt genavigeerd naar het logboek dient het systeem een overzicht te geven van de acties die zijn uitgevoerd over de tijd heen | Must have |
| In het geval dat er wordt geklikt op een specifieke actie dient het systeem de status van Dogecoin op het moment voordat de actie werd ondernomen te weergeven, inclusief een candlestick grafiek en de status van de dogecoin koers | Must have |
| In het geval dat er wordt genavigeerd naar een specifieke datum dient het systeem een overzicht te geven van hoe de koers er in die periode uitzag, inclusief een candlestick grafiek, de status van de dogecoin koers en de status van de balans | Must have |
| Het systeem moet een algoritme gebruiken dat rekening houdt met het sentiment van de handelaren en de aandacht die dogecoin krijgt op sociale media, beurzen, nieuwsplatforms en financiële bronnen | Must have |
| Het systeem moet een webrapportage tonen om informatie te geven over Dogecoin (volume, aantal verkocht, etc.) | Could have |
| Het systeem moet een overzicht geven waarin de status van de balans en ondernemen acties over de tijd heen worden gecombineerd | Could have |
| Het systeem moet een duidelijk overzicht tonen per dag, week, maand, kwartaal, jaar en over het geheel | Could have |
| Het systeem moet naast de vier patronen die koop- en verkoopsignalen moeten herkennen, ook andere patronen gebruiken om signalen te herkennen | Could have |
| Het systeem moet, als er Dogecoin is gekocht of verkocht, na een uur eerst kijken of de trend doorzet voordat een tegentransactie wordt uitgevoerd | Won’t have |
| Het systeem moet om inloggegevens vragen, of registratie voorstellen, voordat er toegang wordt verleend aan de gebruiker | Won’t have |

### 1.2.2. Niet-functionele requirements

Niet-functionele requirements zijn vereiste waaraan het programma of systeem zich aan moet houden. Deze requirements gaan niet over het uitvoeren van bepaalde acties, maar *hoe* die acties worden uitgevoerd. Zo worden vragen over onder andere de beveiliging, prestatie, onderhoudbaarheid, herbruikbaarheid, betrouwbaarheid, etc. beantwoord. Ook deze requirements volgen het ‘MoSCoW’ model.

|  |  |
| --- | --- |
| Requirement | Categorie |
| De applicatie pagina’s moeten binnen 5 seconden volledig geladen zijn | Must have |
| Er wordt in de database logging bijgehouden over de ondernomen acties en veranderingen in de database | Must have |
| Er moet een testrapport worden geschreven | Must have |
| Er moet gebruik worden gemaakt van een ‘staging area’ | Must have |
| Er moet gebruik worden gemaakt van één of meerdere ETL processen | Must have |
| De te gebruiken ETL tool moet gratis (open source) zijn | Must have |
| De te gebruiken ETL tool moet goed kunnen werken met OLAP en OLTP databases | Must have |
| De ETL tool moet automatisch kunnen runnen | Must have |
| Het data warehouse maakt gebruik van MySQL | Must have |
| De te gebruiken ETL tool moet kunnen werken met kleine tot middelgrote datasets | Must have |
| Er moet rekening worden gehouden met ontwerpprincipes en comments | Must have |
| Er moet gebruik worden gemaakt van ‘version control’, als Git | Must have |
| Er moet gebruik worden gemaakt van Kimball’s Data Marts | Must have |
| De taken en scrumonderdelen worden bijgehouden in Trello | Must have |
| Het data warehouse moet een audit trail bijhouden | Must have |
| De ETL software moet zowel een initiële- als een incrementele load ondersteunen | Must have |
| Er moet gebruik worden gemaakt van ‘conformed dimensions’ in het data warehouse | Could have |
| Er wordt gebruikt gemaakt van Inmon’s EDW | Won’t have |
| Vertrouwelijke data als wachtwoorden moeten worden gehasht | Won’t have |
| Er moeten gebruikers zijn die bepaalde rollen krijgen en geautoriseerd worden om bepaalde acties uit te voeren | Won’t have |
| De database moet worden beveiligd op aanvallen, bijvoorbeeld SQL injecties | Won’t have |
| De applicatie moet worden getest op het Linux OS | Won’t have |
| De pagina’s moeten automatisch herkennen wanneer er nieuwe data binnenkomt en herladen | Won’t have |
| Er moeten automische backups worden gemaakt van de data | Won’t have |
| De applicatie moet bruikbaar zijn voor alle soorten apparaten (mobiel, etc.) | Won’t have |
| De applicatie moet offline kunnen werken | Could have |

# 2.0. Business Intelligence architectuur

De business intelligence architectuur geeft inzicht wat er wordt gebruikt ‘onder de motorkap’. Het beantwoord onder andere de vraag: ‘Hoe komt data van een bepaalde bron als informatie op de business intelligence solution terecht?’. Om die vraag te kunnen beantwoorden wordt er een overzicht gegeven van de zogeheten ‘data pipeline’, hoe en waar de data wordt opgeslagen en welke software tools worden gebruikt om de data van het ene naar het andere punt in de pipeline te krijgen.

## 2.1. Input en output

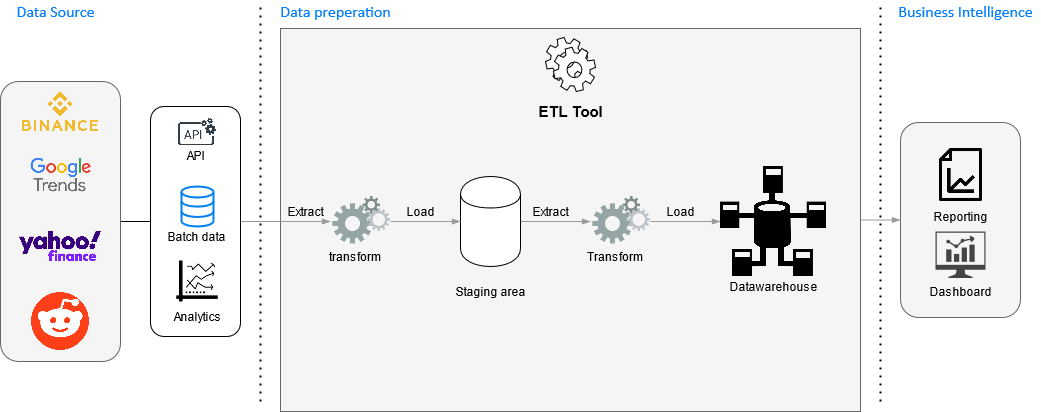
De ‘input en output’ weergeeft hoe de data tot informatie leidt zonder de business intelligence solution en hoe het er uit zou moeten komen te zien met een geïntegreerde data pipeline.

### 2.1.1. Huidige situatie

In de huidige situatie wordt er vanaf 09:00 gekeken of en welk signaal er wordt gegeven doormiddel van verschillende patronen. Op basis van dat signaal wordt er vervolgens voor €100,- aan Dogecoin gekocht (‘long’ gaan) of verkocht (‘short’ gaan). Tot slot wordt er een uur later een tegenovergestelde handeling uitgevoerd: als er is verkocht om 09:00, dan wordt dezelfde hoeveelheid Dogecoin gekocht en als er om 09:00 is gekocht, dan wordt dezelfde hoeveelheid Dogecoin verkocht.

