TECHNISCH ONTWERPRAPPORT

Sippr.

Periode: 4, Backend-development en databases
Studenten: Alexander Boogaard, Aminata Diallo, Danny van Benthem, Isja Nieskens
Studentnummers: 1116228(Alexander), 2208686 (Aminata), 2223502(Danny), 2215679(Isja)
Revisie: 0
Docent: K. Rozendaal
Datum: 16-06-2024

Inhoudsopgave

1 Inleiding

Met de voltooide analyse van de functionaliteiten van de beoogde webshop, zoals beschreven in het functioneel ontwerprapport (Boogaard et al., 2023)en de formele goedkeuring van de opdrachtgever 'Sippr.', kunnen nu de technische randvoorwaarden worden beschreven. Dat wordt gedaan in dit rapport, het Technisch Ontwerp (TO). Het is van essentieel belang dat het TO alle nodige informatie bevat voor een soepele implementatie en dat het begrijpelijk is voor de belanghebbenden.

Dit rapport biedt een gedetailleerde blik op de technische randvoorwaarden die essentieel zijn voor het succesvol implementeren van de webshop. Door te denken aan architectuur, databases, beveiliging en andere aspecten van het systeem, streven we ernaar een solide basis te leggen voor de uiteindelijke webshop.

Het technisch ontwerp wordt gebruikt als overdrachtsdocument naar de programmeurs van Devs2B, die verantwoordelijk zijn voor de realisatie van de webshop.

In periode 2 is de basis gelegd voor dit document, terwijl we ons in deze derde periode specifiek richten op het databasegedeelte. Als reactie op de ontvangen feedback tijdens de beoordelingen van periode 2 zijn er ook andere aanpassingen doorgevoerd. Waardoor de structuur van de documenten aanzienlijk is gewijzigd. Om deze reden hebben we besloten een geheel nieuwe versie van dit document te creëren in plaats van een revisie toe te passen.

2 UML-diagrammen

Om een goede beschrijving van een informatiesysteem te kunnen maken is er door Devs2B een standaard diagrammen set samengesteld die wordt gebruikt voor het modeleren en documenteren van software-en systeemontwerpen. Deze set wordt gebruikt door software-engineers, systeemanalisten en andere belanghebbenden om complexe systemen te begrijpen, ontwerpen en te communiceren. De kracht van UML ligt in het vermogen om abstracte concepten te vertalen naar visuele representaties. Door gebruik te maken van gestandaardiseerde symbolen en relaties kunnen UML-diagrammen fungeren als een communicatiemiddel tussen ontwikkelaars, ontwerpers en andere belangstellenden (Warmer & Kleppe, 2011).

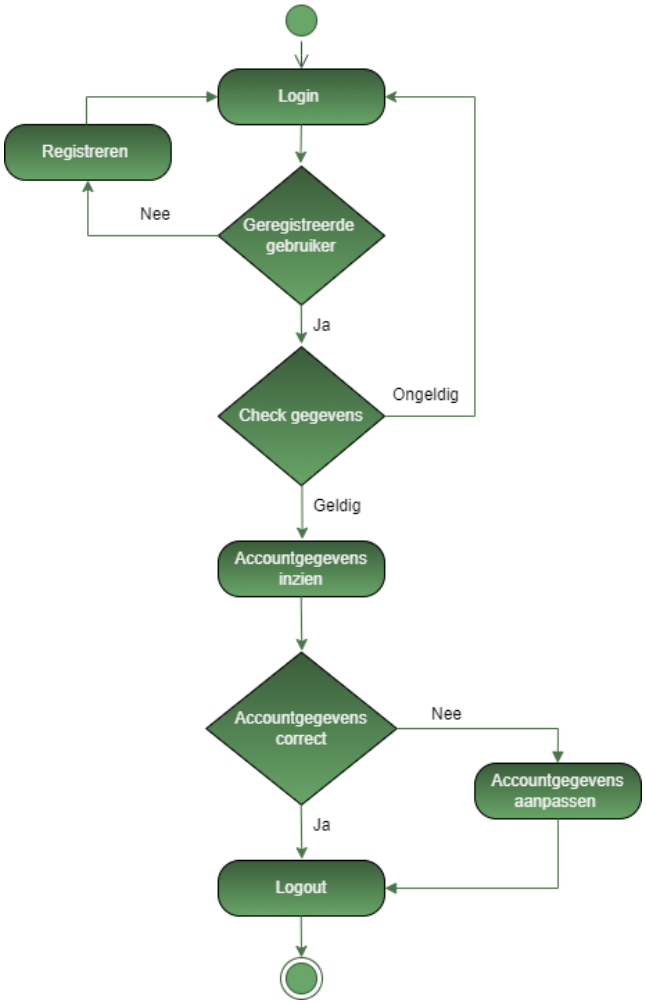
2.1 Activiteitendiagrammen

Activiteitendiagrammen beschrijven processen en kunnen voor meerdere doeleinden gebruikt worden. Als eerste kan een activiteitendiagram worden gebruikt om op een hoog niveau een beschrijving van de processen binnen een organisatie weer te geven. Daarnaast kan het ook gebruikt worden om een algoritme te beschrijven wat in essentie ook een proces is. Binnen UML kan hierbij gedacht worden aan use-cases of operaties.

Activiteitendiagrammen binnen UML2 worden veelvuldig gebruikt om de operaties van een klasse te beschrijven. Het is echter essentieel om ervoor te zorgen dat het abstractieniveau van het activiteitendiagram hoger blijft dan dat van de bijbehorende programmacode. Over het algemeen worden activiteitendiagrammen gebruikt om op een overzichtelijke manier een beperkt aantal operaties weer te geven (Warmer & Kleppe, 2011).

