Technische verslag

Het CMS voor iedereen

Dante Klijn

Samenvatting

Hier komt de nieuwe samenvatting

Contactgegevens

Student

Naam	Dante Klijn
Studentnummer	4565908
Academisch jaar	2023/2024
E-mail	dante.klijn@student.nhlstenden.com
Telefoonnummer	+31 (0)6 24 76 59 74

Onderwijsinstelling

Naam	NHL Stenden University of Applied Sciences
Course	HBO-ICT
Locatie	Rengerslaan 8-10, 8917 DD, Leeuwarden
Telefoonnummer	+31 (0)88 991 7000

Docentbegeleider

Naam	Stefan Rolink
Email	stefan.rolink@nhlstenden.com
Telefoonnummer	+31 (0)6 42 28 30 77

Afstudeercommissie

Email	afstuderenschoolofict@nhlstenden.com

Examencommissie

Email	examencommissiehboict@nhlstenden.com

Organisatie

Naam	Snakeware New Media B.V.
Locatie	Veemarktplein 1, 8601 DA, Sneek
Telefoonnummer	+31 (0)515 431 895

Bedrijfsbegeleider

Naam	Thom Koenders
Email	thom@snakeware.com
Telefoonnummer	+31 (0)6 13 09 18 51
Rol	Senior software developer

Versiebeheer

Versie	Datum	Veranderingen
0.1	TBD	Eerste hoofdstukken

Woordenlijst

Contentmanagementsysteem Een contentmanagementsysteem is een softwaretoepassing, meestal een webapplicatie, die het mogelijk maakt dat mensen eenvoudig, zonder veel technische kennis, documenten en gegevens op internet kunnen publiceren (contentmanagement). Als afkorting wordt ook wel CMS gebruikt.

Graphical user interface Een graphical user interface (GUI), is een manier van interacteren met een computer waarbij grafische beelden, widgets en tekst gebruikt worden.

Search engine optimization Search Engine Optimisation (SEO), zijn alle processen en verbeteringen die als doel hebben een website hoger in Google te laten verschijnen.

Inhoudsopgave

Hoofdstuk 1

Inleiding

Dit is het technische verslag voor het "Het CMS voor iedereen" project. het verslag is een onderdeel van de afstudeerperiode binnen NHL Stenden Hogeschool. Dit document dient als verantwoording voor de gebruikte processen en het eind product dat geraliseerd is in de afstudeerperiode. In de volgende sectie wordt de organisatie beschreven verder wordt de aanleiding en de context van de afstudeeropdracht beschreven.

Het technische verslag is een onderdeel van de afstudeerperiode binnen NHL Stenden Hogeschool. Het technische verslag dient als documentatie en verantwoording voor het product dat gemaakt is tijdens de afstudeerperiode. In de volgende sectie wordt de organisatie beschreven verder wordt de aanleiding en de context van de afstudeeropdracht beschreven.

1.1 Organisatieomschrijving

Snakeware New Media B.V. (Snakeware) is een E-business bureau gevestigd in Nederland. Haar aangeboden diensten omvatten het adviseren, bouwen en onderhouden van digitale producties, met een focus op websites, webshops en mobiele apps (Snakeware, 2022b). Op het moment van schrijven telt Snakeware meer dan 60 werknemers, elk met verschillende specialiteiten. Ze leveren services aan welbekende organisaties zoals DPG Media, DekaMarkt en Poiesz supermarkten (Snakeware, 2022a).

1.2 Context

Snakeware heeft een platform genaamd "Snakeware Cloud" dit platform is een contentmanagementsysteem (CMS) waarmee ze digitale content kunnen leveren voor haar (grotere) klanten. Snakeware Cloud is een applicatie waarmee Snakeware en haar klanten webapplicaties kan inrichten en voorzien van content.

De klant van Snakeware kan zijn of haar website zelf inrichten door middel van het specificeren van de content op de verschillende pagina's. Dit wordt gedaan door middel van artikelen die door het CMS gebruikt kunnen worden. De content van het artikel kan verschillen tussen simpele tekst, vragenlijst, webshop producten, etc. Hiernaast zijn er ook search engine optimization (SEO) opties binnen Snakeware Cloud om de site goed te kunnen vinden op het internet. Hierbij zijn er opties zoals de mogelijkheid om de title tags en zoekwoorden toe te kunnen voegen in de head (Mozilla, 2023c)

Hierom heeft Snakeware Cloud veel features en configuratie stappen wat het complex en duur

maakt om een relatief kleine webapplicatie te maken voor kleinere klanten. Dit zorgt ervoor dat Snakeware zich niet kan vestigen in een markt met veel kleinere klanten, en hierdoor omzet misloopt.

1.3 Aanleiding

Het huidige platform is 21 jaar oud en er is veel functionaliteit in de loop der jaren aan toegevoegd. Omdat Snakeware Cloud een oud platform is zijn er veel technieken en best practices gebruikt die nu niet meer als optimaal worden beschouwd. Deze technieken waren erg geïntegreerd in Snakeware Cloud en er is het verleden gekozen om niet de code te herschrijven om het aan de huidige standaarden te voldoen van andere projecten. Een voorbeeld hiervan is tabel naam prefix afkortingen bij elke kolom zetten, of gigantische C# (Microsoft, 2022) files van 10 000 regels met verschillende functies. Deze functies houden zich niet aan de *Single Responsibility Principle* van de SOLID ontwerpmethode (Watts, 2020) wat het moeilijk maakt om het huidige CMS te onderhouden.

Ook zijn er technieken toegepast die nu niet meer relevant zijn. Een voorbeeld hiervan is dat het CMS gebruikmaakt van JavaScript (Mozilla, 2023b) en toen ze er mee begonnen bestonden JavaScript classes (Mozilla, 2023a) nog niet, dus hebben ze die zelf geïmplementeerd. Deze oudere technieken en standaarden zorgen ervoor dat het meer tijd kost om het CMS te onderhouden vanwege de extra code. Dit zorgt ervoor dat het meer tijd en geld kost om het Snakeware Cloud uit te breiden.

Een van de voornaamste uitdaging met Snakeware Cloud betreft de verouderde datastructuur van de applicatie. Deze veroudering is het gevolg van een initïele ontwikkeling waarbij onvoldoende rekening werd gehouden met toekomstige functionaliteitsuitbreidingen in het systeem. Als gevolg daarvan is de onderliggende datastructuur niet aangepast, maar zijn er elementen aan toegevoegd. Dit heeft geresulteerd in database query's van duizenden regels en complexe relaties tussen tabellen in de database. Dit huidige scenario bemoeilijkt aanzienlijk het toevoegen van nieuwe functionaliteiten, wat resulteert in aanzienlijke tijd en kosten investeringen.

Hierom wil Snakeware een nieuw systeem met een nieuwe datastructuur. Door het gebruiken van een nieuwe softwarearchitectuur zouden er velen problemen opgelost kunnen worden die nu voor komen. Omdat er een nieuwe datastructuur moet komen en de logica van het oude systeem nauw verbonden is met de datastructuur is het niet mogelijk om de oude code opnieuw te gebruiken.

