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

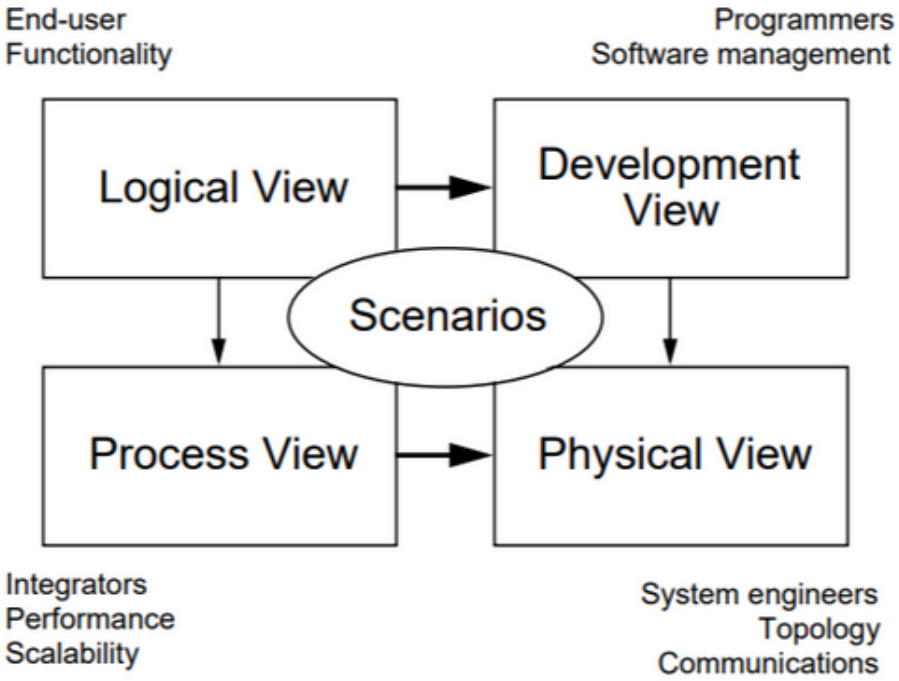
Hoofdstuk 2

Ontwerp

In dit hoofdstuk wordt het ontwerp van het eindproduct beschreven. Het ontwerp is uitgewerkt op basis van de requirements en randvoorwaarden die vastgesteld zijn in de requirement analyse. Deze requirements zijn terug te vinden in (Klijn, 2023).

Om het ontwerp proces op een gestandaardiseerde methode te maken is er gebruikt van het 4 + 1 view model. Het 4+1 model maakt gebruik van 5 verschillende perspectieven van de software. Deze perspectieven zijn Scenarios, Logical, Development, Process en Physical. In de volgende secties worden de perspectieven uitgelegd en ingevuld aan de hand van de afstudeeropdracht.