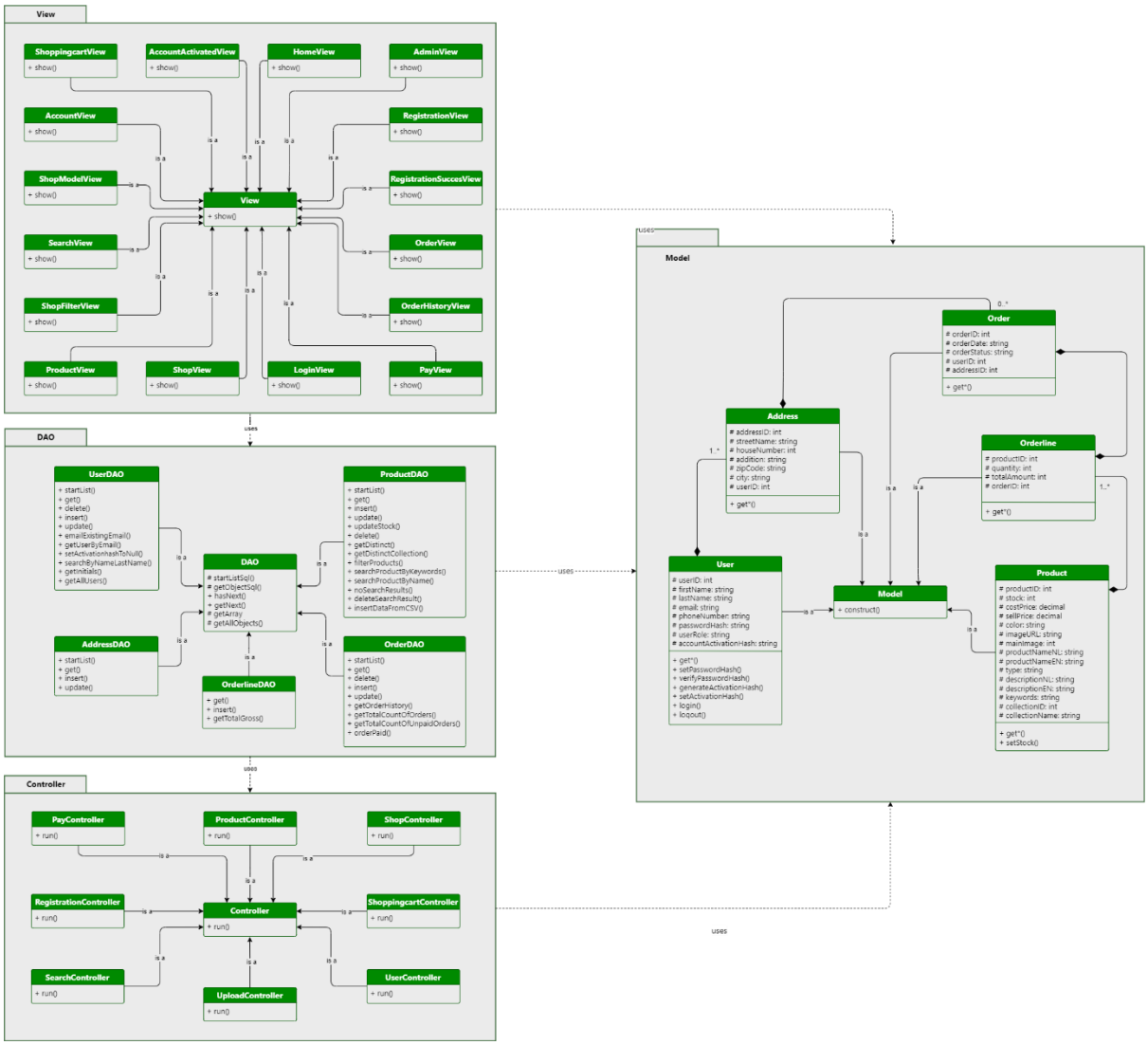
In **Figuur 1** wordt een activiteitendiagram getoond voor de activiteiten die een klant kan uitvoeren om in te loggen.

Figuur 1. Activiteitendiagram



2.2 Klassendiagram

In



vindt u het klassendiagram dat wordt gebruikt om de structuur van een systeem weer te geven. Een klassendiagram is een type diagram in de Unified Modeling Language (UML) dat de structuur van een systeem modelleert door klassen, hun eigenschappen en onderlinge relaties weer te geven. Klassen vertegenwoordigen typen objecten in het systeem, zoals entiteiten, concepten of modules. De eigenschappen van een klasse worden weergegeven als attributen, terwijl de relaties tussen klassen worden aangegeven met associaties.

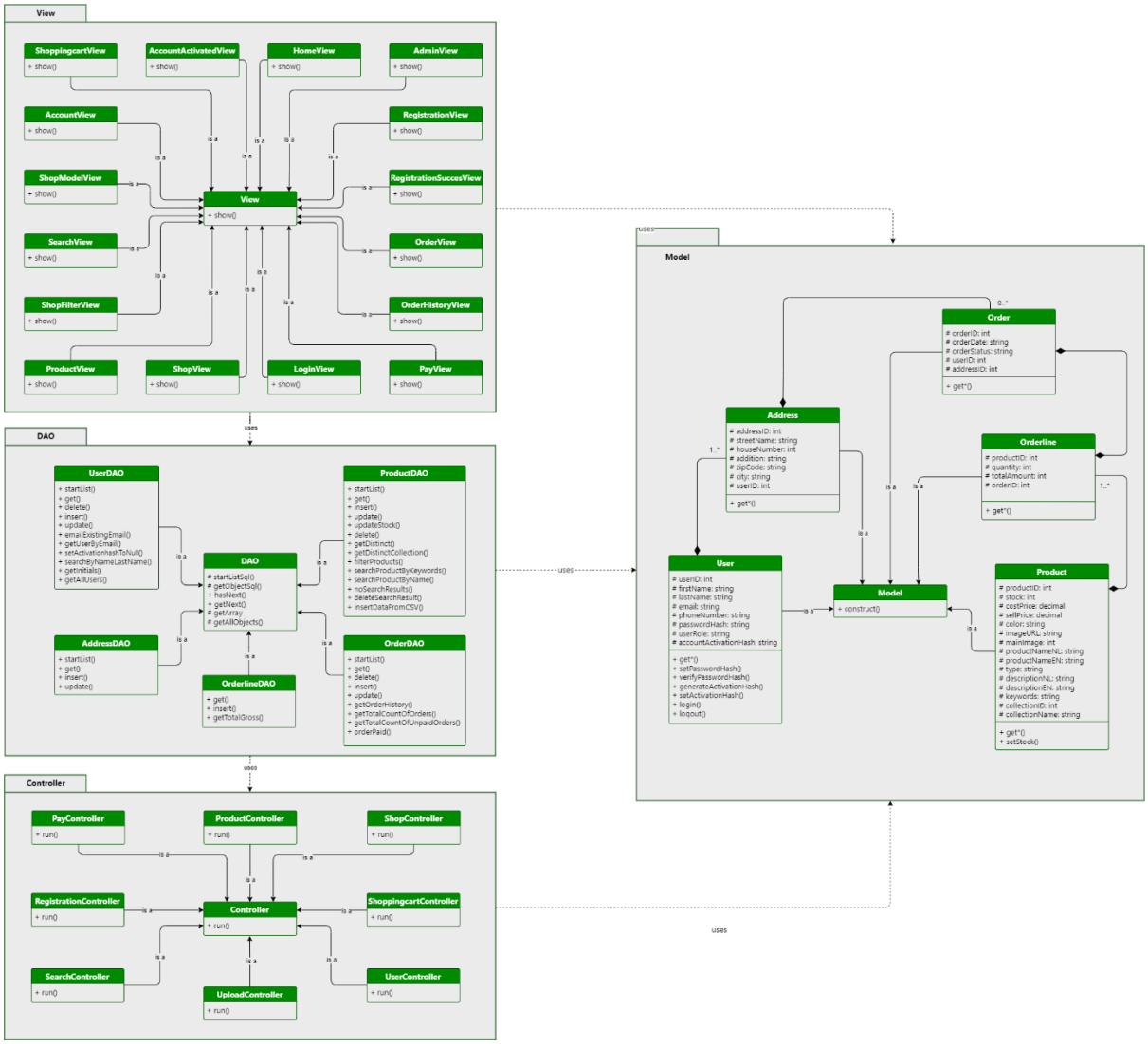
Klassendiagrammen worden veel gebruikt in software engineering om de architectuur van een systeem te begrijpen, de interacties tussen verschillende componenten te analyseren en als basis te leggen voor het ontwerpen en implementeren van software.

Elke klasse heeft bepaalde eigenschappen die het definieert, zoals bijvoorbeeld de naam van een gebruiker (User), de prijs van een product (sellPrice), of het type van een dienst. Deze eigenschappen worden in het diagram weergegeven als attributen van de klassen.