1.4 Opdrachtomschrijving

De opdracht is om een proof of concept CMS-API te ontwikkelen die gebruikt maakt van een datamodel en systeemarchitectuur dat flexibeler, onderhoudbaarder is en gebruik maakt van moderne best practices. Tijdens de afstudeeropdracht wordt er primair op het datamodel en de systeemarchitectuur gefocust. Omdat er nog geen concreet datamodel en systeemarchitectuur is zal dit onderzocht en ontworpen moeten worden.

De opdracht omvat het achterhalen van de requirements, ontwerpen en ontwikkelen van het proof of concept met als focus een nieuw datamodel, met de essentiële functionaliteiten.

Het huidige Snakeware Cloud platform bestaat uit 2 verschillende graphical user interfaces (GUI):

- Snakeware Cloud GUI
- Klant webapplicatie

Met de Snakeware Cloud GUI kan de klant de content van de website aanpassen. Door middel van de webapplicatie kan de eindgebruiker de content bekijken en er mee interacteren. Er is voor gekozen om niet de Snakeware Cloud GUI te realiseren om de afstudeeropdracht in scope te houden. Er is wel voor gekozen om de klant webapplicatie in zijn minimale vorm uit te werken. Om de userflows van de applicatie toch te kunnen testen wordt er gebruik gemaakt van postman workflows (Postman.com, 2023)

Het doel van het proof of concept is dat er aangetoond kan worden dat door het gebruiken van een nieuw datamodel en systeemarchitectuur ook services verleend kunnen worden aan kleinere klanten. Dit zou eventueel ook een startpunt zijn om op verder te bouwen.

1.5 Leeswijzer

als laatste

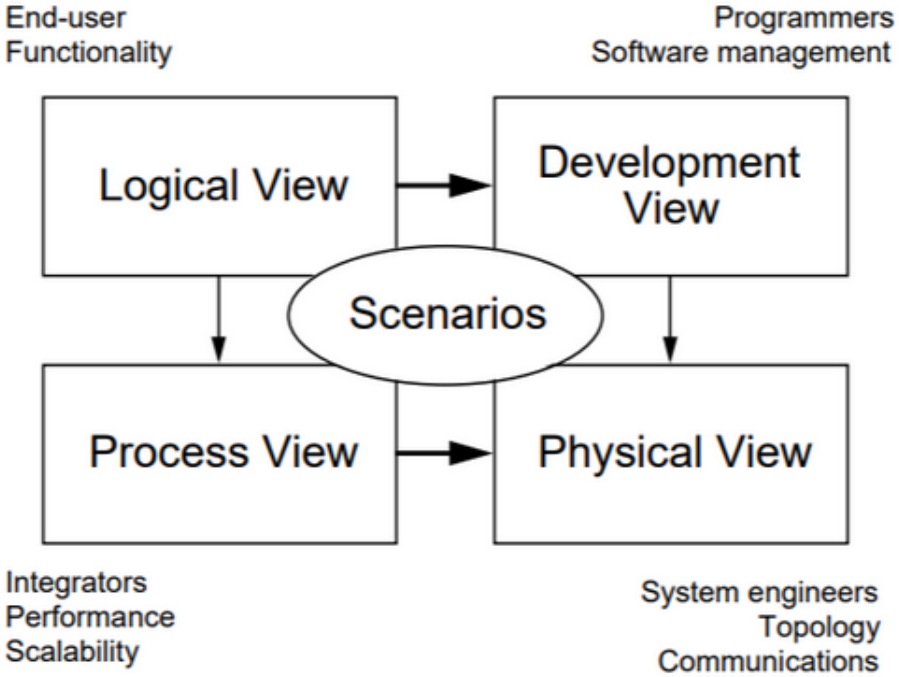
Hoofdstuk 2

Ontwerp

In dit hoofdstuk wordt het ontwerp van de softwareproducten beschreven. Tijdens de eerste fase van de afstudeerperiode is er een onderzoek gedaan naar de requirements van het systeem (Klijn, 2023). Het resultaat van dit onderzoek is een lijst van geprioriteerde requirements en randvoorwaarden waar het systeem aan moet voldoen. Deze resultaten zijn gebruikt om het softwareproduct te ontwerpen.

Voor het ontwerpen van het systeem is er gebruikgemaakt van een ontwerp framework genaamd het 4 + 1 view model (Kruchten, 1995). Het 4 + 1 view model maakt gebruik van 5 verschillende perspectieven om de software inbeeldt te krijgen. Deze perspectieven zijn scenarios, logical, process, development en physical view (visualisatie is te zien in figuur 2.1). In de volgende secties worden de perspectieven uitgelegd en ingevuld.

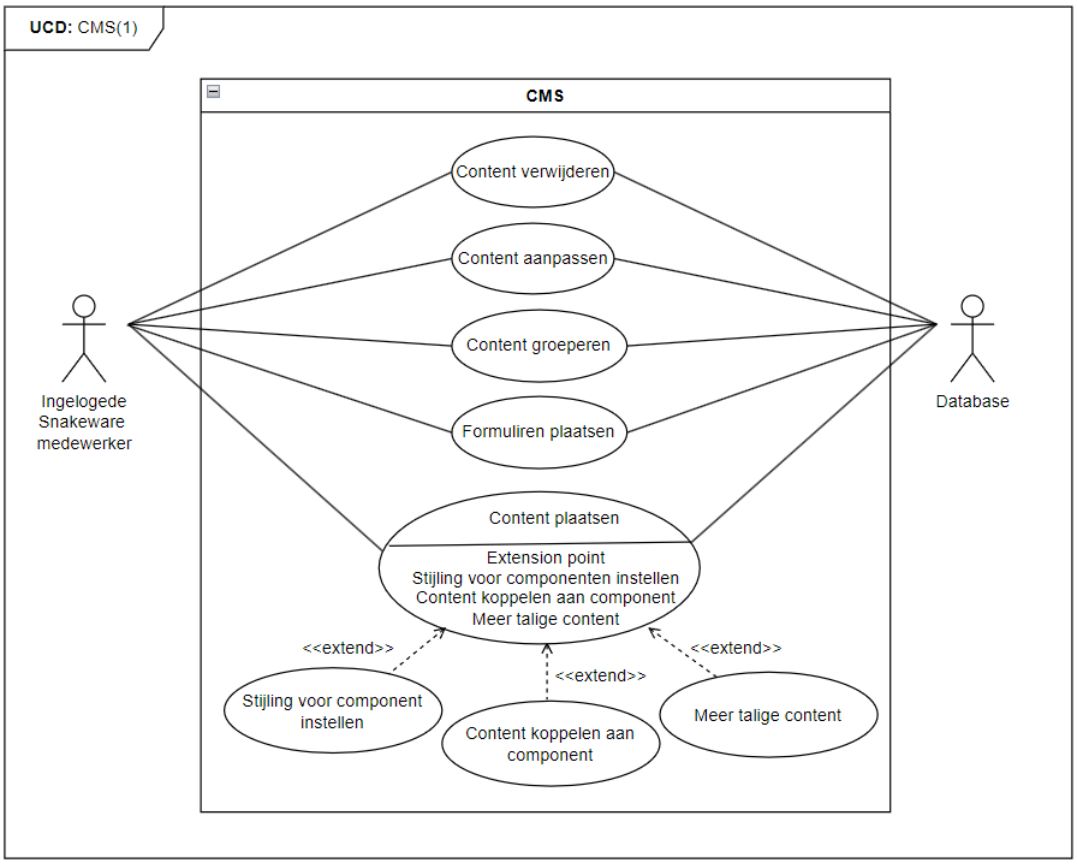
Figuur 2.1: 4 + 1 Model view model (Kruchten, 1995)