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

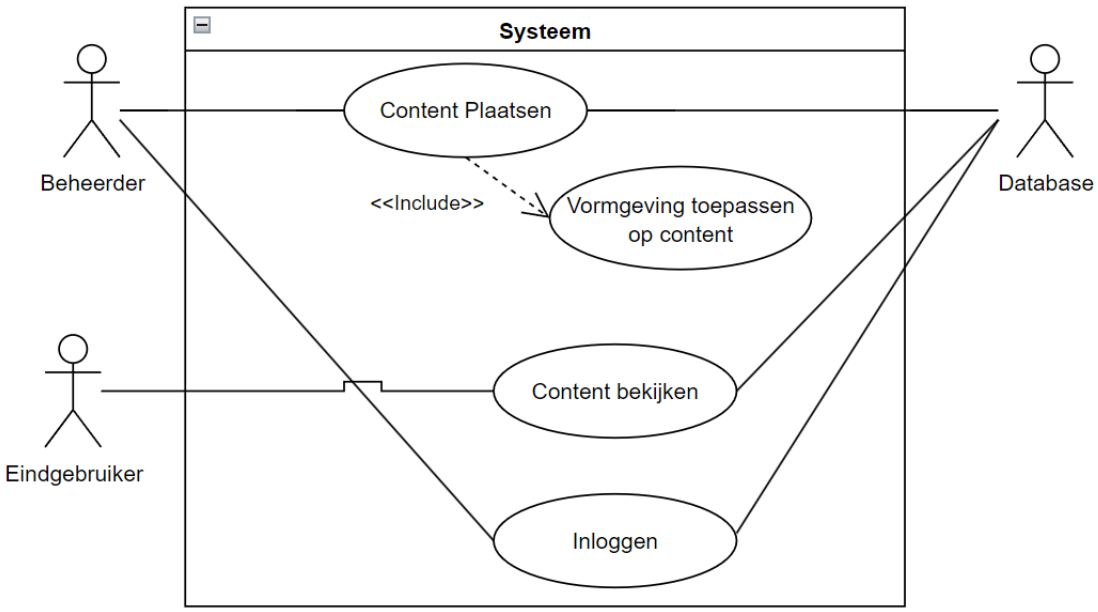


2.1 Scenario's

De scenario's zijn de meest belangrijkste usecases en de interactie tussen abstracte componenten in het systeem (Kruchten, 1995). De verschillende usecases zijn opgesteld doormiddel van de resultaten van het onderzoek (Klijn, 2023) Om de verschillende usecases en interactie met andere actoren in het systeem in beeld te krijgen wordt er gebruik gemaakt van een usecase diagram (lucidchart, 2023).

Alle verschillende user stories zijn vertegenwoordigt in het usecase diagram zie figuur 2.2. Het Content plaatsen vertegenwoordigt de stories (KB-FR 1,4,5,8,9,10, SW-FR14 en SW-FR15). Vormgeven van de content wordt gerpresenteerd door de user stories KB-FR3, KB-FR7 en SW-13. De Content Bekijken van vertegenwoordigt door KB-FR12 en het inloggen door KB-FR6.

Figuur 2.2: Use case diagram



2.2 Logical View

In deze sectie wordt het datamodel end de structuur van de software architectuur toegelicht. Dit wordt gedaan op een abstract niveau zonder implementatie ditails. Er wordt vooral aangetoont hoe de gewenste functionaliteiten worden bereikt en hoe de componenten met elkaar communiseren.

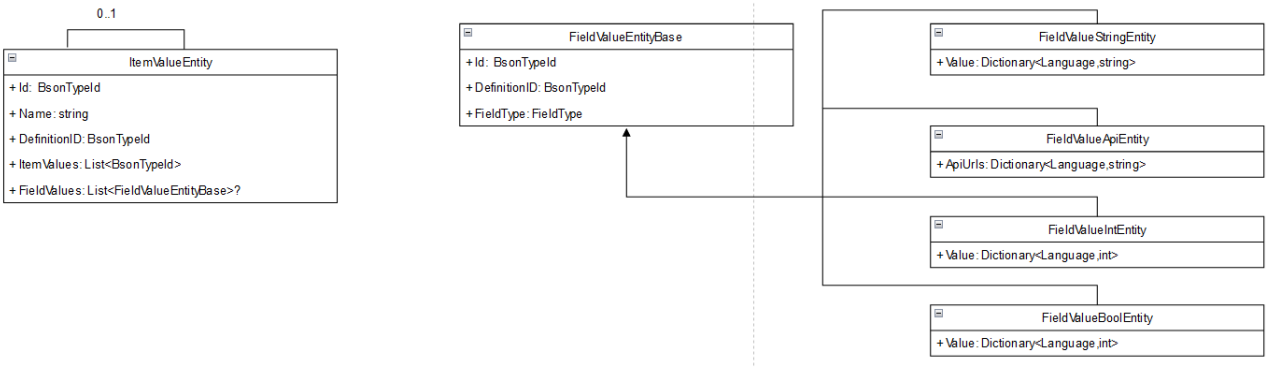
2.2.1 Datamodel

Het Datamodel bestaat uit 4 **objecten** wat in zijn geheel leid naar de uitendelijke data die naar de frontend wordt gestuurd. Deze objecten zijn Item Definition, Item Value, Visual Component en ItemVisual.