Daarnaast toont het klassendiagram ook hoe deze klassen onderling gerelateerd zijn. Deze relaties worden aangegeven met lijnen tussen de klassen, die associaties worden genoemd. Deze associaties laten zien hoe de verschillende klassen met elkaar verbonden zijn en hoe ze samenwerken binnen het systeem.

Het belangrijkste doel van een klassendiagram is om ontwikkelaars (Devs2B) te helpen de structuur en de onderlinge relaties van een systeem te begrijpen. Door het diagram te analyseren, kan men een dieper begrip ontwikkelen van hoe de diverse onderdelen van het systeem met elkaar in verbinding staan en samenwerken.

Figuur 2. Klassendiagram



2.3 Sequencediagram

Het sequencediagram is een visuele representatie van de interactie tussen de verschillende objecten en entiteiten van de toekomstige webshop. Het diagram helpt om objecten te begrijpen en verduidelijkt de logica die wordt toegepast in de code. Het kan dienen als communicatiemiddel voor het ontwikkelteam en andere belanghebbenden waardoor alle neuzen dezelfde richting op staan. Bovendien kan het ook dienen als waardevolle systeemdocumentatie wanneer er een overdracht moet plaatsvinden naar andere ontwikkelaars of onderhoudsteams.

Zoals in de inleiding van dit hoofdstuk is toegelicht, zal niet alles uitgewerkt worden in dit bestand. We hebben besloten om één use-case uit te werken, namelijk het registreren van een gebruiker. Aan de hand van het sequencediagram in **Figuur 3** zal er een Proof of Concept (PoC) worden gemaakt.

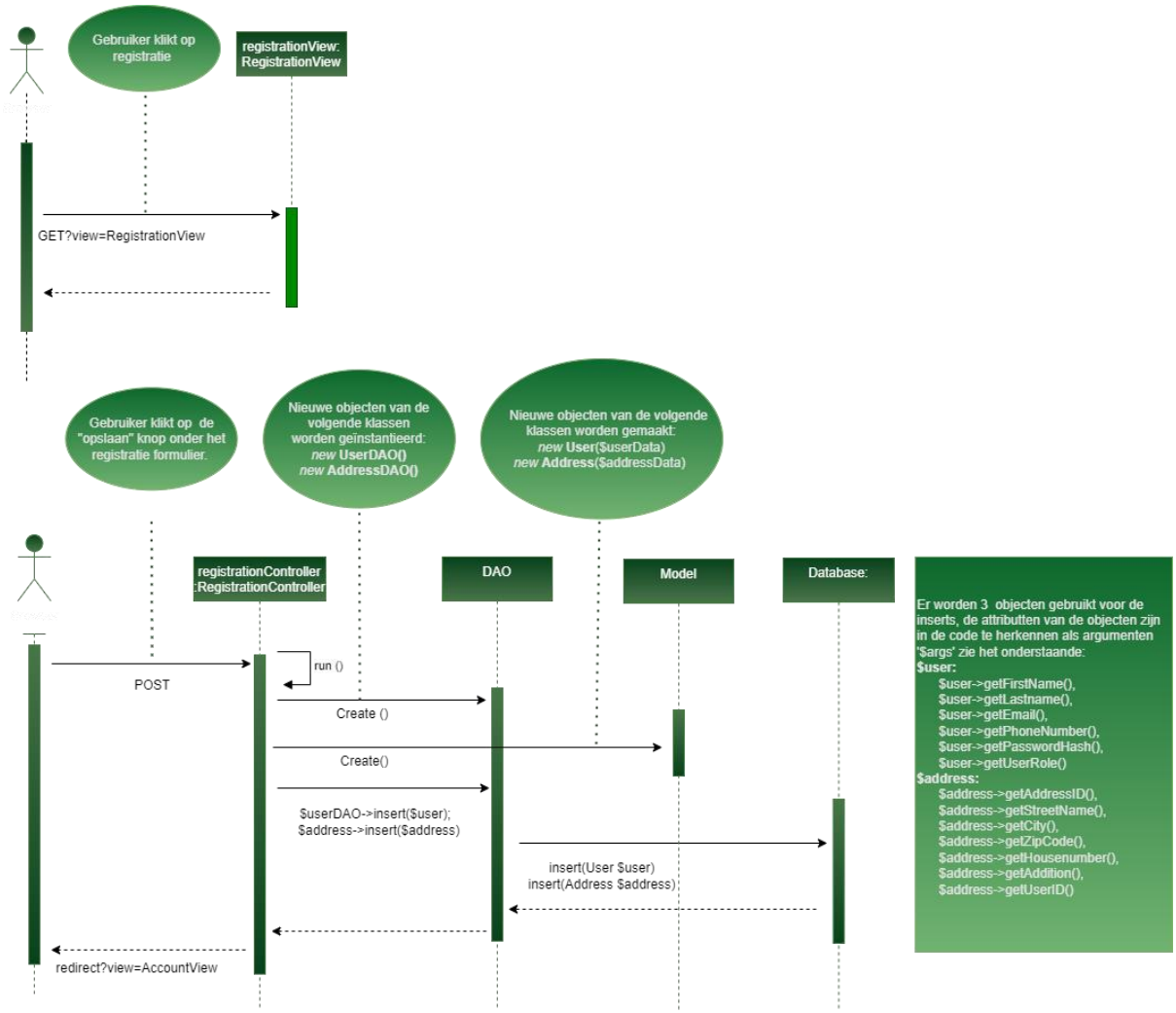
Een PoC is een experiment van een implementatie dat wordt uitgevoerd om de haalbaarheid en de praktische toepasbaarheid van het concept te valideren. Op deze manier kun je achterhalen of het concept realistisch is, en kun je dit ook aantoonbaar maken. Een PoC maakt een concept meetbaar. In het Proof of Concept laten we zien dat een gebruiker via het registratieformulier een account kan aanmaken. Om te bewijzen dat er ook daadwerkelijk een account wordt aangemaakt, zul je na het opslaan doorverwezen worden naar het accountoverzicht waar de nieuw aangemaakte gegevens te zien zijn.

In het Proof of Concept (PoC) zijn alle gegevens van alle aangemaakte accounts te zien. Dit is puur bedoeld om te bewijzen dat het PoC werkt en om te laten zien wat er op de achtergrond gebeurt. Bij de daadwerkelijke realisatie van de webshop zal de klant uiteraard alleen zijn eigen gegevens kunnen inzien.

In hoofdstuk 5.2 van dit rapport worden de criteria en de testen benoemd voor de acceptatietest. Aan de hand van deze test kan geconcludeerd worden of het Proof of Concept goedgekeurd wordt.

Het registreren van de gebruiker hoort bij requirement 7.0, uit de requirements lijst. (Boogaard A., Diallo A., Van Benthem D., Nieskens I., & Pearson J, 2023)

Figuur 3. Sequencediagram



3 Architectuur

3.1 Technologiestack

Voor de correcte werking van de webshop zullen verschillende softwarecomponenten worden gebruikt. Deze zijn hieronder onderverdeeld op basis van de verschillende functionaliteiten waarvoor ze worden gebruikt.