Momenteel wordt er data opgehaald van een API en doormiddel van een ETL proces in een OLTP database gezet. Hierbij wordt niet gekeken naar marktsentiment en wat voor effect dat heeft op de koers van de Dogecoin.

### 2.1.2. Gewenste situatie – data pipeline



**Toelichting**

* Data Source
  + Data van een API, batch data en analytics van bronnen als Binance, Google Trends, Yashoo Finance en Reddit.
    - Deze data bevat de koers van Dogecoin en data voor de sentiment analyse
* Data preperation
  + Hier wordt de data voorbereid om klaar te maken voor gebruik
  + Doormiddel van een ETL Tool wordt de data uit de verschillende bronnen verzameld, ondergaat minimale transformatie (vandaar de kleine ‘t’) en wordt ingeladen in een ‘staging area’: een database waar de data uit de verschillende bronnen tijdelijk op één plek wordt opgeslagen.
  + De data uit de staging area wordt wederom geëxtraheerd, getransformeerd (denk aan het opschonen, filteren, samenvoegen, etc. van de data) en in het datawarehouse ingeladen
  + Tijdens deze fase worden de acties uitgevoerd op basis van herkende signalen en het marktsentiment.
* Business Intelligence
  + Op basis van de data in het datawarehouse kunnen analyses worden uitgevoerd waarmee onder andere rapportages en dashboards kunnen worden gemaakt

## 2.2. Technology Stack

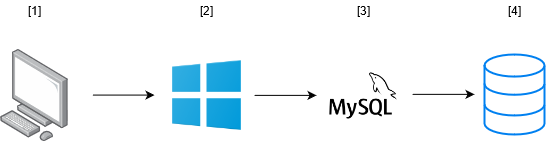
Over het algemeen volgt de technology stack de volgende structuur:

Hardware -> OS -> Data architectuur -> Data

Aan het begin van de ontwikkeling van de BI solution wordt gewerkt met onder andere een relationele database, wat later wordt overgezet naar een data warehouse model. Dit onderscheidt wordt hier beneden duidelijk gemaakt.

### 2.2.1. Ontwikkeling

**OLTP database**



[1] X86 Computer -> [2] Windows 10 -> [3] MySQL database -> [4] handeldata

**OLAP database**

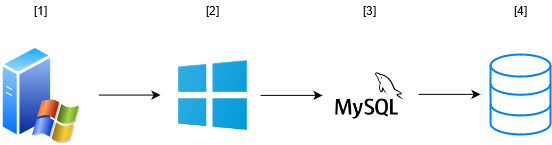
Afbeelding met tekst, klok

Automatisch gegenereerde beschrijving

[1] X86 Computer -> [2] Windows 10 -> [3] data warehouse (data marts) -> [4] handeldata

### 2.2.2. Productie

**OLTP database**



[1] Windows Server -> [2] Windows -> [3] MySQL database -> [4] handeldata

**OLAP database**

Afbeelding met tekst, klok, antenne

Automatisch gegenereerde beschrijving

[1] Windows Server -> [2] Windows -> [3] data warehouse (data marts) -> [4] handeldata

## 2.3. Pakketselectie

Er zijn veel verschillende ETL beschikbaar voor gebruik, wat de mogelijkheden ook erg breed maakt. Om tot uiteindelijk de beste ETL tool te komen voor dit project kijken wij als eerst naar de zes meest gebruikte tools op het moment. Vervolgens kijken we naar de niet-functionele BI requirements om eventuele tools die niet kunnen voldoen aan de eisen uit te sluiten. Tot slot houden we een stel tools over en na raadpleging van het projectteam kiezen we er één of meerdere en schrijven daar een advies over. Uiteindelijk wordt beschreven waar het besluit op is gevallen en waarom.

### 2.3.1. Mogelijkheden

**Pandas**

Pandas is de meest gebruikte toolkit voor gegevensmanipulatie en analyse in het Python universum. Dankzij constante ontwikkeling en een intuïtieve API is het mogelijk om bijna alles in Pandas te doen. Het is ook heel makkelijk te begrijpen vanwege de uitgebreide documentatie. Een nadeel van Pandas ia dat het behoorlijk traag kan zijn als er met Big Data wordt gewerkt. Het is een goede keuze voor het bouwen van een ‘proof of concept’ ETL-pipeline of kleine datasets, maar het is niet heel ‘scalable’, dus als je veel data hebt of een geavanceerdere, grote ETL-pipeline nodig hebt zijn er betere opties.

**Apache Spark**

Apache Spark is een uniforme analyse-engine voor grootschalige gegevensverwerking. In tegenstelling tot Pandas is Apachje Spark ontworpen om te werken met enorme datasets op enorme clusters van computers. Spark is technisch gezien geen Python tool, maar je kan gemakkelijk taken voor Python workflows af laten handelen. Overweeg Spark als je snelheid en omvang nodig hebt bij je gegevensbewerkingen.

**Petl**

Petl is een tool die is ontworpen met gebruiksgemak als belangrijkste focus. Als je werkt met gemengde kwaliteit, onbekende en heterogene gegevens is Petl de beste keuze. Met Petl kan je tabellen in Python bouwen vanuit verschillende databronnen (denk aan CSV, XLXS, HTML, TXT, JSON, etc.) en ze uitvoeren naar het door jou gewenste opslagformaat. Petl is alleen gefocust op ETL, wat het dus efficiënter maakt dan Pandas, omdat het niet elke keer de database in het geheugen laadt als het een regel code uitvoert. Aan de andere kant bevat het geen extra functionaliteiten, zoals ingebouwde gegevensanalyse of visualisatie. Voor puur een ETL-pipeline is Petl heel goed te gebruiken. Kortom: Voor het bouwen van een eenvoudige ETL-pipeline is het goed, maar voor complexere pipelines met meer eisen is het beter om verder te kijken.

**Bonobo**

Als je graag met Python werkt, geen nieuwe API wilt leren en semi-complexe ETL-pipelines wilt bouwen is Bonobo een goede keuze. Bonobo heeft ETL-tools voor het bouwen van datapipelines die meerdere databronnen parallel kunnen verwerken en het heeft een SQLAlchemy-extensie (waarmee pipelines rechtstreeks verbonden kunnen worden met SQL-databases). Een nadeel van Bonobo is dat het nog volop in ontwikkeling is, wat het als lange termijnoplossing niet ideaal maakt, maar voor op korte termijn of voor eenvoudige doeleinden wel.

**Mara**

Mara is een ETL-tool die licht van gewicht is, maar wel alle standaardfuncties aanbiedt. Een aantal aandachtspunten voor Mara zijn:

* PostgreSQL moet worden gebruikt als gegevensverwerkingsengine.
* Declaratieve Python code moet worden gebruikt om data integratie te definiëren.
* De ‘command-line’ is van groot belang bij de interactie met de databases.
* Voor de inspectie, uitvoering en debugging van de pipelines wordt een web-interface gebruikt.

**Pygrametl**

Pygrametl is een Python framework dat veelgebruikte functionaliteiten aanbiedt om ETL processen mee te ontwikkelen. Pygrametl biedt objectgeoriënteerde abstracties aan voor veelgebruikte bewerkingen; zoals interfacing tussen verschillende databronnen, het uitvoeren van parallelle dataverwerking of het maken van snowflake-schema’s. Het is makkelijk te integreren met andere Python code en heel uitgebreid om je eigen datawarehouse mee op te bouwen.