2.1 Scenario's

De scenario view is een representatie van de belangrijkste use cases van het systeem (Kruchten, 1995). De use cases zijn opgesteld door middel van de geprioriteerde lijst van requirements van het onderzoek (Klijn, 2023). Om de verschillende use cases en interactie met andere actoren in het systeem in beeld te krijgen wordt er gebruik gemaakt van een use case diagram (lucidchart, 2023). In figuur 2.2 2.3 2.4 zijn de verschillende use case diagramen te zien. Er is voor gekozen om de use cases voor het CMS-platform op te delen om het meer overzichtelijk te maken.

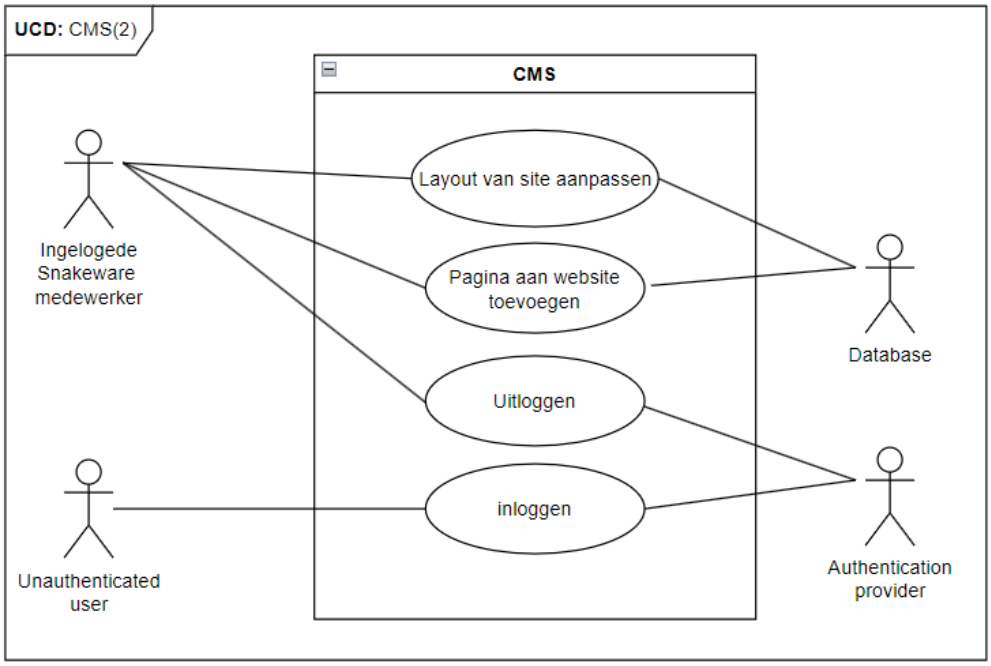
Figuur 2.2: Use case diagram CMS(1)



In figuur 2.2 zijn de verschillende use cases die te maken hebben met content binnen het CMS. Content wordt gedefineerd als informatie dat op de site te vinden hierbij kun je denken aan een card, text of een afbeelding.

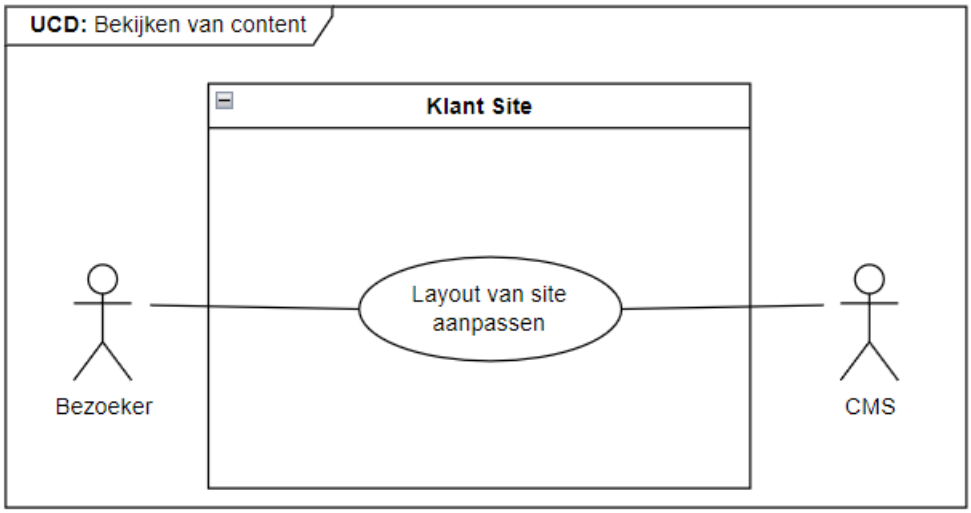
Voor de overige functionaliteiten is er in figuur 2.3 ook een use case diagram te vinden. Dit use case diagram toont het authenticatie en algemene site functionaliteit.

Figuur 2.3: Use case diagram CMS(2)



Het laatste diagram is gemaakt voor de klanten site. Dit is de locatie waar de (website) bezoeker de content kan lezen.

Figuur 2.4: Use case diagram Klant site



2.2 Logical View

In de volgende sectie wordt de logical view van het 4 + 1 model toegelicht. Het doel van de logical view is de functionaliteiten van het systeem in beeld te brengen (Kruchten, 1995). Dit wordt gedaan door op een abstract niveau naar de structuur en datamodel van het systeem te kijken zonder implementatie details. De volgende subsecties gaan dieper in op het datamodel en de softwarearchitectuur van het systeem.

2.2.1 Datamodel

Een van de belangrijkste doelen van de afstudeeropdracht is om een nieuw datamodel te maken voor een CMS-systeem. Dit datamodel moet veel verschillende soorten datastructuren kunnen ondersteunen en deze kunnen presenteren op een klant zijn website. Het huidige CMS van Snakeware doet dit doormiddel van een complexe structuur. Deze structuur zorgt ervoor dat het moeilijk is om nieuwe functionaliteit toe te voegen en dat het systeem lastig is te onderhouden.