Front-end:

HyperText Markup Language (HTML) vormt de kern van het web. Het is een gestandaardiseerde opmaaktaal die gebruikt wordt om de structuur en inhoud van webpagina's te definiëren. Opgebouwd uit elementen die worden weergegeven door tags, zoals headers, paragrafen en lijsten, stelt HTML-ontwikkelaars in staat om tekst, afbeeldingen, multimedia en andere inhoud te integreren en te organiseren op een gestructureerde wijze.

Cascading Style Sheets (CSS) vormen een bijna onmisbaar onderdeel van webontwikkeling, omdat ze de presentatie en stijl van HTML-documenten beheren. CSS stelt ontwikkelaars in staat om de visuele presentatie van webpagina's te definiëren. Zo kunnen ze kleuren, lettertypen, afstanden, lay-outs en andere visuele aspecten aanpassen. Wanneer goed geïmplementeerd bevordert het een scheiding van presentatie en structuur, waardoor ontwikkelaars efficiënter kunnen werken en de onderhoudsinspanningen verminderen.

JavaScript (JS) is een programmeertaal die veel wordt gebruikt in webontwikkeling om de interactie met gebruikers te verbeteren en dynamische content toe te voegen aan webpagina's. In tegenstelling tot HTML en CSS, die voornamelijk verantwoordelijk zijn voor de structuur en presentatie van een webpagina, biedt JavaScript functionaliteit voor het manipuleren van de inhoud en te reageren op gebruikersacties.

Met JavaScript kunnen ontwikkelaars taken automatiseren, formulieren valideren, dynamische updates van pagina-inhoud uitvoeren en interactieve gebruikersinterfaces creëren.

In een webshop wordt JavaScript gebruikt om productfilters toe te passen zonder de pagina te herladen, het besteloverzicht bij te werken zonder een volledige pagina-refresh en om interactieve elementen zoals beeldgalerijen toe te voegen. Het is een onmisbaar onderdeel voor het creëren van een moderne en dynamische gebruikerservaring op een e-commercewebsite.

Back-end technologieen:

PHP: *Hypertext Preprocessor* is een krachtige server-side programmeertaal die specifiek is ontworpen voor webontwikkeling. Het stelt ontwikkelaars in staat dynamische en interactieve webpagina's te bouwen. Doordat PHP server-side is, worden de resultaten van de scriptuitvoering naar de browser gestuurd, waardoor het een essentieel instrument is voor het ontwikkelen van robuuste en interactieve webapplicaties. Het wordt veel gebruikt in combinatie met andere technologieën, zoals HTML, CSS, JS en verschillende databases, om uitgebreide en dynamische weboplossingen te creëren.

Met zijn veelzijdigheid kan PHP verschillende taken uitvoeren, zoals het verwerken van formuliergegevens, genereren van dynamische pagina-inhoud, beheer van gebruikerssessies, en communicatie met databases zoals SQLite, MySQL of PostgreSQL. PHP biedt ook ondersteuning voor objectgeoriënteerd programmeren (OOP) en heeft een uitgebreide set ingebouwde functies, waardoor ontwikkelaars efficiënt kunnen werken. Zo is het bijvoorbeeld mogelijk om gebruik te maken van PDO (PHP Data Objects) dat een implementatie is van het DAO-ontwerppatroon wat verder wordt omschreven in hoofdstuk 3.2 (The PHP Group, z.d.).

Apache is een veelgebruikte webserver die een essentiële rol speelt in het hosten van websites op het internet. Ontwikkeld en onderhouden door de Apache Software Foundation, behoort Apache een van de meest gebruikte webserverprogramma's ter wereld. Als webserver verwerkt Apache verzoeken van clients, meestal webbrowsers, en levert het de bijbehorende webpagina's terug. Het ondersteunt verschillende protocollen, waaronder HTTP en HTTPS (met behulp van SSL/TLS), en biedt functies zoals virtuele hosting, URL-omleidingen, en toegangscontrole via configuratiebestanden. Apache is configureerbaar en uitbreidbaar dankzij modules, waardoor ontwikkelaars extra functionaliteit kunnen toevoegen die voldoet aan de specifieke eisen van hun webapplicaties (Website Rating, 2023).

Database:

SQLite is een lichtgewicht, relationele databasesysteem dat zich onderscheidt door zijn eenvoud en minimale configuratievereisten. Het is ontworpen als een zelfstandige, serverloze database, waardoor het gegevens kan opslaan in een enkel bestand, wat het gemakkelijk maakt om te integreren en te beheren is in verschillende applicaties zonder de noodzaak van een aparte database-server. Ondanks zijn compacte formaat ondersteunt SQLite de meeste SQL-functies. Vanwege zijn gebruiksgemak, draagbaarheid en brede inzetbaarheid wordt SQLite vaak gekozen als een lokale opslagoplossing voor mobiele apps, desktopapplicaties en andere situaties waar eenvoudige implementatie en beheer de voorkeur hebben (ostezer & Drake, 2022). Deze database kan gemakkelijk worden beheerd en bewerkt met behulp van de tool DB Browser. Hiermee kunnen gebruikers de structuur van de database bekijken en query's uitvoeren zonder uitgebreide kennis van complexe databasebeheersystemen te hebben.

Beveiliging:

SSL-certificaten, afgeleid van Secure Sockets Layer, zijn digitale certificaten die worden gebruikt om een beveiligde verbinding tot stand te brengen tussen een webserver en een browser. Deze certificaten zorgen ervoor dat gegevens die tussen de server en de gebruiker worden overgedragen, worden versleuteld, waardoor de privacy en veiligheid van informatie worden gewaarborgd. Bovendien authenticeren SSL-certificaten de identiteit van een website, waardoor gebruikers kunnen vertrouwen dat ze verbinding maken met de beoogde website en niet met een kwaadwillende partij.

HTTPS, wat staat voor HyperText Transfer Protocol Secure, is het resultaat van het implementeren van SSL-certificaten en geeft aan dat de communicatie met de website beveiligd is. SSL-certificaten kunnen worden verkregen van certificeringsinstanties, zoals Let's Encrypt, en kunnen variëren in termen van validatie (bijvoorbeeld domeinvalidatie, organisatievalidatie of uitgebreide validatie) en looptijd. Ze zijn van cruciaal belang voor het waarborgen van de beveiliging van gegevensoverdracht op het internet, vooral voor websites die persoonlijke of gevoelige informatie verwerken (Cloudflare, z.d.).

Testing Framework:

PHPUnit is een krachtig test framework voor PHP dat ontwikkelaars helpt bij het schrijven en uitvoeren van tests voor hun PHP-code. Met PHPUnit kunnen ontwikkelaars unit tests maken om de functionaliteit van individuele eenheden van code, zoals klassen of methoden, te verifiëren (Aweda, 2022). PHPUnit biedt een gestructureerd framework voor het schrijven van testcases, het uitvoeren van tests, en het rapporteren van resultaten. Door regelmatig gebruik te maken van PHPUnit kunnen ontwikkelaars de betrouwbaarheid van hun code verbeteren, regressiefouten opsporen en de algehele stabiliteit van hun PHP-toepassingen waarborgen (Patric, 2023).