### 2.3.2. Advies

We zullen hier nog eventjes het advies duidelijk eruit toelichten per tool:

**Pandas**

Wanneer je kiest voor Pandas is dit een goede keuze voor het bouwen van een ‘proof of concept’, ETL-pipeline of als je werkt met kleine datasets. Het is niet heel ‘scalable’, dus als je veel data hebt of een geavanceerdere, grote ETL-pipeline nodig hebt zijn er betere opties denk bijvoorbeeld dan aan Apache Spark.

**Apache Spark**

Overweeg je keuze voor Spark als je snelheid, omvang nodig hebt bij je gegevensbewerkingen en moet werken met grote datasets.

**Petl**

Kies je voor Petl dan moet je goed onthouden dat dit voor het bouwen van een eenvoudige ETL-pipeline heel goed is, maar voor complexere pipelines met meer requirements is het beter om verder te kijken.

**Bonobo**

Kies je voor bonobo moet je goed onthouden dat het nog volop in ontwikkeling is maar het een goede tool is als je graag wilt werken met Python en moet werken met semi-complexe ETL-pipelines.

**Mara**

Mara is zeker een interessante keuze om in gedachte te houden omdat het alle standaardfuncties biedt en als ETL-tool ook nog in ontwikkeling is. Maar er zitten ook bepaalde beperkingen aan vast zoals je alleen PostgreSQL kan gebruiken als gegevensverwerkingsengine. Wij zouden dan aanraden om toch een van de andere 5 ETL-tools te kiezen die meer mogelijkheden bieden ook op het gebied interactie, debugging, uitvoering.

**Pygrametl**

Wanneer je voor Pygrametl wilt kiezen is het een goede keuze wanneer je je code met andere Python code wilt integreren en het is heel uitgebreid om je eigen datawarehouse mee op te bouwen.

### 2.3.3. Besluit

Uiteindelijk hebben wij gekozen om pandas te gebruiken omdat pandas voor ons al een bekende tool was en we hier zelf ook al veel ervaring mee hebben. Pandas is daarnaast ook heel makkelijk te begrijpen en er zijn op internet veel hulpbronnen wanneer je er zelf niet uit zou komen. Het is ook makkelijk mee te werken met kleine datasets zoals nu daarom heeft dit ook geholpen om de keuze tot pandas te nemen. Pandas is daarnaast ook gratis wat een groot voordeel is. Het is al heel ver in ontwikkeling dus de stabiliteit van pandas is ook goed. Het heeft duidelijke documentatie en is ook altijd beschikbaar dus om al deze redenen is ons besluit gevallen op pandas.

## 2.4. Systeemarchitectuur

Op basis van de Technology Stack.

### 2.4.1. Hardware

In productie wordt gebruik gemaakt van een Windows Server waarnaar toegang kan worden verkregen.

### 2.4.2. Netwerk

Qua netwerk worden de volgende eisen gesteld:

* Een constante verbinding
* Een stabiele verbinding
* Open staan voor externe verbindingen
* Kunnen verbinden met externe bronnen

### 2.4.3. Software

Het operating system is Windows. Voor verbinding met een andere OS zal een Virtual Machine moeten worden gebruikt. Verder wordt gebruik gemaakt van MySQL voor de database.

## 2.5. Datawarehouse architectuur

Hier wordt gekeken naar de verschillende opties voor het datawarehouse architectuur. Via het ETL proces wordt data opgehaald en getransformeerd, maar het moet ook ergens worden ingeladen; namelijk het datawarehouse. Er zijn verschillende bestaande architecturen voor verschillende use-cases, requirements en einddoelen.

De architecturen zijn erg complex aangezien het naast transactie data zoals in een OLTP database ook nog historische data moet kunnen bevatten. De data architecturen en componenten waar wij naar kijken wij het besluit nemen zijn: Een staging area, EWD (Inmon), data marts (Kimball), EDW + data marts als combinatie, data vault en comformed dimensions.

**Staging area**

Aangezien we van meerdere verschillende bronnen data zullen ontvangen zal het vanwege efficiëntie wenselijk zijn om alles op één plek te hebben, voordat het naar het data warehouse wordt gestuurd. Binnen het ophalen van de data en besluiten nemen over acties die ondernomen kunnen worden is tijd van groot belang en dus zal een staging area nodig zijn voor het project.

**EDW (Inmon)**

Wegens de complexiteit van EDW’s door het moeten overlappen en integreren van verbanden en relaties zal een EDW niet wenselijk zijn voor dit project. De data pipeline van bron tot data warehouse zal van redelijk kleine schaal zijn en dus is een EDW niet de beste optie.

**Data marts (Kimball)**

Aangezien er veel verschillende data binnenkomt voor verschillende doeleinden zal het gebruik maken van data marts een goede toevoeging zijn, omdat er dan scheidingen kunnen worden gemaakt. Het voordeel van data marts is ook de query performance en de simpliciteit, wat het een goede keuze maakt voor dit project.

**Data vault**

Een data vault zit tussen de data marts en de staging area in met als voornaamste doel om eventuele veranderingen in de brondata goed op te kunnen vangen. In theorie is dat een goede keuze voor het project, omdat de API’s waarvan de brondata komt regelmatig wordt veranderd, maar in de praktijk is het een overbodige architectuur voor een project van deze schaal waar eventuele veranderingen niet een enorm probleem opleveren en dus extra complexiteit toevoegen.

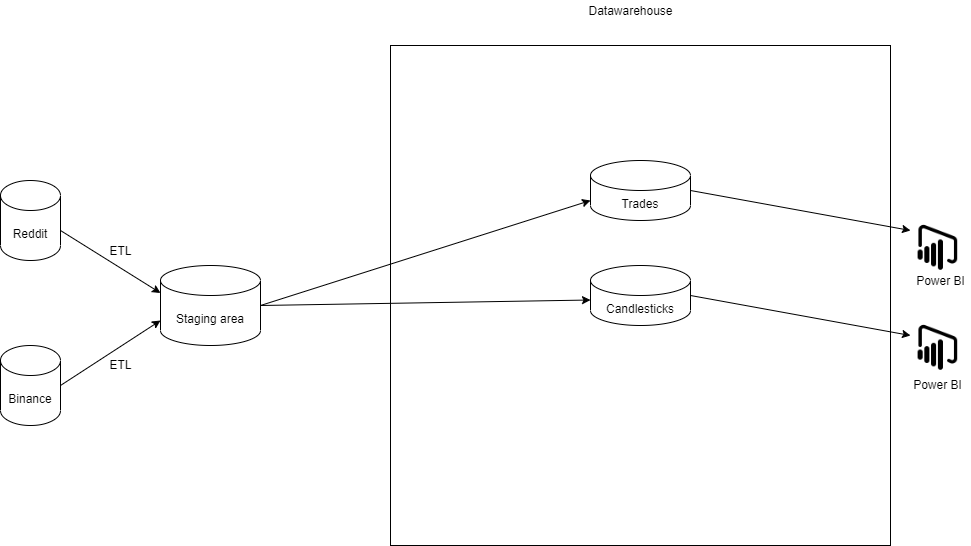
**Conformed dimensions**

Conformed dimensions adresseren de problemen van data marts, namelijk dat er overlap ontstaat tussen de verschillende data marts. Conformed dimensions zijn niet als te complex en makkelijk te integreren met data marts.