Item Value: De Item Value slaat encapsuleert de content/data van het systeem op. Item Values kunnen meerdere Item values bevatten, waardoor je een geneste structuur krijgt. De Item value bevat ook 1 of meerdere FieldValueEntityBase. Deze abstracte class zorgt

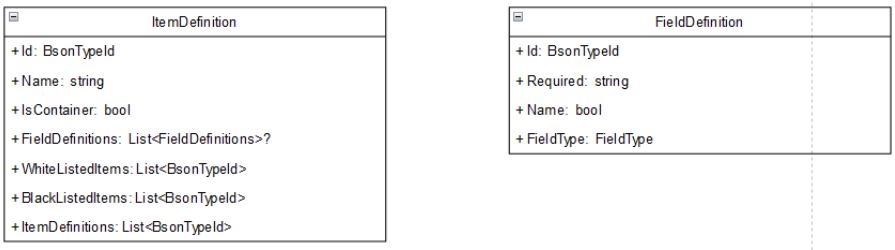
er voor dat er verschillende data types kunnen opgeslagen worden in het het zelfde item. In het klassen diagram (zie figuur 2.3) zijn de huidige implementatie mogelijkheden. Deze zijn String,Bool,Int en Api deze lijst zou in de toekomst uitgebruikt kunnen worden. Een bezondere is de Api FieldType, met dit veld maakt je een http get request naar een ander url. Hierdoor kan je dynamisch externe content ophalen, bijvoorbeelde de Oembed data van een youtube video.

Figuur 2.3: klassen diagram ItemValue



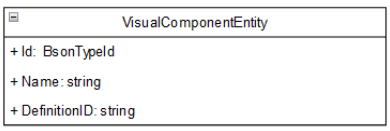
Item Definition: Om structuur te geven aan de Item values wordt er gebruik gemaakt van een item Definition. De belangrijkste functionaliteit van de definition is om aan te geven welke velden er op verschillende items zitten en welke daarvan verplicht zijn.

Figuur 2.4: klassen diagram ItemDefinition



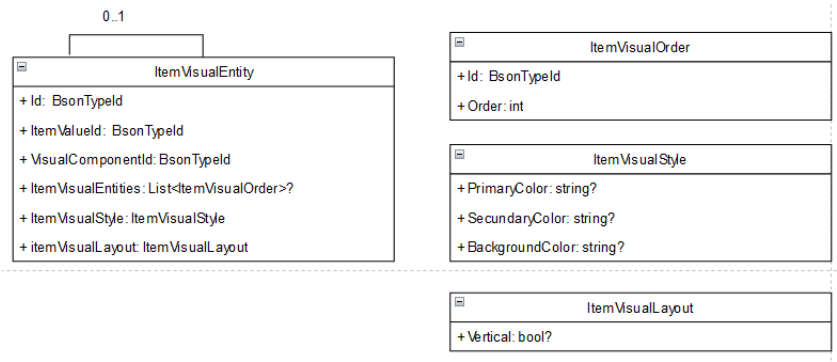
VisualComponent: Om de data te renderen moeten er components gebruikt worden in de frontend om dit af te handelen waar nodig. De VisualComponent wordt gebruikt om deze componenten aan te geven welke er zijn en welke definition er bij hoordt.