3.2 Ontwerppatronen

Bij Object Oriented Programming (OOP) is een gestructureerde aanpak vereist om complexiteit en onder houdbaarheid te vergroten. Twee veelgebruikte ontwerppatronen bieden een robuuste basis voor het ontwikkelen van schaalbare en onderhoudbare applicaties. Deze patronen zijn het Model-View-Controller (MVC) patroon en het Data Access Object patroon.

3.2.1 Model-View-Controller

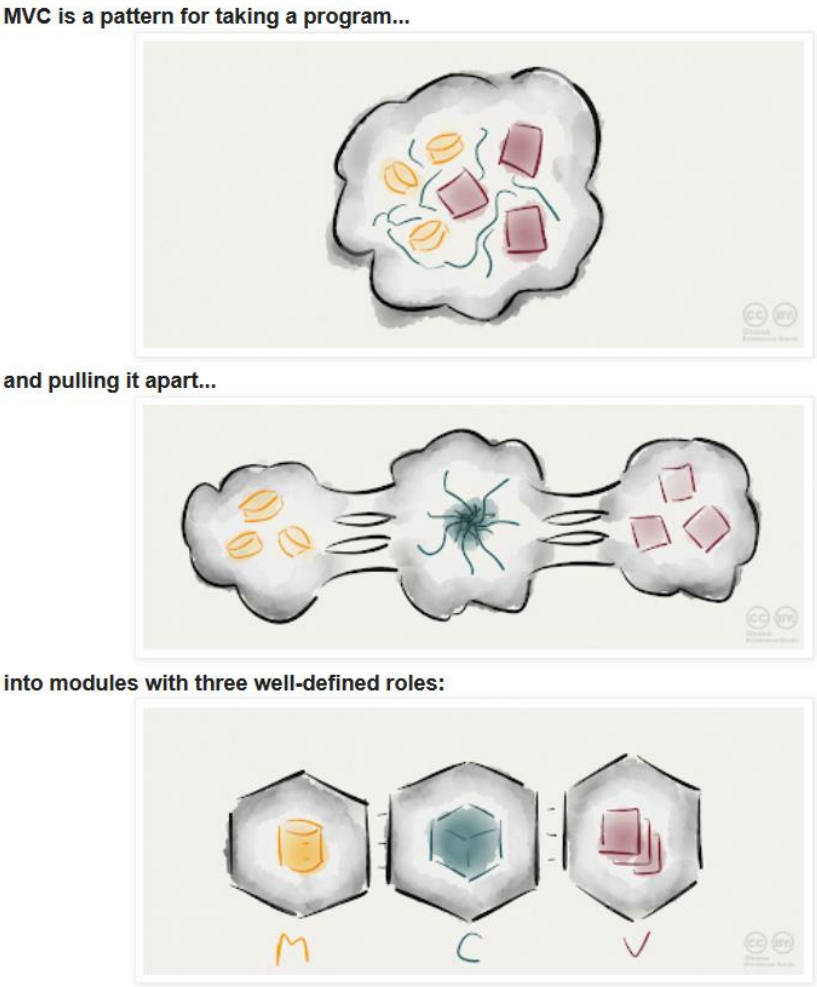
Het MVC-patroon is een architecturaal ontwerppatroon dat zich richt op het scheiden van de presentatie, logica en gegevens van een applicatie. Hierbij worden drie hoofdcomponenten geïdentificeerd: het Model, de View en de Controller (Brinkman-Davis, 2012).

Wanneer code geschreven wordt zonder een duidelijke scheiding van functies, ontstaat het risico van spaghetti-code. Dit is een rommelige structuur waarin het moeilijk wordt bij te houden welke code op welk moment wordt aangeroepen. Het gevolg is een gebrek aan overzicht en complexiteit bij het onderhouden en aanpassen van de code. Wanneer bij de start van een project een onderverdeling gemaakt wordt in de functies van de code, kunnen de bestanden gegroepeerd opgeslagen worden wat de overzichtelijkheid bevordert. Ook maakt dit het makkelijker om, zonder consequenties voor de andere delen van de code, codeblokken aan te passen.

Het MVC-ontwerppatroon stimuleert niet alleen een betere organisatie van de code, maar ook een flexibeler en onderhoudbaar ontwikkelproces.

Een visuele representatie wat de correcte toepassing van MVC voor code kan doen staat hieronder in **Figuur 4** weergegeven. In **Figuur 5** is de structuur van een op MVC-gebaseerd programma weergegeven.

Figuur 4. *Visuele representatie MVC-ontwerppatroon*



Opmerking. Overgenomen uit *Essence of MVC* door S. Brinkman-Davis, 4 december 2012 (https://www.essenceandartifact.com/2012/12/the-essence-of-mvc.html). Copyright december 2012, S. Brinkman-Davis.

***Figuur 5**. Visuele representatie van een op MVC gebaseerd programma*

Opmerking. Overgenomen uit *Essence of MVC* door S. Brinkman-Davis, 4 december 2012 (https://www.essenceandartifact.com/2012/12/the-essence-of-mvc.html). Copyright december 2012, S. Brinkman-Davis.

De toepassing van het MVC-ontwerppatroon impliceert het groeperen van bij elkaar horende code in drie hoofdcomponenten:

Het Model:

Het Model vertegenwoordigt de kern van de applicatie, waarin gegevenslogica en database-interactie plaatsvinden. De business logica, inclusief validatie en gegevensmanipulatie, is hier geïmplementeerd.

De View:

De View is verantwoordelijk voor het presenteren van informatie aan de gebruiker op een visuele manier. Hier wordt de gebruikersinterface ontworpen met aandacht voor responsief design om een consistente ervaring op diverse apparaten te waarborgen. De View haalt gegevens op van het Model en zorgt voor een heldere presentatie aan de eindgebruiker.

Meerdere views kunnen dezelfde gegevens van het model presenteren op verschillende manieren. De View heeft toegang tot bedrijfsgegevens via het Model en specificeert hoe deze gegevens moeten worden weergegeven.

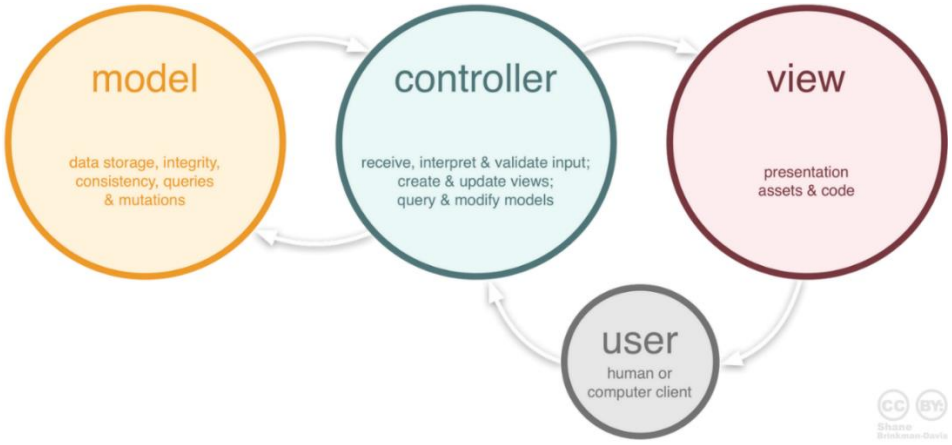
De Controller:

De Controller fungeert als de tussenlaag tussen de gebruiker en het systeem. Het ontvangt gebruikersinvoer van de View, verwerkt deze en roept vervolgens de juiste acties aan in het Model.