Daarom heeft het nieuwe datamodel twee belangrijke uitgangspunten om de pijnpunten van het oude CMS te voorkomen. Het datamodel moet generiek blijven, zodat er veel verschillende datastructuren in opgeslagen kunnen worden. Door het datamodel generiek en flexibel te houden hoeft het systeem niet uitgebreid te worden om een nieuwe datastructuren te ondersteunen. Verder moet de structuur van het datamodel simpel blijven zodat het makkelijk te onderhouden is. Het ontwerp bestaat uit verschillende onderdelen met verschillende taken. Er wordt op elk onderdeel ingezoomd om een beter beeld te schetsen aan het ontwerp.

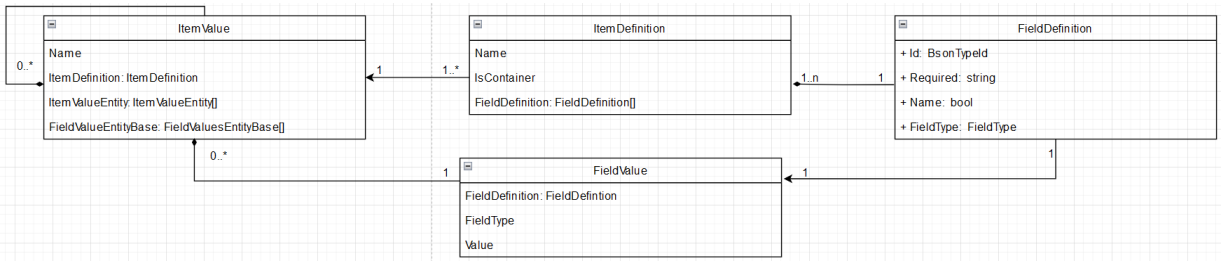
De content

Om de content op te slaan wordt er gebruikgemaakt van een geneste structuur. Om deze geneste structuur op te bouwen is er gekozen om de data op te delen in 2 groepen.

1 **Simpele data**: hier worden de basis types van het systeem in opgeslagen. Dit zijn de primaire types van de content voorbeelden van deze types zijn tekst, nummer en boolean. In het ontwerp is simpele data genoteerd als **fieldvalues**.
2 **Complexe data** is een samenstelling van 0 of meerdere **fieldvalues**. Verder kan complexe data meerdere complexe data bevatten. Hierdoor kunnen er complexe structuren gemaakt worden. Dit is gerepresenteerd als **ItemValues** in het ontwerp.

Om structuur aan de data te geven worden ze voorzien door een definitie. Er zijn definities voor **itemvalues** en **fieldvalues**. Dit is gedaan zodat als er een nieuwe instantie van een stuk complexe data gemaakt moet worden dat dit op basis van de definitie gedaan kan worden. Om dit weer te geven is er een klassendiagram gemaakt die te zien is in figuur 2.5.

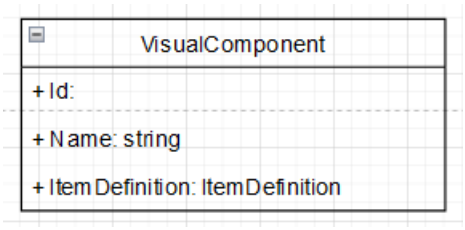
Figuur 2.5: Klassendiagram ItemValue



De componenten

Om de content een vorm te geven wordt er gebruik gemaakt van componenten. Voorbeelden van componenten zijn card, artikel en afbeelding. Door de componenten en de content te scheiden van elkaar geeft dat de mogelijkheid om verschillende componenten te gebruiken op hetzelfde stuk content. Om ervoor te zorgen dat een component een stuk content kan interperteren wordt er gebruik gemaakt van definities. Als een stuk content alle *required FiedlDefinitions* heeft kan de component de content renderen. De componenten worden gerepresenteerd door **visualcomponent** in het ontwerp. Een klasse diagram van de visual component is te zien in figuur 2.6.

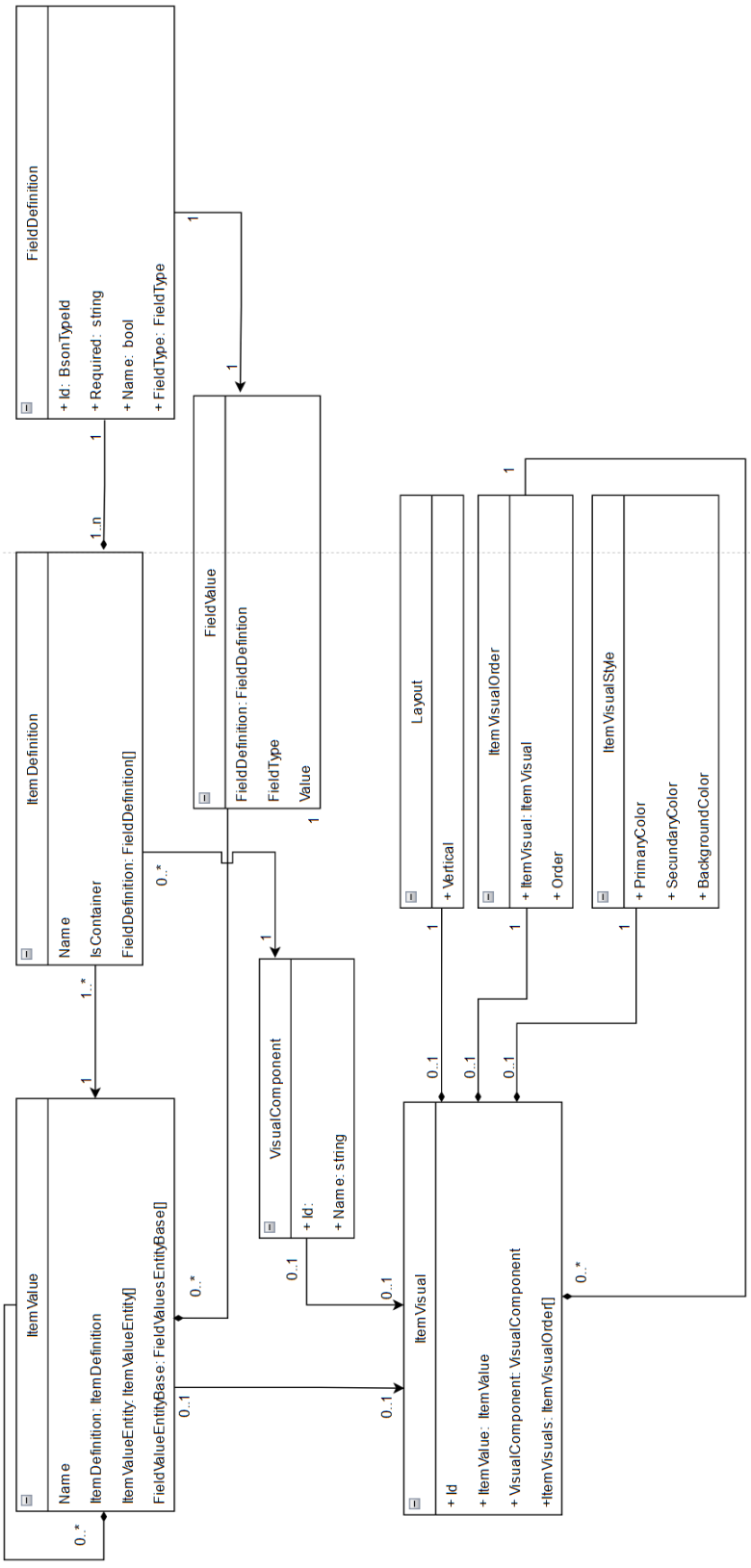
Figuur 2.6: Klassendiagram VisualComponent