Figuur 2.5: Klassen diagram VisualComponent



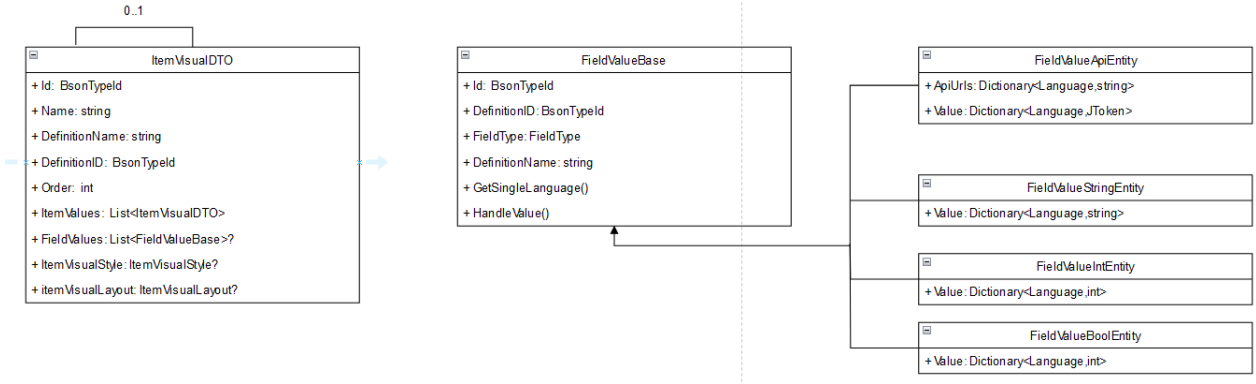
ItemVisual: Dit is het object dat de VisualComponent en de Item Value samen voegt tot een geheel waardoor er content gerenderd kan worden op de pagina. Verder geeft dit object ook aan welke mogelijke stijling of layout op het item moet worden toegepast.

Figuur 2.6: Klassen diagram ItemVisual



Om de data te renderen op een froentend wordt er gebruik van de ItemVisualDTO zie figuur 2.7. De data wordt hier genest en geordend op basis van de ordering.

Figuur 2.7: Klassen diagram ItemVisual



2.2.2 Software architectuur

Om de onderhoudbaarheid te verbeteren van het huidige CMS is er ook nagedacht over het opzetten van de code. De principe houd het volgende in:

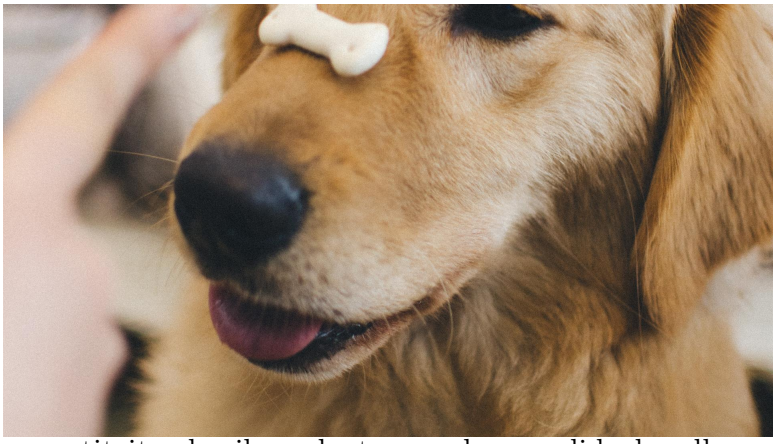
1. **Single-responsibility principle** Dit betekent dat een class, maar 1 verantwoordelijk mag hebben en bij definitie ook maar 1 taak. Als een class te veel verantwoordelijkheden heeft dan wordt het moeilijker om de code te begrijpen en aan te passen.
2. **Open-closed principle** Dit houdt in als class of functie of een andere software 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 LSP verhoog je de consistentiteit en de verwachte uitkomst van je progamma.
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 kan maken. Dit uit zich vaak in plaats van grote interface gebruik meerdere kleinere interfaces. Deze interfaces zijn verantwoordelijk voor meer specifike

usecases inplaats 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.

Deze principes hebben geleid tot de keuze van een handler software architectuur. De handler structuur heeft elementen van het Chain of responsibility design pattern. Hierbij heeft elke handler 1 functie waar die aan voldoet. Hierdoor blijft de code makkelijk te begrijpen en beter onderhoudbaar.

Figuur 2.8: Sequence diagram Handler structuur



- misschien software entiteit gebruiken plaats van class - solid - handlers - units of work aan kaarten plaatje als voor beeld

2.3 Development view

Dit wordt de development view

2.4 Process View

dit wordt de process view

2.5 Physical view

dit wordt de Physical view

Hoofdstuk 3

Realisatie

dit is het realisatie hoofdtstuk ER is voor gekozen om niet overal comments bij te zetten omdat de code (naar mijn mening) heel erg clutterd.