**Conclusie**

Op basis van deze analyse lijkt de beste keuze voor dit project om naast de staging area ook data marts van Kimball met een optionele toevoeging van conformed dimensions (eventueel op een later moment) te gebruiken voor de architectuur van het data warehouse.

### 2.5.1 Architectuur



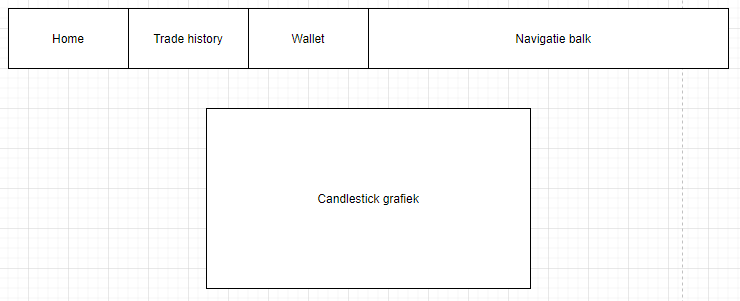
**Toelichting**

Dit is ons datawarehouse architectuur. De data komt vanuit onze bronsystemen(Reddit, Binance) via het ETL proces in de staging area. Hier wordt de data verzameld en is doormiddel van ETL al getransformeerd. Na de staging area wordt het met ETL weer door geladen naar de datamarts. Als het eenmaal in de datamarts zit kunnen er analyses, rapportages op gedaan worden.

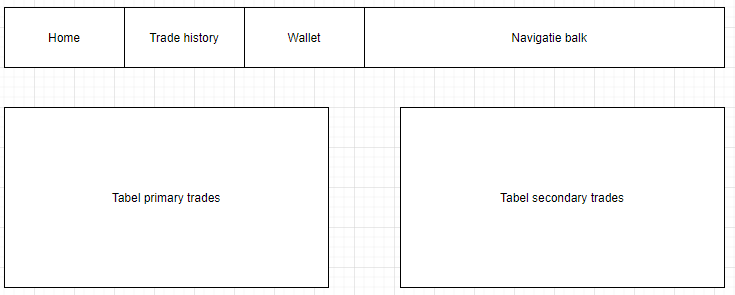
# 3.0. Processen en ontwerp

## 3.1. Wireframes

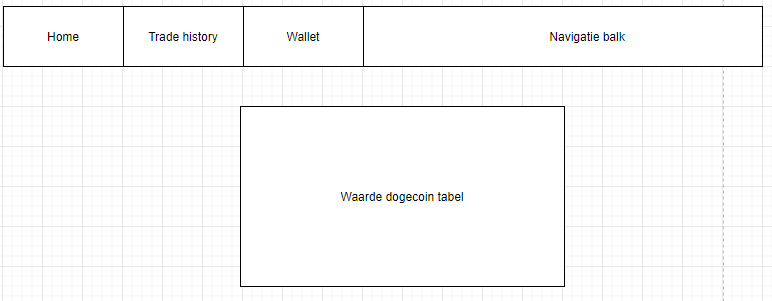
Startscherm:



Trade history:

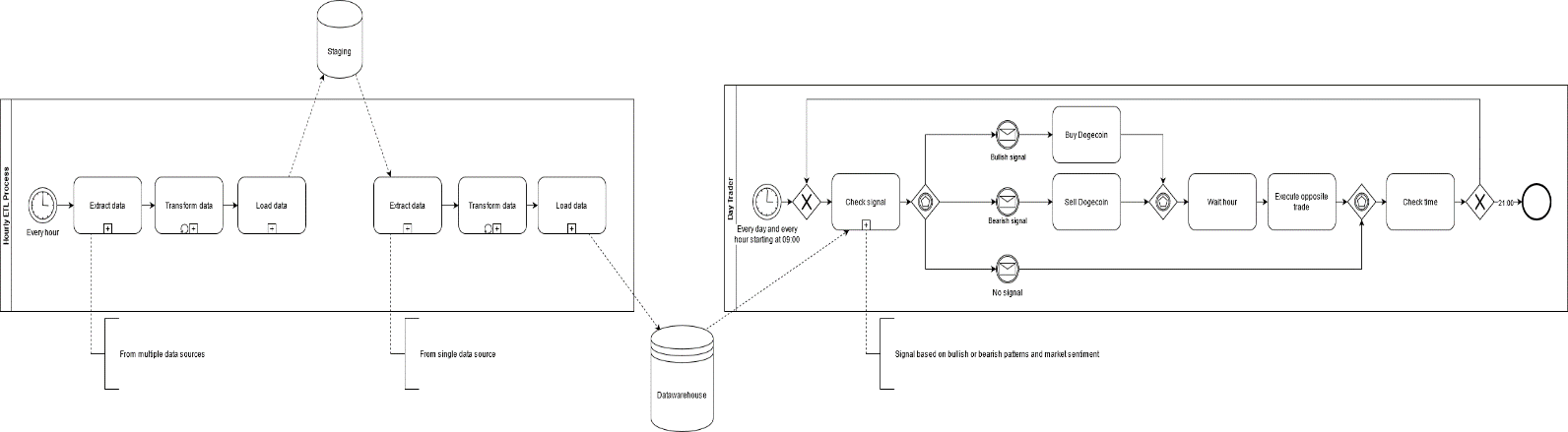


Wallet:



## 3.2. Processen

### 3.2.1. Dagelijks



(Zie bijlage 1 voor een vergrootte versie)

**Toelichting**

Het gehele dagelijkse proces is afhankelijk van twee processen: Het ETL proces elk uur en de daghandelaar. De twee processen zijn verbonden doormiddel van het ‘datawarehouse’, wat wordt gevuld door het ETL proces en gebruikt wordt door de daghandelaar.

* ETL proces
  + Wordt elk uur automatisch opnieuw gestart
  + Is verbonden met twee externe databases: Het ‘staging database’ en het ‘datawarehouse’

1. Als eerst worden van verschillende databronnen (zie de data pipeline) data verzameld
2. Vervolgens wordt er door alle data heen gelopen en minimale transformatie uitgevoerd
3. Nu de data enigszins is opgeschoond wordt het in een staging database ingeladen. Dit zorgt ervoor dat de data van de verschillende bronnen nu op één plek staat.
4. Dan wordt er technisch gezien een nieuw ETL proces gestart en wordt de data uit de staging database geëxtraheerd, getransformeerd naar gewenst formaat door onder andere door alle data heen te lopen om tot slot in een datawarehouse te worden ingeladen.