De Controller implementeert de routing en gebruikersinvoer verwerking. Het coördineert de interactie tussen View en Model door de juiste acties op het Model aan te roepen. Hier wordt gezorgd voor een soepele navigatie en respons op gebruikersinteractie.

In **Figuur 6** wordt een schematische weergave van het MVC-patroon getoond.

Figuur 6. MVC-patroon



Opmerking. Overgenomen uit *Essence of MVC* door S. Brinkman-Davis, 4 december 2012 (https://www.essenceandartifact.com/2012/12/the-essence-of-mvc.html). Copyright december 2012, S. Brinkman-Davis.

3.2.2 Data Access Object

Naast MVC is het DAO-ontwerppatroon een essentieel onderdeel van effectief softwareontwerp. Het richt zich op het scheiden van de gegevensbron (bijv. een database) van de rest van de applicatie. Het vormt een integraal onderdeel van het Model van MVC. Het neemt specifieke functionaliteiten over waardoor de rest van het Model zich kan concentreren op de kernlogica van de applicatie zonder rechtstreeks met de database te communiceren

Het DAO Patroon:

Het Data Access Object (DAO) patroon wordt gebruikt om de interactie met de database abstract te maken. Het Model in een MVC-architectuur kan gebruikmaken van het DAO-patroon om gegevens op te halen en bij te werken zonder zich zorgen te maken over de details van databaseconnectiviteit. Het DAO-patroon omvat configuratie van databaseverbindingen, beheer van databaseobjecten en uitvoering van databasequery's. Omdat door middel van DAO de database gescheiden is van de rest van het systeem is het mogelijk om te wisselen van databasemanagementsysteem zonder de werking van het systeem als geheel aan te moeten passen.

3.2.3 Samenhangend geheel

In een samenhangend ontwerp werken MVC en DAO-hand in hand. Het Model, dat de kernlogica en gegevens beheert, kan gebruikmaken van het DAO-patroon om naadloos gegevens uit een database te halen. De View presenteert deze gegevens op een visuele manier aan de gebruiker, en de Controller interpreteert gebruikersinvoer, stuurt instructies naar het Model en eventueel de DAO. Ook coördineert de Controller de algehele gebruikerservaring.

Dankzij MVC kan de ontwikkelaar wijzigingen in de gebruikersinterface (View) implementeren zonder het Model te beïnvloeden en vice versa. Het DAO-patroon maakt het mogelijk om de gegevensbron te wijzigen zonder de kernlogica van de applicatie te beïnvloeden.

In combinatie bieden MVC en DAO een flexibel en modulair raamwerk voor het ontwikkelen van robuuste en onderhoudbare softwareapplicaties. Door het scheiden van verantwoordelijkheden kunnen ontwikkelaars effectief samenwerken en kunnen systemen gemakkelijker worden aangepast aan veranderende vereisten en technologieën.

4 Databaseontwerp

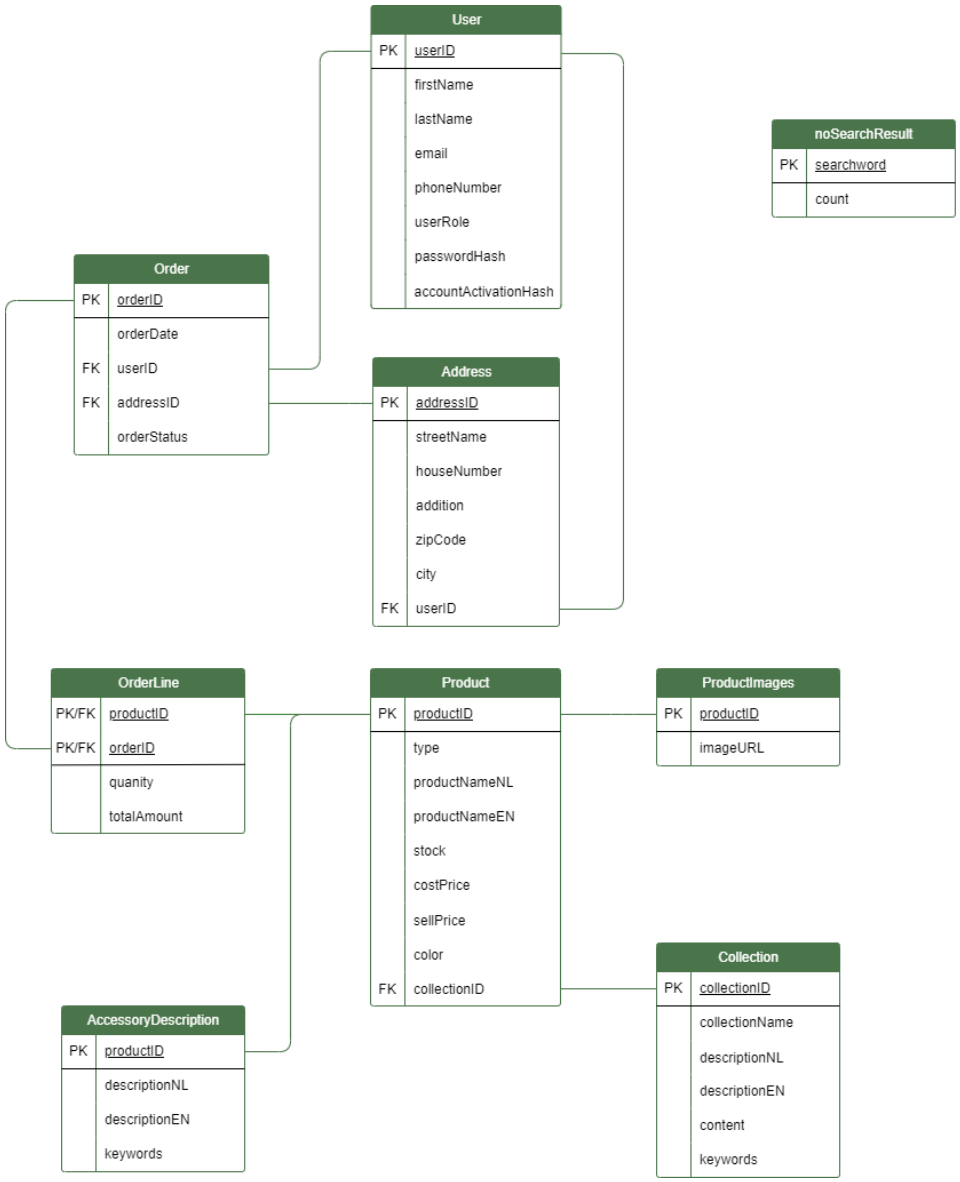
Databaseontwerp speelt een cruciale rol in de effectieve werking van informatiesystemen en is nauw verbonden met het eerder opgestelde functioneel ontwerp. Het biedt een overzicht van de scope, doelstellingen en verwachte inhoud, inclusief belangrijke aspecten zoals entiteiten, relaties en normalisatie. Deze benadering is essentieel voor het gestructureerd opzetten van een goed functionerende database.