De visualalisatie

Om content op een pagina te renderen moet de content gekoppeld worden samen met de componenten. Dit wordt gerepresenteerd door het **itemvisual** in het ontwerp. Verder kan een itemvisual ook meerdere itemvisuals bevatten waardoor er een geneste structuur ontstaat. Hierdoor is het mogelijk om complexe websitestructuren te maken. Verder wordt ook op het itemvisual stijling en layout meegegeven zodat dit niet component afhankelijk is. Het klassen diagram voor het gehele ontwerp is te zien in figuur 2.7

Figuur 2.7: Klassendiagram ItemValue



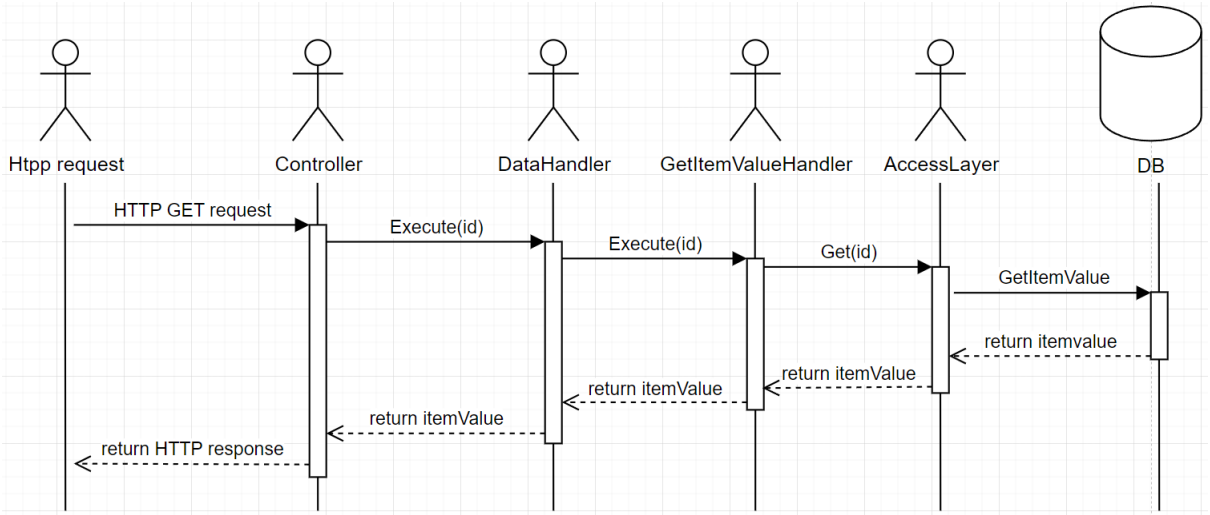
2.2.2 Software architectuur

Om de onderhoudbaarheid te verbeteren van het huidige CMS is er ook nagedacht over het opzetten van de code. Hiervoor is er gekeken hoe de architectuur zich houdt aan de verschillende SOLID principes (Watts, 2020). SOLID is een acroniem dat 5 verschillende principes inhoudt. Deze principes zorgen ervoor dat de code beter te onderhoudbaar is en makkelijker te begrijpen is. SOLID bestaat uit de volgende principes:

1. **Single-responsibility principle** Dit betekent dat een class / module, maar 1 verantwoordelijk mag hebben en bij definitie ook maar 1 taak. Als een class / module te veel verantwoordelijkheden heeft dan wordt het moeilijker om de code te begrijpen en aan te passen.
2. **Open-closed principle** Dit houdt in als class of functie of een andere software entiteit uitgebreid moet worden dat het wordt gedaan door middel van een extentie(open) in plaats van modificatie(closed). Hierdoor hou je oude code intact en heb je geen risico dat je bestaande code stuk gaat vanwege de nieuwe functionaliteit die geschreven is.
3. **Liskov substitution principle** Een class die afgeleid is van een base class, zou moeten vervangen kunnen worden voor een andere instantie van die base class. Dit moet gedaan kunnen worden zonder de validiteit (corectness) van het programma te beïnvloeden. Door het gebruik van het Liskov substitution principle verhoog je de consistentiteit en de verwachte uitkomst van je programma.
4. **Interface segregation** Een interface moet alleen de methods geven die nodig is voor de client. Geen client moet geforceerd zijn om methodes te implementeren waar die geen gebruik van maakt. Dit kan verminderd worden om meerdere kleinere interfaces te maken inplaatst van een grote interface. Deze interfaces zijn verantwoordelijk voor meer specifieke usecases in plaats van generaliseerde usecases
5. **Dependency inversion** Een class of module zou niet moeten afhangen van implementaties maar van abstracties. Hier door verminder je de koppeling van de modules/classes, en verhoog je de code onderhoudbaarheid. Om aan dit principe te voldoen wordt er vaak gebruik gemaakt van dependency injection.

Om de verschillende aspecten van SOLID te implementeren is er gekozen om gebruik te maken van handlers. Hierbij heeft elke handler één taak. Daarnaast wordt er ook gebruikgemaakt van een repository pattern om met de database te communiceren. Er is gekozen om gebruik te maken van een repository pattern zodat er geen afhankelijkheid is van de database. Een sequencediagram van deze architectuur is te zien in figuur 2.8

Figuur 2.8: Sequencediagram Handler structuur



De verschillende handlres zorgen ervoor dat het single-responsibility na gevolgt wordt. Omdat elke handler maar een taak heeft. Verder worden er kleine interfaces gebruikt om aan het Interface segregation principle te volgen. En als laatste worden de verschillende dependencies geinjecteerd (dependency injection pattern) om harde koppeling te voorkomen.

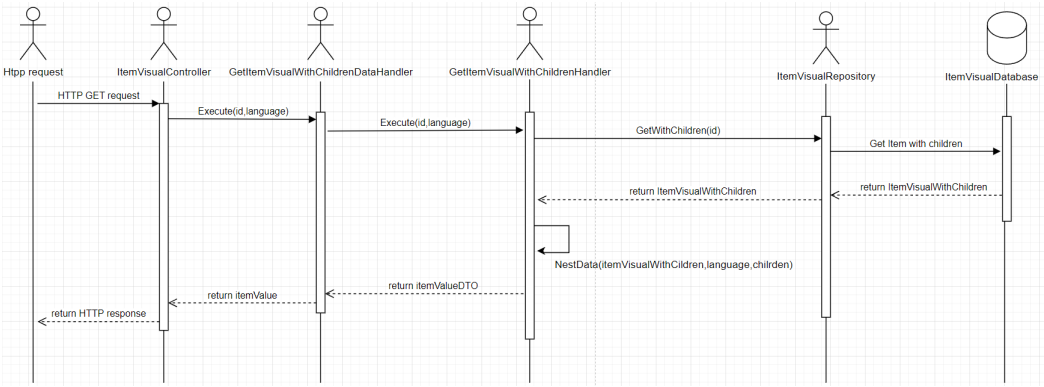
2.3 Process View

De process view van het 4 + 1 model dient om het run-time gedrag van het systeem in beeld te brengen (Kruchten, 1995). In deze sectie wordt er gekeken hoe het ophalen van de data vanuit de backend werkt. Verder wordt er ook gekeken hoe de data gerenderd wordt in de frontend.