* Daghandelaar (Day trader)
  + Het proces van de daghandelaar wordt elke dag opnieuw gestart, maar herhaald elk uur een aantal stappen
  + Het gebruikt de data uit de datawarehouse die voorheen is gevuld door het ETL proces

1. Wanneer het 09:00 ’s ochtends is start het proces
2. Er wordt gekeken welk signaal wordt gegeven door de BI solution
   1. Dit signaal wordt gebaseerd op bepaalde ‘candlestick’ patronen en marktsentiment.
      1. De data hiervoor wordt uit het datawarehouse gehaald
   2. Het signaal kan ‘bullish’, ‘bearish’ of niks zijn.
      1. Dat houdt in dat de koers waarschijnlijk een stijgende, dalende of geen trend zal ervaren.
3. Op basis van het afgegeven signaal zal er voor €100,- aan Dogecoin worden gekocht of verkocht, of in het geval van geen signaal wordt er niks gedaan.
4. Na het kopen of verkopen van Dogecoin wordt na een uur wachten een tegenovergestelde transactie uitgevoerd. Dat wilt zeggen dat als er voor €100,- aan Dogecoin is ingekocht wordt er dezelfde hoeveelheid Dogecoin weer verkocht, en andersom als er voor €100,- aan Dogecoin is verkocht.
5. Vervolgens wordt er naar de tijd gekeken: als het 21:00 is dan eindigt het proces van de dag, anders begint het proces opnieuw bij het kijken naar welk signaal er wordt gegeven.

### 3.2.2. Controles

Om te controleren of de juiste acties worden ondernomen in de BI solution kunnen de volgende stappen worden gevolgd:

1. Kijk welke actie is ondernomen (koop, verkoop, geen) door naar het logboek te gaan in de applicatie
2. Kijk of er een dalende of stijgende trend plaatsvond *na* de genomen actie
3. Controleer wat de koer *voor* en *na* de genomen actie was
4. Bepaal of de genomen actie correct was
5. Controleer welk patroon (of geen) werd gegeven *voor* de genomen actie
   1. Past het gegeven patroon bij de genomen actie?
      1. Een bullish patroon past bij het inkopen van Dogecoin
      2. Een bearish patroon past bij het verkopen van Dogecoin
   2. Ga door naar **stap 6** als het niet klopt
6. Controleer welk marktsentiment er was (zeer slecht, slecht, neutraal, goed, zeer goed)
   1. Past het marktsentiment bij de genomen actie?
      1. Een heel slecht sentiment past bij een dalende trend
      2. Een heel goed sentiment past bij een stijgende trend
   2. Ga door naar **stap 7** als het niet klopt
7. Ga naar het datawarehouse en bekijk de data van de candlestick en de currency op het moment van de genomen actie
8. Controleer of de data correct en compleet is
   1. Zo ja: dan ligt het probleem niet aan de data of het ETL proces, maar aan het algoritme of aan een fout in de code
      1. Ga in dit geval van de productie versie af en debug het probleem
   2. Zo nee: dan ligt het probleem aan de data
      1. Ga in dit geval door naar **stap 9**
9. Het probleem ligt bij het proces voordat het wordt gebruikt in de applicatie. Controleer ten eerste of er meer data foutief is opgeslagen in het datawarehouse
10. Doorloop en debug de data pipeline. Kijk bij welke van de onderstaande stappen het fout gaat:
    1. Extraheer testdata in het eerste ETL proces
    2. Transformeer de data en laadt het in de staging database
    3. Controleer of de data correct in de staging database staat
    4. Extraheer de data uit de staging database
    5. Transformeer de data en laadt het in de datawarehouse
11. Debug het gevonden probleem in de development omgeving, test het en voer dan door naar de productie omgeving

# 4.0. Beheer

## 4.1. Dagelijks

Het is niet benodigd om dagelijks beheer of onderhoud uit te voeren.

## 4.2. Backup

Naar aanleiding van het interview worden er geen backups gemaakt, en wordt er geen proces gedefinieerd. Als dit in de toekomst veranderd zien we het dan wel weer.

## 4.3. In gebruik neming van nieuwe versie

Voor het in gebruik nemen van een nieuwe versie zal de oude versie compleet worden verwijderd, minus de databases, om vervolgens de nieuwe versie op de Windows Server te zetten.

# 5.0. Gegevensmodel

## 5.1. Bronbeschrijving – Profiling

Om de data op te halen die we nodig hebben van de doge coin gebruiken we een API van api.binance.com. Je kan hiermee een API request doen voor een gekozen crypto coin met wat variabelen die je mee kan geven, de belangrijkste zijn welke coin, welk interval en de starttijd in ms (UNIX-timestamp). Je krijgt hier een JSON-object van terug met allemaal arrays. In het JSON-bestand zitten allemaal arrays vanaf het opgegeven moment in intervallen die opgegeven zijn. Om deze API te gebruiken heb je een API key nodig die je kunt aanvragen.

Een voorbeeld van hoe je de API moet gebruiken:

* <https://api.binance.com/api/v3/klines?symbol=DOGEEUR&interval=1h&startTime=1615553000000>
  + Symbol (welke coin en welke currency)
    - DOGE & EURO = DOGEEUR
    - DOGE & USD = DOGEUSDT
  + Interval (Tussen welke periodes wil je data zien)
    - Interval=1h laat de OHLCV candlestick van het afgelopen uur zien
  + startTime (vanaf wanneer wil je data zien, hier zet je een UNIX-timestamp neer in ms)

De JSON die je dan terugkrijgt is dan:

[

[

1499040000000, // Open time

"0.01634790", // Open

"0.80000000", // High

"0.01575800", // Low

"0.01577100", // Close

"148976.11427815", // Volume

1499644799999, // Close time

"2434.19055334", // Quote asset volume

308, // Number of trades

"1756.87402397", // Taker buy base asset volume

"28.46694368", // Taker buy quote asset volume

"17928899.62484339" // Ignore.

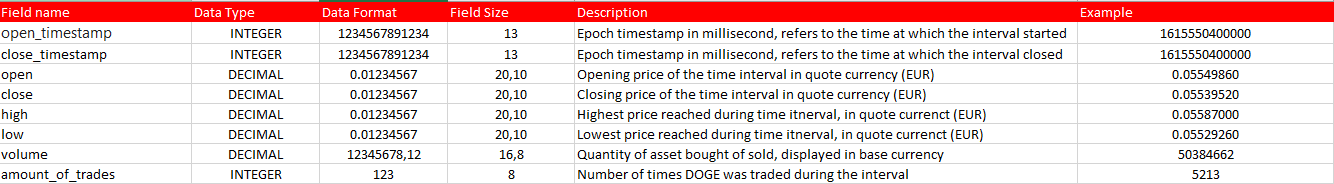
]

]

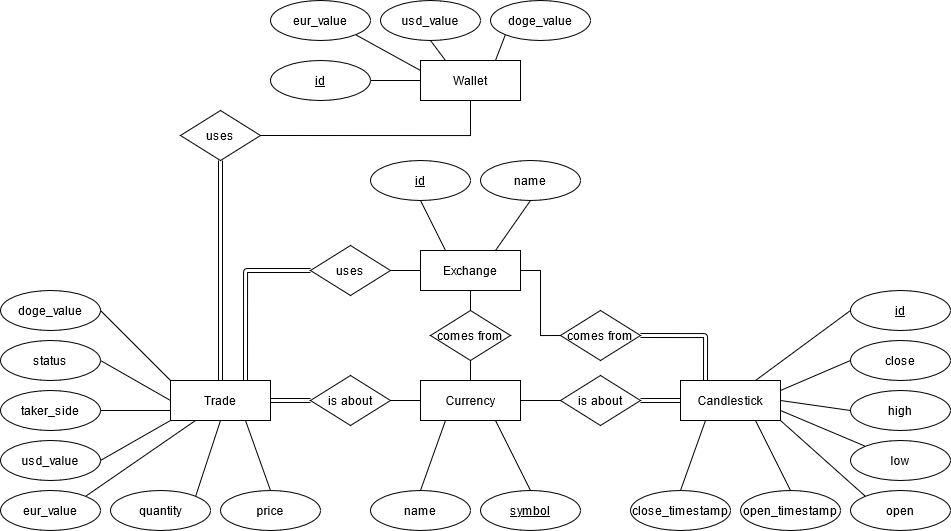
* De base asset is de hoeveelheid coins voor het gekozen symbol.
* De quote asset is de prijs voor het gekozen symbol.