4.1 Relationeel model

Het relationele model is een manier om gegevens te structureren in databases. Het draait om het gebruik van tabellen, waarbij elke tabel een verzameling van gerelateerde gegevens bevat. Elke rij in zo'n tabel representeert een specifiek gegevensrecord, terwijl de kolommen de verschillende kenmerken(attributen) van die records voorstellen. Verbanden tussen deze gegevens worden vastgelegd door middel van sleutels (Foreign Keys), waardoor complexe

relaties tussen verschillende tabellen kunnen worden vastgelegd. In wordt het relationeel model weergegeven die geïmplanteerd wordt bij ‘Sippr’.

Figuur 7. Relationeel model



4.2 Datamodellering

4.2.1 Datamodellering

Bij het ontwerpen van onze database, denken wij na over hoe we onze informatie op een slimme manier kunnen organiseren. We bepalen welke soorten informatie we hebben, zoals namen, getallen of datums, en bedenken hoe deze met elkaar in verband staan. Dit is al beschreven en ontwikkeld in het functioneel ontwerp (Boogaard et al., 2023).

We creëren als het ware een blauwdruk voor de database, waarin wordt aangegeven welke gegevens beschikbaar zijn, hoe deze met elkaar in verband staan en op welke wijze ze zijn gestructureerd. Bijvoorbeeld, bij het beheren van gegevens over klanten en bestellingen, wordt nagedacht over de organisatie van informatie met betrekking tot klanten en hun bestellingen in de database.

4.2.2 Attribuut type

In de onderstaande tabellen wordt duidelijk aangegeven welke type alle attributen hebben en of deze attributen een primaire of vreemde sleutel zijn. De structuur is georganiseerd per entiteit, zodat alles direct overgenomen kan worden in de database tabellen.

Tabel 1. *Attributen entiteit User.*

Attribuut	Type	PF/FK
userID	INT (8)	PK
firstName	VARCHAR (20)	
lastName	VARCHAR (60)	
email	VARCHAR (50)	
phoneNumber	INT (15)	
userRole	VARCHAR (20)	
passwordHash	VARCHAR (60)	
accountActivationHash	VARCHAR (60)	

Tabel 2. *Attributen entiteit Address*

Attribuut	Type	PK/FK
addressID	INT (8)	PK
streetName	VARCHAR (50)	
houseNumber	INT (5)	
addition	VARCHAR (4)	
zipCode	VARCHAR (6)	
city	VARCHAR (30)	
userID	INT (8)	FK

Tabel 3. *Attributen entiteit Product*

Attribuut	Type	PK/FK
productID	INT (8)	PK
type	VARCHAR (30)	
productNameNL	VARCHAR (30)	
productNameEN	VARCHAR (30)	
stock	INT (4)	
costPrice	DECIMAL (5,2)	
sellPrice	DECIMAL (5,2)	
color	VARCHAR (20)	
collectionID	INT (8)	FK

Tabel 1

Attribuut	Type	PK/FK
productID	INT (8)	PK
descriptionNL	VARCHAR (300)	
descriptionEN	VARCHAR (300)	
keywords	VARCHAR (300)	

Tabel 7. *Attributen entiteit ProductImages*

Attribuut	Type	PK/FK
imageURL	VARCHAR (300)	PK
productID	INT (8)	FK

Attribuut	Type	PK/FK
colllectionID	INT (8)	PK
collectionName	VARCHAR (30)	
descriptionNL	VARCHAR (300)	
descriptionEN	VARCHAR (300)	
Content	INT (4)	
Keywords	VARCHAR (300)	

Tabel 5. *Attributen entiteit Order*

Attribuut	Type	PK/FK
orderID	INT (8)	PK
orderDate	DATE	
orderStatus	VARCHAR (20)	
userID	INT (8)	FK
addressID	INT (8)	FK

Tabel 6. *Attributen entiteit Orderline*

Attribuut	Type	PK/FK
quantity	INT (4)	
totalAmount	DECIMAL (5,2)	
orderID	INT (8)	PK/FK
productID	INT (8)	PK/FK

Tabel 8. *Attributen entiteit noSearchResult*

Attribuut	Type	PK/FK
searchWord	VARCHAR (80)	PK
count	INT (4)	

De tabel noSearchResult zal enkel te zien zijn op de beheer pagina. Dit is een losse tabel die bijhoud wanneer klanten een zoek opdracht uitvoeren die geen resultaten oplevert.

4.3 Databasebeheer en –structuur

4.3.1 Databasebeheer

We zorgen er zorgvuldig voor dat alleen geautoriseerde gebruikers toegang hebben tot de klantgegevens en dat er regelmatig back-ups worden gemaakt om gegevensverlies te voorkomen. Daarnaast monitoren we de prestaties, zoals de snelheid waarmee de database reageert op verzoeken, en optimaliseren we deze indien nodig.

4.3.2 Databasestructuur

De databasestructuur omvat het ontwerp van tabellen, velden, sleutels en indexen die worden gebruikt om gegevens te organiseren en op te slaan in een database. Elke tabel vertegenwoordigt een specifiek type gegevens, zoals klanten, producten of bestellingen, en bestaat uit rijen en kolommen waarin individuele gegevensvelden worden opgeslagen. Sleutels, zoals de primaire sleutel en vreemde sleutels, worden gebruikt om rijen in een tabel te identificeren en relaties tussen tabellen te definiëren. Indexen worden gebruikt om de zoekprestaties te verbeteren door snelle toegang tot specifieke gegevens mogelijk te maken. Een goed gestructureerde database is essentieel voor het beheren, organiseren en efficiënt benaderen van gegevens, en draagt bij aan de gegevensintegriteit, prestatieoptimalisatie en ontwikkeling van toepassingen.

4.4 Database-interacties

Bij database-interacties gaat het om hoe verschillende componenten van een systeem communiceren met de database, en hoe gegevens worden opgevraagd, bijgewerkt of verwijderd (Osman, 2023). Hier zijn enkele belangrijke aspecten van database-interacties:

4.4.1 Gegevens invoeren (Create)

Het toevoegen van nieuwe gegevens aan de database. Bijvoorbeeld, het invoegen van een nieuw klantrecord of het toevoegen van een nieuw product. Het doel is om records te creëren en deze op een gestructureerde manier in de database op te slaan.

4.4.2 Gegevens opvragen (Read)

Het ophalen van gegevens uit de database zonder deze te wijzingen. Dit omvat het uitvoeren van zoekopdrachten en ophalen van specifieke informatie. Bijvoorbeeld, het lezen van klantgegevens of het opvragen van productinformatie.

4.4.3 Bijwerken van bestaande gegevens (Update)

Bestaande gegevens kunnen worden bijgewerkt om wijzigingen weer te geven. Als een klant zijn adres wijzigt, kan een updatequery worden gebruikt om de relevante informatie in de database aan te passen.

4.4.4 Verwijderen van gegevens (Delete)

Oude of onnodige gegevens kunnen uit de database worden verwijderd. Bijvoorbeeld, als een gebruiker zijn account sluit, kan een verwijderquery worden gebruikt om alle bijbehorende gegevens te wissen.

5 Integratie van invoerbronnen

In dit hoofdstuk ligt de focus op het valideren van de ingevoerde gegevens, het integreren van externe systemen en de verwerking van de gegevens. Ook wordt met behulp van tabellen de normalisatie getoond.