2.3.1 Backend

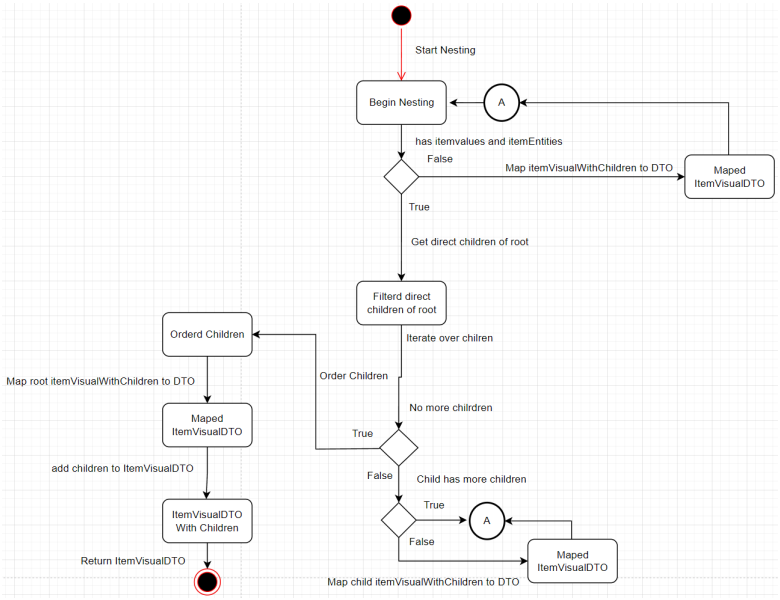
Om het proces van het ophalen van de data in beeld te krijgen is er gebruik gemaakt van een sequencediagram. In figuur 2.9 is te zien dat er 4 lagen zijn. Dit zijn de Controller, Datahandler, handler en access layer. De logica om te bepalen of het een succesvolle request was zit in de datahandler. Logica om het verwachte object terug te krijgen zit in de handler. En als laatste is er de access layer die de data uit de database haalt.

Figuur 2.9: Sequencediagram ItemValue



Een belangrijke functionaliteit in het nesten van de data zodat de frontend het makkelijk kan renderen. Dit wordt gedaan in de NestData functie om meer inzicht te geven van het proces van de functie is er een flowchart gemaakt (zie figuur 2.10).

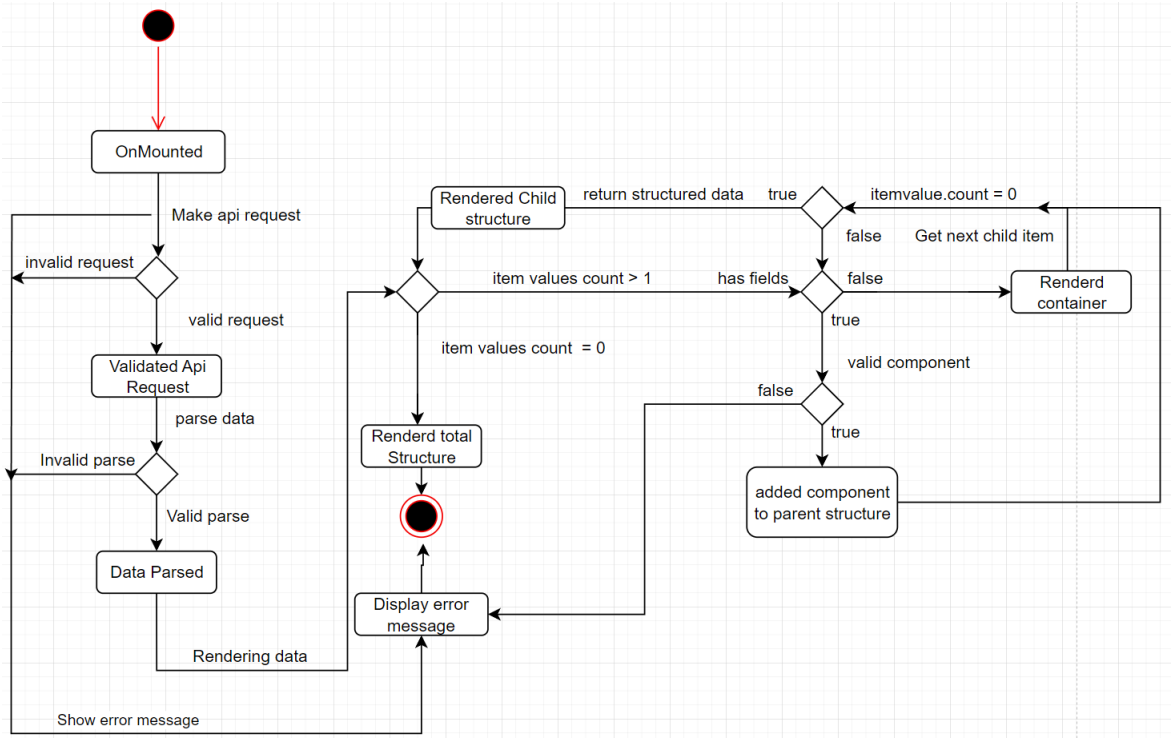
Figuur 2.10: flowchart diagram NestData



2.3.2 Frontend

Om de data te renderen wordt er gebruikgemaakt van 2 verschillende entiteiten types. Type 1 is een voor gedefinieerd component, hier bij kun je denken aan een artikel, card, afbeelding etc. Het andere type is een generiek component dat het type 1 componenten encapsuleerd. Deze componenten worden containers genoemd omdat ze verschillende items encapsuleren. Verder kunnen containers ook meerdere containers encapsuleren.