## 5.2. Databeschrijving

Hier de beschrijving van de brondata. Dit gaat over de data die we gebruiken en verkrijgen van de in 5.1 beschreven bron.



## 5.3. Conceptueel gegevensmodel



**Toelichting**

*Trade:*

* Gaat over een bepaalde handeling (koop, verkoop)
* Bevat informatie over de transactie
  + Wat er is gedaan qua koop of verkoop (taker\_side)
  + Hoeveel in kwantiteit is verhandelt (quantity)
  + De waarde van de handel in verschillende valuta (eur\_value, usd\_value, doge\_value)
  + De status van de transactie
    - Is het gelukt, eventuele errors
* Heeft een *total* participatie met *Wallet, Exchange* en *Currency*
  + Elke instantie van *Trade* moet te maken hebben met een instantie van de andere tabellen
  + Een Trade maakt gebruik van een wallet en een bepaalde exchange waarop het wordt verhandelt
  + De Trade gaat altijd over een bepaalde valuta (Bijvoorbeeld Dogecoin)

*Wallet*

* Gaat over de digitale portemonnee van de gebruiker
* Bevat informatie over hoeveel de gebruiker heeft van verschillende valuta
  + Euro, USD, Dogecoin
* Heeft een relatie met de *Trade* tabel

*Candlestick*

* Gaat over de data die de visuele weergave van een candlestick mogelijk maakt
* Bevat relevante data over de koers op een bepaald moment
  + De openings- en sluittijd van de meetperiode (open\_timestamp, close\_timestamp)
  + Op welke prijs de koers tijdens het begin en het einde van de meetperiode (open, close)
  + Wat de hoogst en laagst gemeten prijzen waren in de meetperiode (high, low)
* Heeft een *total* participatie met *Currency* en *Exchange*
  + Elke instantie van *Candlestick* moet te maken hebben met een instantie van de andere twee tabellen
  + Elke candlestick gaat over een bepaalde valuta en de data komt van een bepaalde exchange

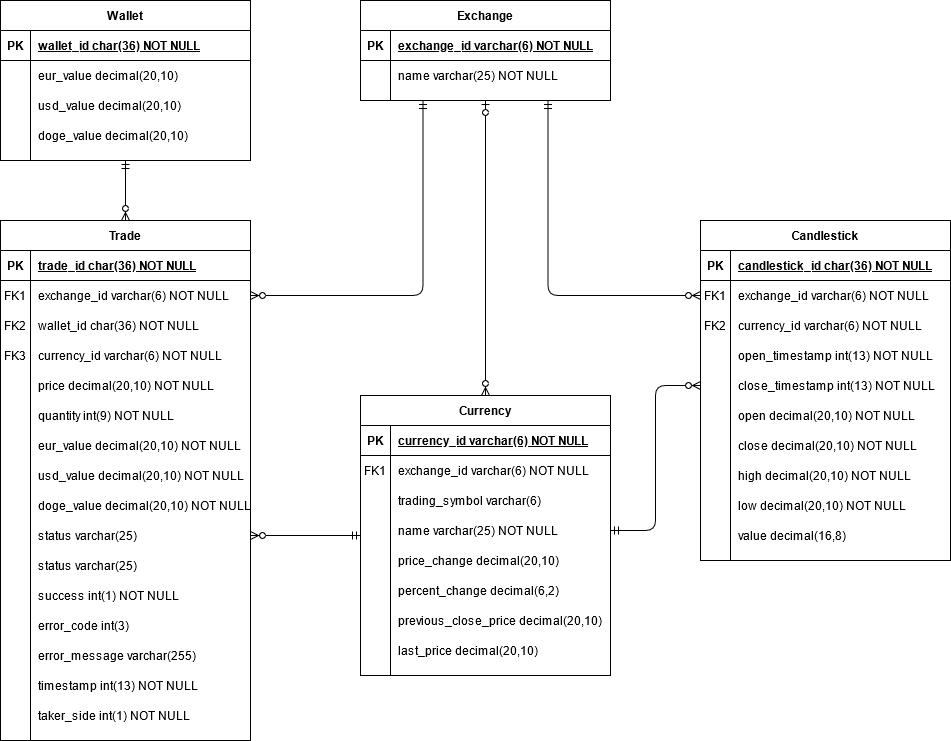
*Currency*

* Gaat over een bepaalde valuta
* Bevat relevante informatie die inzicht geven over de actuele stand van zaken van de currency
  + Het symbool en de naam die worden gebruikt om mee te handelen
* Heeft relaties met *Exchange, Candlestick* en *Trade*

*Exchange*

* Gaat over een bepaalde exchange waarop wordt gehandeld
* Bevat relevantie informatie om te exchange te kunnen herkennen
  + Het symbool en de naam waarmee wordt gehandeld
* Heeft relaties met *Trade, Currency* en *Candlestick*

## 5.4. OLTP database ontwerp – ERD



**Toelichting staat onder 5.5. beschreven**

## 5.5. OLTP database omschrijving

### 5.5.1. Candlestick tabel

De *Candlestick* table bevat alle candlesticks per uur, per dag. Elke rij in de tabel is één candlestick.

#### 5.5.1.1. Relaties

* De *Candlestick* tabel heeft een één op meer (1:N) relatie met de *Exchange* tabel. Elke candlestick komt **altijd** van één exchange, terwijl een exchange meerdere candlesticks **kan** hebben.
* De *Candlestick* tabel heeft een één op meer (1:N) relatie met de *Currency* tabel. Elke candlestick gaat **altijd** over een bepaalde currency, terwijl een currency meerdere candlesticks **kan** hebben.

#### 5.5.1.2. Sleutels

* Elke *Candlestick* rij **moet** een foreing key bevatten die wijst naar een bepaalde *Exchange*.
* Elke *Candlestick* rij **moet** een foreing key bevatten die wijst naar een bepaalde *Currency*.