3.1 Tools

Er zijn 2 verschillende applicaties ontwikkeld dit zijn de frontend en de backend applicatie.

Frontend

Voor de frontend is er gekozen om te werken met Vue en Nuxt 3. Vue is een Javascript / Typescript framework dat het makkelijker maakt om grotere frontenden te maken voor web applicaties. Nuxt 3 is een Vue framework wat het makkelijker maakt om vue applicaties te maken en meerdere rendering opties aan bied. Er is voor Vue en Nuxt 3 gekozen om dat de gebruikte standaard is binnen Snakeware. Door dit te doen kunnen de frontend developers snel het proof of concept op pakken. Voor het schrijven van de frontend code is gebruik gemaakt van **Neo Vim / webstorm**.

Backend

Voor de Backend is gebruik gemaakt van C# en .Net 8 een C# framework. Dit is gedaan omdat C# de taal is waar alle andere services en developers gebuik van maken. Er is voor .Net 8 gekozen omdat de laatste *LTS* versie is van .Net. De code voor de backend wordt geschreven in Visual Studio 2022.

Database Er is voor een NoSQL database gekozen (kijk naar ontwerp). Voor de NoSQL database is er gekozen voor MongoDB. MongoDB is de grootste NoSQL database provider en heeft goede drivers voor C#.

Hoofdstuk 4

Beheer, Validatie en Verificatie

In dit hoofdstuk wordt beschreven hoe de kwaliteit van het eindproduct is bewaard. Eerst wordt in beeld gebracht hoe de broncode op kwaliteit gebracht. Daarna wordt er beschreven hoe het ontwikkelproces is uitgevoerd en bewaakt.

4.1 Codereviews en Afstudeerstage voortgang

Tijdens het ontwikkelproces is gebruik gemaakt van codereviews, deze codereviews werden uitgevoerd door specialisten van hun discipline. Voor de backend code is Kevin Snijder (Backend software engineer) benaderd, en voor de frontend is ..... (frontend developer).

Nog een frontender vragen om dit te doen

Voor het bijhouden van de afstudeerstage progressie is er wekelijks een afspraak met Thom Koenders (Senior software engineer). Ook hier werd vaak de code besproken en waar mogelijk verbeteringen gemaakt kunnen worden.

4.2 Versiebeheer

Voor het versiebeheer van de broncode is er gebruik gemaakt van git en Bitbucket als git repository platform. Voor elke nieuwe functionaliteit is er een nieuwe branche aangemaakt waar op ontwikkeld werd. Wanneer de functionaliteit klaar was werd er een pull request gemaakt die vervolgens gemerged werd.

4.3 Codestandaarden

Tijdens het ontwikkelen van de afstudeeropdracht is er gecontrolleerd op de code standaarden die gebruikt zijn. Omdat Snakeware niet een vaste guideline heeft voor code standaarden is er zelf een opgesteld die gevolgt is tijdens de afstudeeropdracht. Deze standaarden staan vermeld in de bijlage **(bron)**.

4.4 Process en bewaking

Tijdens de afstudeerperiode is er gewerkt met een agile methode genaamd scrum. Er is gewerkt met sprints van 2 weken waar bij de requirements opgedeeld werden in kleine stukjes.

Dit is gedaan om de taken behapbaar te maken en dat er snel verandering gemaakt kon worden. Elke week werdt er met de afstudeerbegeleider de progressie gesproken van de week waar bij mogelijk bijgestuurd werdt. De verschillende sprints werden bij gehouden in een notitie programma **Obsidian** hierbij is elke sprint opgedeeld in een apparte note. Vervolgens is er gebruik gemaakt van een scrumboard om de progressie bij te houden dit werd ook gedaan in Obsidian.

4.5 Testrapport

Test raport

Hoofdstuk 5

Conclusie

dit is de conclusie

Hoofdstuk 6

Reflectie

dit is de reflectie

Bibliografie

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/