Figuur 2.11: flowchart diagram frontend



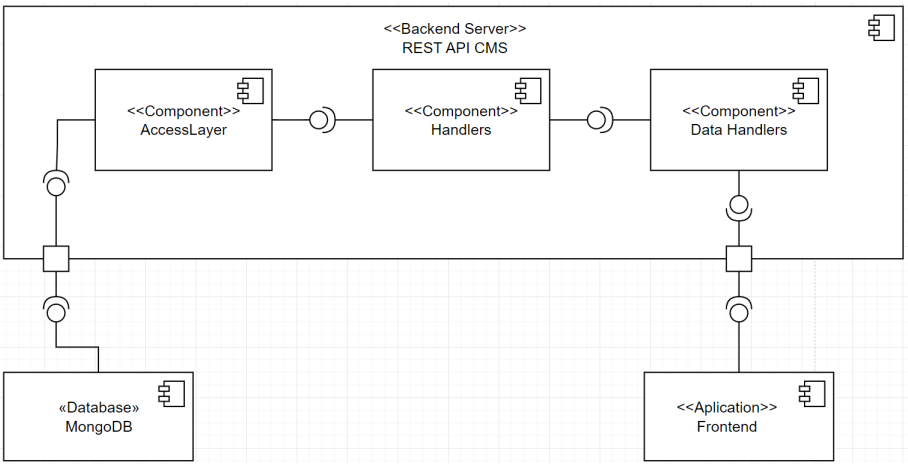
In figuur 2.11 wordt er getoond hoe de frontend met deze container structuur om gaat. Na het initiële ophalen en valideren van de data begint de main loop van de frontend. Eerst wordt er gekeken of het huidige object 1 of meer items heeft. Daarna wordt elk item gerenderd en als het een container is, wordt deze recursief ook gerenderd. Dit gaat door tot dat alle items gerenderd zijn.

2.4 Development view

De development view is gefocust op het inbeeldt brengen van de organisatie van software modules en de software development omgeving (Kruchten, 1995). Om dit in beeld te brengen is er gebruikgemaakt van een component diagram (zie figuur 2.12).

Ik ben het meest onzeker over dit diagram. Want ik heb een gevoel dat ik het veel te erg gegeneraliseerd heb.

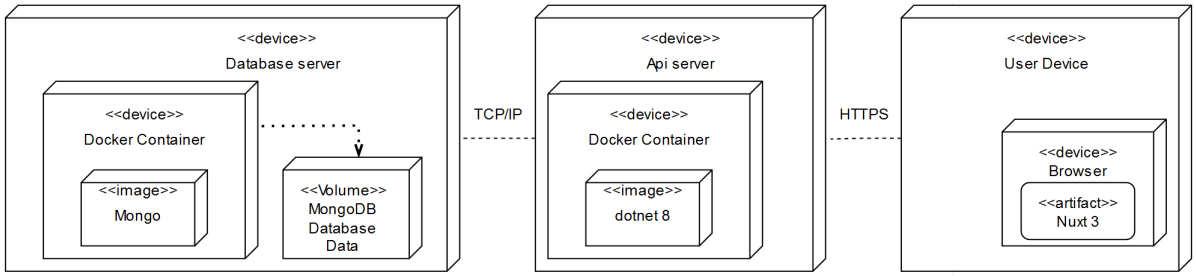
Figuur 2.12: Deployment diagram van het afstudeer product



2.5 Physical view

In de physical view wordt er gekeken naar hoe het systeem gedeployed moet worden en waar. Om dit in beeld te brengen is er gebruikgemaakt van een development diagram. In figuur 2.13 is te zien dat er gebruik gemaakt van Docker. Docker is een middel waarmee je software kan laten draaien op elke machine (Docker.com, g.d.). Er is voor gekozen om docker te gebruiken, zodat Snakeware niet aan een cloud provider vast hoeft te zitten.

Figuur 2.13: Deployment diagram van het afstudeer product



Bibliografie

Docker.com. (g.d.). *What is Docker*. Verkregen 28 februari 2024, van https://www.docker.com/

Klijn, D. (2023, augustus). *Onderzoeksverslag* (PDF) (Verkregen febuari 2024). NHL Stendejn Hogeschool.

Kruchten, P. (1995, augustus). *The 4+1 View Model of Architecture* (PDF) (Verkregen 15 febuari 2024).

lucidchart. (2023). *UML Use Case Diagram Tutorial*. Verkregen 15 februari 2024, van https://www.typescriptlang.org/

Microsoft. (2022). *Een rondleiding door de C#-taal*. Verkregen 10 oktober 2023, van https://learn.microsoft.com/nl-nl/dotnet/csharp/tour-of-csharp/

Mozilla. (2023a). *Classes*. Verkregen 11 oktober 2023, van https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

Mozilla. (2023b). *JavaScript*. Verkregen 10 oktober 2023, van https://www.javascript.com/

Mozilla. (2023c). *What's in the head? Metadata in HTML*. Verkregen 11 oktober 2023, van https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/The_head_metadata_in_HTML

Postman.com. (2023). *Customzie request order in a collection run*. Verkregen 6 oktober 2023, van https://learning.postman.com/docs/collections/running-collections/building-workflows/

Snakeware. (2022a). *Cases*. Verkregen 10 oktober 2023, van https://www.snakeware.nl/cases

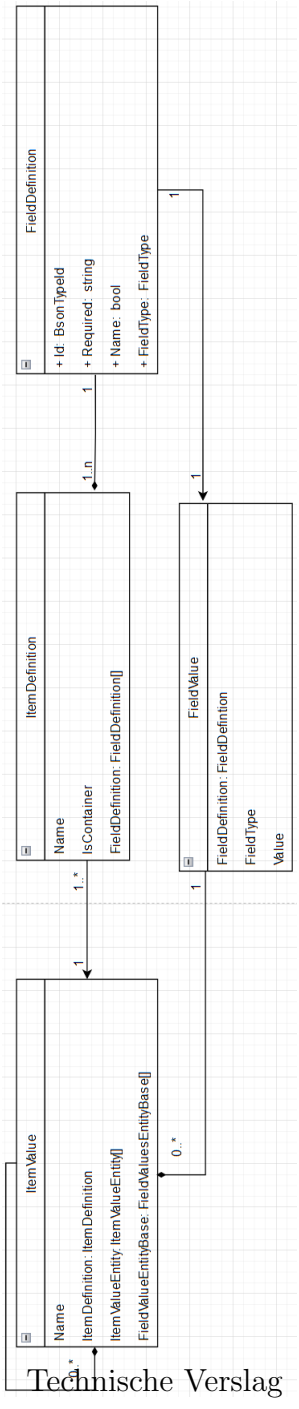
Snakeware. (2022b). *What we do*. Verkregen 10 oktober 2023, van https://www.snakeware.com/what-we-do

Watts, S. (2020). *The importance of SOLID Design Principles*. Verkregen 11 oktober 2023, van https://www.bmc.com/blogs/solid-design-principles/