#### 5.5.1.3. Columns

* *candlestick\_id:* Een unieke surrogate key die wordt gebruikt om elke candlestick te kunnen identificeren
* *exchange\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Exchange* tabel
* *currency\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Currency* tabel
* *open\_timestamp:* Het tijdsmoment dat het begin van een meetperiode aangeeft. Dit is altijd rond het begin van een nieuw uur, zoals: 2-2-2000 09:00:00 of 2-2-2019 14:00:36.
  + Genoteerd als Epoch timestamp in millisecondes.
  + Data type is een Integer
  + Lengte is 13
  + Voorbeeld: 1615550400000
* *close\_timestamp:* Het tijdsmoment dat het einde van een meetperiode aangeeft. Dit is altijd rond het einde van een uur, zoals: 2-2-2000 09:59:59 of 2-2-2019 14:59:36.
  + Genoteerd als Epoch timestamp in millisecondes.
  + Data type is een *Integer*
  + Lengte is 13
  + Voorbeeld: 1615550400000
* *open:* De waarde van de Dogecoin koers (prijs) op het moment dat de meetperiode (*open\_timestamp*) begon.
  + Genoteerd in ‘quote’ currency (EUR)
  + Data type is een *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434
* *close:* De waarde van de Dogecoin koers (prijs) op het moment dat de meetperiode (*close\_timestamp*) eindigde.
  + Genoteerd in ‘quote’ currency (EUR)
  + Data type is een *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434
* *high:* De hoogste waarde dat de Dogecoin koers (prijs) bereikte gedurende de meetperiode.
  + Genoteerd in ‘quote’ currency (EUR)
  + Data type is een *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434
* *low:* De laagste waarde dat de Dogecoin koers (prijs) bereikte gedurende de meetperiode.
  + Genoteerd in ‘quote’ currency (EUR)
  + Data type is een *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434
* *value:* De kwantiteit gekochte of verkochte Dogecoin.
  + Genoteerd in ‘base’ currency
  + Data type is *Decimal*
  + Lengte is 16,8
  + Voorbeeld: 50384662
* *amount\_of\_trades:* Aantal keren gedurende de meetperiode dat er werd gehandeld in Dogecoin.
  + Genoteerd in aantallen
  + Data type is *Integer*
  + Lengte is 8
  + Voorbeeld: 523

### 5.5.2. Currency tabel

De *Currency* table bevat actuele informatie over een bepaalde currency.

#### 5.5.2.1. Relaties

* De *Currency* tabel heeft een één op meer (1:N) relatie met de *Candlestick* tabel. Elke candlestick gaat **altijd** over een bepaalde currency, terwijl een currency meerdere candlesticks **kan** hebben.
* De *Currency* tabel heeft een één op meer (1:N) relatie met de *Exchange* tabel. Elke currency informatie komt **meestal** van een bepaalde exchange.
* De *Currency* tabel heeft een één op meer (1:N) relatie met de *Trade* tabel. Elke trade wordt **altijd** gedaan in een bepaalde currency, terwijl een currency meerdere trades kan hebben.

#### 5.5.2.2. Sleutels

* Elke *Currency* rij **moet** een foreign key bevatten die wijst naar een bepaalde *Exchange*.

#### 5.5.2.3. Columns

* *currency\_id:* Een unieke letter combinatie die wordt gebruikt om currencies te kunnen identificeren. Wordt ook vaak een ‘symbol’ genoemd.
  + Data type is *Varchar*
  + Lengte is 6
  + Voorbeeld: BTC
* *exchange\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Exchange* tabel
* *trading\_symbol:* Een letter combinatie die wordt gebruikt voor het handelen in de currency. Kan hetzelfde of anders zijn dan de *currency\_id*.
  + Data type is *Varchar*
  + Lengte is 6
  + Voorbeeld: USDT
* *name:* De naam van de currency.
  + Data type is *Varchar*
  + Lengte is 25
  + Voorbeeld: Dogecoin
* *price\_change:* Hoeveel (positief of negatief) de prijs in het afgelopen interval (uur) is veranderd.
  + Genoteerd in ‘quote’ currency (EUR, USD)
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: -1.095544
* *percent\_change:* Hoeveel (positief of negatief) de prijs in het afgelopen interval (uur) is veranderd.
  + Genoteerd in percentages (decimalen)
  + Data type is *Decimal*
  + Lengte is 6,2
  + Voorbeeld: -3,28
* *previous\_close\_price:* De close prijs van het vorige interval (uur).
  + Genoteerd in ‘base’ currency (DOGE)
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434
* *last\_price:* De meest actuele prijs van de currency.
  + Genoteerd in ‘base’ currency (DOGE)
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,43434

### 5.5.3. Exchange tabel

De *Exchange* tabel bevat informatie over een bepaalde exchange die wordt gebruikt voor het ophalen van data en het handelen.

#### 5.5.3.1. Relaties

* De *Exchange* tabel heeft een één op meer (1:N) relatie met de *Candlestick* tabel. Elke candlestick data **moet** van een bepaalde exchange komen, terwijl van een exchange meerdere candlesticks **kunnen** komen.
* De *Exchange* tabel heeft een één op meer (1:N) relatie met de *Currency* tabel. Elke currency **moet** zijn data van een bepaalde exchange krijgen, terwijl van een exchange meerdere currencies kunnen komen.
* De *Exchange* tabel heeft een één op meer (1:N) relatie met de *Trade* tabel. Elke trade **moet** op een bepaalde exchange worden gedaan, terwijl op een exchange meerdere trades **kunnen** worden gedaan.

#### 5.5.3.2. Sleutels

De *Exchange* tabel bevat geen foreign keys.

#### 5.5.3.3. Columns

* *exchange\_id:* Een unieke letter combinatie die wordt gebruikt om exchanges te kunnen identificeren. Wordt ook vaak een ‘symbol’ genoemd.
  + Datatype is *Varchar*
  + Lengte is 6
  + Voorbeeld: BIN
* *name:* De naam van de exchange.
  + Datatype is *Varchar*
  + Lengte is 25
  + Voorbeeld: Binance

### 5.5.4. Wallet tabel

De *Wallet* tabel bevat informatie over de ‘wallet’ van de gebruiker. Vooral hoeveel de gebruiker van een bepaalde currency op het moment heeft.

#### 5.5.4.1. Relaties

* De *Wallet* tabel heeft een één op meer (1:N) relatie met de *Trade* tabel. Elke trade **moet** gedaan zijn met een bepaalde wallet, terwijl er meerdere trades **kunnen** zijn gedaan met een wallet.

#### 5.5.4.2. Sleutels

De *Wallet* tabel bevat geen foreign keys.

#### 5.5.4.3. Columns

* *wallet\_id:* Een unieke combinatie van letters en cijfers die wordt gebruikt om een wallet mee te kunnen identificeren.
  + Genoteerd als surrogate key
  + Data type is *Char*
  + Lengte is 36
  + Voorbeeld: 79c9e1ac-96f1-11e5-85a6-000c29f1f6c4
* *eur\_value:* De waarde van de wallet in Euro’s
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 530,70
* *usd\_value:* De waarde van de wallet in Dollars
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 530,70
* *doge\_value:* De waarde van de wallet in Dogecoins
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,76560

### 5.5.4. Trade tabel

De *Trade* tabel bevat informatie over een handeling (trade) die is gedaan. Dit kan zijn dat er iets is gekocht of verkocht.