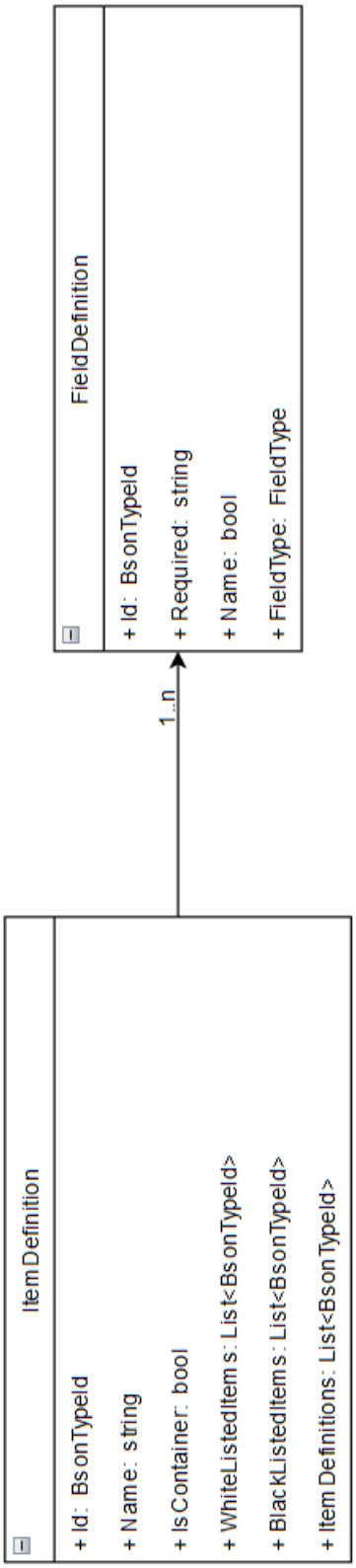
Bijlage A

Figuren UML

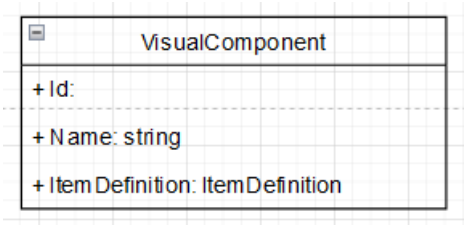
Figuur A.1: Klassendiagram ItemValue



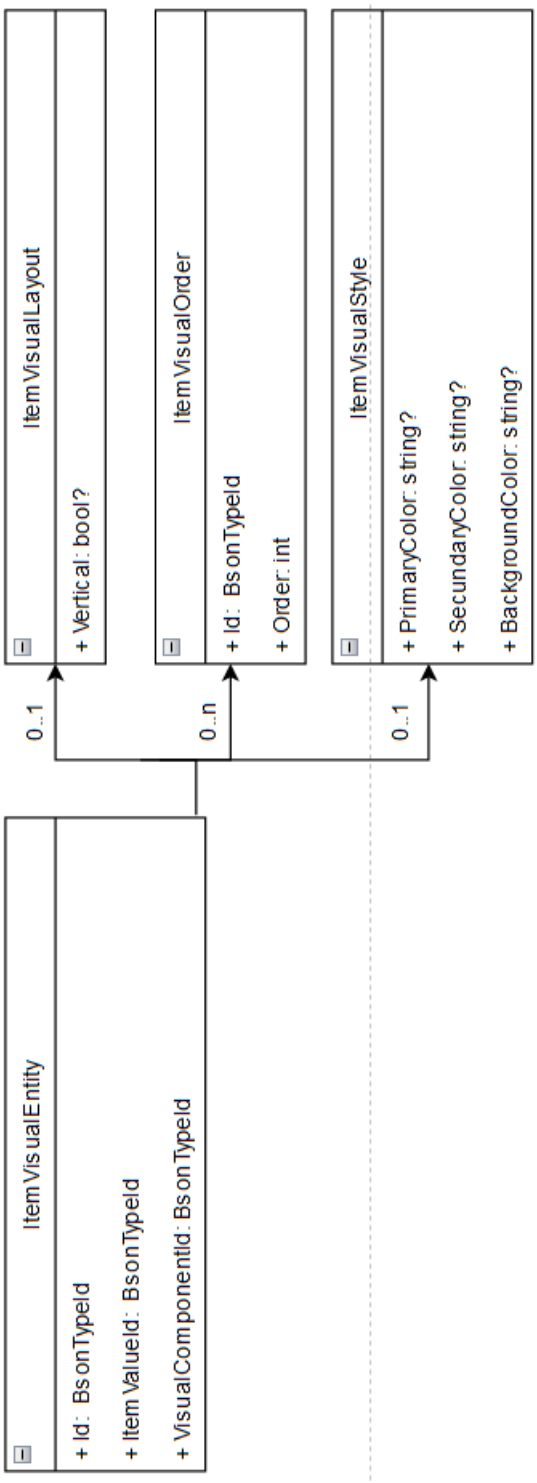
Figuur A.2: Klassendiagram ItemDefinition



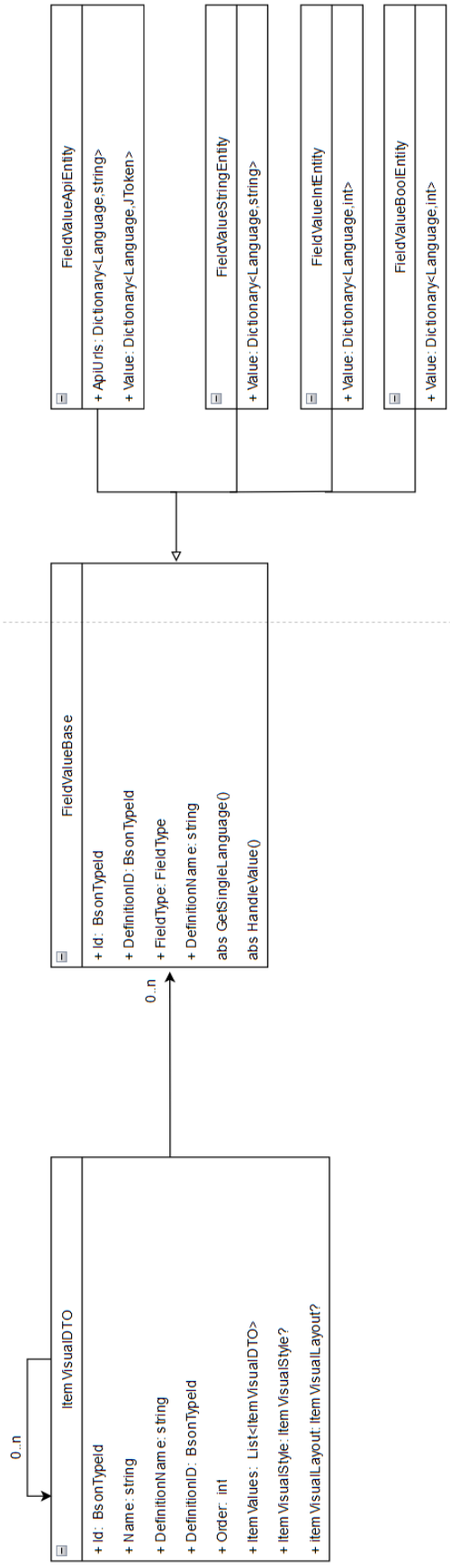
Figuur A.3: Klassendiagram VisualComponent



Figuur A.4: Klassendiagram ItemVisual



Figuur A.5: Klassendiagram ItemVisualDTO



Figuur A.6: Klassendiagram ItemValue