#### 5.5.4.1. Relaties

* De *Trade* tabel heeft een één op meer (1:N) relatie met de *Wallet* tabel. Elke trade **moet** gedaan zijn met een bepaalde wallet, terwijl er meerdere trades **kunnen** zijn gedaan met een wallet.
* De *Trade* tabel heeft een één op meer (1:N) relatie met de *Exchange* tabel. Elke trade **moet** gedaan zijn met een bepaalde exchange, terwijl er meerdere trades **kunnen** zijn gedaan met een exchange.
* De *Trade* tabel heeft een één op meer (1:N) relatie met de *Currency* tabel. Elke trade **moet** gedaan zijn met een bepaalde currency, terwijl er meerdere trades **kunnen** zijn gedaan met een currency.

#### 5.5.4.2. Sleutels

* Elke *Trade* rij **moet** een foreign key bevatten die wijst naar een bepaale *Exchange*.
* Elke *Trade* rij **moet** een foreign key bevatten die wijst naar een bepaale *Currency*.
* Elke *Trade* rij **moet** een foreign key bevatten die wijst naar een bepaale *Wallet*.

#### 5.5.4.3. Columns

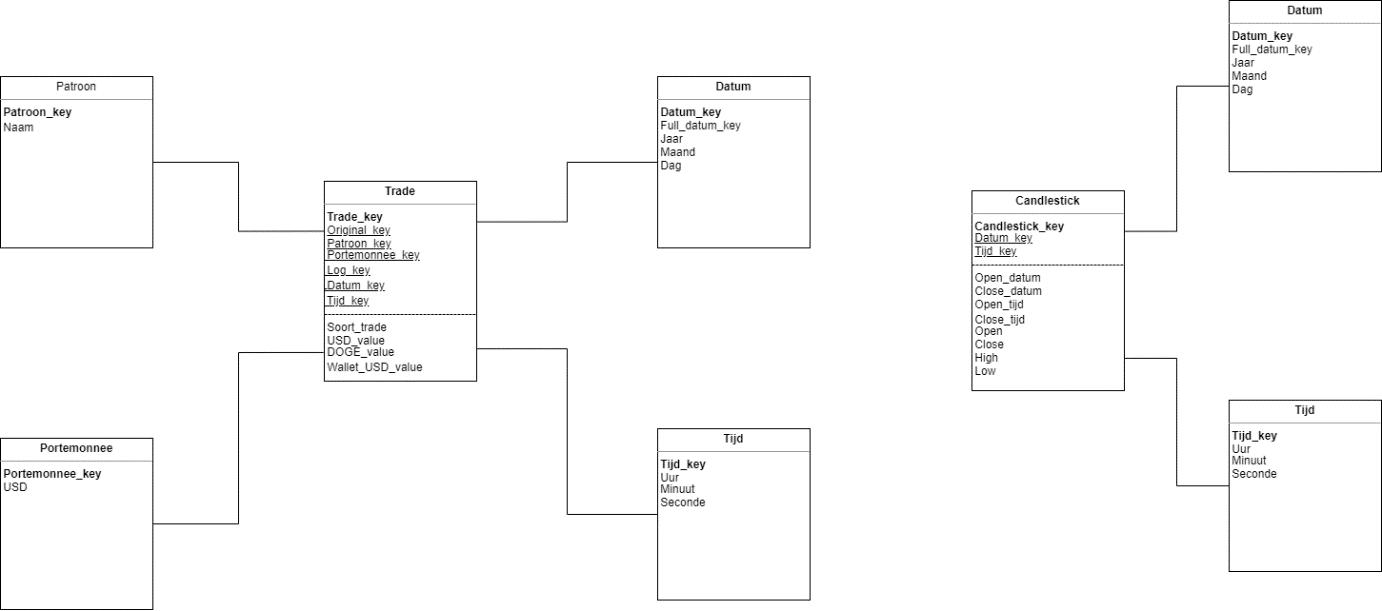
* *trade\_id:* Een unieke combinatie van letters en cijfers die wordt gebruikt om een trade mee te kunnen identificeren.
  + Genoteerd als surrogate key
  + Data type is *Char*
  + Lengte is 36
  + Voorbeeld: 79c9e1ac-96f1-11e5-85a6-000c29f1f6c4
* *exchange\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Exchange* tabel
* *currency\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Currency* tabel. Als er bijvoorbeeld voor €100,- aan Dogecoin is gekocht, dan verwijst deze foreign key naar de Dogecoin currency.
* *wallet\_id:* Een foreign key die wijst naar een uniek identificeerbare rij in de *Wallet* tabel
* *price:* De prijs van de currency op het moment van de trade.
  + Data type is een *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 5,06868
* *quantity:* De hoeveelheid die is gekocht of verkocht. Als je voor €100,- aan Dogecoin koopt, dan is dit de hoeveelheid die je krijgt.
  + Data type is *Integer*
  + Lengte is 9
  + Voorbeeld: 45
* *eur\_value:* De hoeveelheid dat de trade kostte of opleverde in Euro’s
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 1050,00
* *usd\_value:* De hoeveelheid dat de trade kostte of opleverde in Dollars
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 1050,00
* *doge\_value:* De hoeveelheid dat de trade kostte of opleverde in Dogecoin. Dit is het aantal dat is gekocht/verkocht maal de prijs van de currency op het moment van de trade.
  + Data type is *Decimal*
  + Lengte is 20,10
  + Voorbeeld: 6354,43534
* *status:* De huidige status van de trade. Dit kan bijvoorbeeld zijn dat het pending, failed, processed of completed is.
  + Genoteerd als één van een aantal statusberichten
  + Data type is *Varchar*
  + Lengte is 25
  + Voorbeeld: Pending
* *success:* Of de trade succesvol was of niet.
  + Genoteerd als getal (0 of 1)
  + Data type is *Integer*
  + Lengte is 1
  + Voorbeeld: 1
* *error\_code:* Een bepaalde code die zegt of er iets is misgegaan.
  + Genoteerd als http status code getallen
  + Data type is *Integer*
  + Lengte is 3
  + Voorbeeld: 200
* *error\_message:* Een bepaald bericht die zegt wat er is misgegaan.
  + Data type is *Varchar*
  + Lengte is 255
  + Voorbeeld: Not enough currency in wallet to make that trade
* *timestamp:* Het moment van de trade
  + Genoteerd als Epoch timestamp in millisecondes
  + Data type is *Integer*
  + Lengte is 13
  + Voorbeeld: 1234567890000
* *taker\_side:* Welke kant is gekozen (buyer of seller)
  + Genoteerd als getal (0 of 1)
  + Data type is *Integer*
  + Lengte is 1
  + Voorbeeld: 1

**Algemene kolommen**

Naast de bovenstaande kolommen bevat elke tabel ook de volgende twee kolommen:

* *last\_added:* Wanneer een rij is toegevoegd aan de tabel
  + Genoteerd als Epoch timestamp in millisecondes
  + Data type is *Integer*
  + Lengte is 13
  + Voorbeeld: 1234567890000
* *last\_updated:* Wanneer een rij is gewijzigd in de tabel
  + Genoteerd als Epoch timestamp in millisecondes
  + Data type is *Integer*
  + Lengte is 13
  + Voorbeeld: 1234567890000

## 5.6. Data warehouse model



**Toelichting**

In het bovenstaande figuur zie je de datamarts waar wij analyses op baseren. In het midden staat de feit ook wel measure genoemd met daarom heen de dimensies die de feit ondersteunen. Doormiddel van de foreignkey relatie kunnen we ze aanroepen.