5.1 Invoervalidatie

Om te waarborgen dat ingevoerde gegevens voldoen aan gespecificeerde criteria, zullen de ingevoerde gegevens eerst gevalideerd moeten worden voordat ze worden verwerkt.
De vereisten waaraan de invoergegevens aan moeten voldoen zijn op voorhand gedefinieerd in het functioneel ontwerp, hoofdstuk 3 tabel 4 t/m 9 (Boogaard et al., 2023).

Na het definiëren van de vereisten, wordt de validatie geïmplementeerd op zowel de client-side als de server-side, gebruikmakend van HTML, PHP en JavaScript. Client-side validatie vindt plaats in de browser van de gebruiker, waarbij de gebruikersinvoer direct wordt gecontroleerd. Op deze manier ontvangt de gebruiker direct feedback bij onjuiste invoer, voordat de gegevens naar de server worden verzonden.

In HTML kunnen de volgende validatie attributen gebruikt worden (w3schools, z.d.):

- *Required*: Dit attribuut geeft aan dat het betreffende invoerveld verplicht is. Het voorkomt dat het formulier naar de webserver wordt verzonden als het veld nog leeg is.
- *Pattern*: Dit attribuut maakt het mogelijk om een reguliere expressie te definiëren waaraan de invoer moet voldoen. Het voorkomt dus dat het formulier naar de webserver wordt verzonden op het moment dat niet aan de gestelde eisen wordt voldaan.
- *Min en max*: met deze attributen kun je voor numerieke invoervelden een minimale en maximale waarde instellen.
- *minLength* en *maxLength*: met deze attributen kun je voor tekst invoervelden een minimale en maximale waarde instellen
- *Email, URL, tel*: met deze attributen kan de browser een specifieke validatie uitvoeren voor e-mailadressen, URL'S en telefoonnummers.

Met **reguliere expressies** (RegEx) kun je een string valideren tegen een specifiek patroon. Wanneer de invoer niet overeenkomt met het verwachte patroon, wordt de functie *alert* gebruikt om een foutmelding te genereren wanneer de invoer incorrect is.

Vervolgens wordt er doormiddel van 'return false' gezorgd dat het formulier niet naar de server wordt verzonden. Als controlemechanisme wordt er ook een server-side validatie geïmplementeerd.

De organisatie zal hiervoor PHP gebruiken. Voor elk invoerveld moet een vereiste worden gedefinieerd. Met behulp van loops en functies om de logica toe te passen, worden de invoervelden gevalideerd. Foutmeldingen zoals 'Telefoonnummer ongeldig' of 'Ongeldig e-mailformaat' worden teruggegeven bij incorrecte invoer. In het Proof of Concept wordt nog geen server-side validatie toegepast, behalve om te controleren of de gewenste gebruikersnaam al in de database bestaat. Als dit het geval is, ontvangt de gebruiker een foutmelding.

5.1.1 Serverside validatie met PHP

Om er zeker van te zijn dat de invoer goed gevalideerd kan worden op de server, zullen wij ook serverside validatie toevoegen. Dit zal gemaakt worden in de server taal PHP.

Deze validatie heeft een aantal redenen,

- Hierdoor worden invoer fouten door de gebruiker automatisch goed in de database verwerkt. Denk bijvoorbeeld aan een spatie verwijderen die per ongeluk is gezet.

- Ingevoerde SQL of HTML-code zal niet wordt uitgevoerd door het systeem. Op deze manier is het moelijker voor kwaadwillende om het systeem aan te vallen.

- Het netter maken van de invoer. Namen krijgen bijvoorbeeld automatisch een hoofdletter

- Hierdoor wordt de invoert gecontroleerd en eventueel aangepast zodat hij pas in de data base en niet tegen restricties aanloopt.

Door server validatie blijft de webshop en database veilig.

5.2 Acceptatietest registratie D

Zoals eerder is benoemd, wordt er één use-case uitgewerkt tot een Proof of Concept, namelijk het registreren. In **Figuur 8** zijn er acceptatiecriteria te zien voor het registreren. Tijdens het schrijven van de code wordt deze criteria aangehouden. Tijdens het ontwikkelen van de code worden er tests uitgevoerd om te controleren of de validatie werkt.

Per functie wordt getest of de eisen voldoen en wordt er gecontroleerd dat er een correcte en duidelijke foutmelding wordt gegenereerd op het moment dat er niet aan de eisen wordt voldaan.

Doormiddel van JavaScript worden de volgende validatiefuncties gebruikt:

- function isPositiefNummer(input)
- function valideerTelefoonnummer()
- function valideerPostCode()
- function valideerWachtwoord()
- function valideerEmail()
- function validateAndSubmit()

Nb: de gehele code is in de PoC te zien te vinden op proof_of_concept\webshop\public\js\script.js

In het HTML-document worden de volgende knoppen toegevoegd:

```
<div>
   <h5> Validatie testen</h5>
   <button onclick="isPositiefNummer(input)">Valideer Huisnummer</button>
   <button onclick="valideerTelefoonummer()">Valideer Telefoonnummer</button>
   <button onclick="valideerPostCode()">Valideer postcode </button>
   <button onclick="valideerWachtwoord()">Valideer wachtwoord </button>
   <button onclick="valideerEmail()">Valideer email</button>
   <button onclick="validateAndSubmit()">Valideer wachtwoord </button>
</div>
```

Nb: De validatieknoppen zijn niet opgenomen in het Proof of Concept omdat de tests succesvol waren. De validatie wordt geactiveerd door de 'Opslaan'-knop.

Na het testen en afronden van de code gaan Sippr en Devs2b bij elkaar komen om de acceptatiecriteria te toetsen, en wordt de beoordeling in de acceptatiecriteria toegevoegd en wordt het bewaard als documentatie.

Figuur 8. Acceptatiecriteria registratie

Acceptatiecriteria registratie				
Requirement ID	Test Case ID	Criteria	Beoordeling	Bijzonderheden
R-7.0	TC-010	Het invullen en opslaan van een registratieformulier resulteert in een account.	Behaald	
R-7.0	TC-011	Het ingevoerde huisnummer is een positief getal.	Behaald	
R-7.0	TC-012	Het ingevoerde postcode bevat 4 getallen en 2 letters.	Behaald	
R-7.0	TC-013	Het ingevoerde postcode bevat niet de volgende combinatie letters : 'SS', ' SA 'of 'SD'.	Behaald	
R-7.0	TC-014	Het ingevoerde postcode mag met kleine en grote letters.	Behaald	
R-7.0	TC-015	Het ingevoerde postcode mag niet met '0' beginnen.	Behaald	
R-7.0	TC-016	Het ingevoerde e-mailadres volgt het volgende patroon: tekst@teskt.tekst.	Behaald	
R-7.0	TC-017	Het ingevoerde telefoonnummer moet beginnen met een van de volgende reeksen: +31 , 0 of 0031.	Behaald	
R-7.0	TC-018	Het ingevoerde telefonnummer moet minimaal 10 tekens bevatten.	Behaald	
R-7.0	TC-019	Het ingevoerde telefoonnummer mag geen letters of speciale tekens bevatten.	Behaald	
R-7.0	TC-020	Het wachtwoord moet minimaal 8 tekens lang zijn .	Behaald	
R-7.0	TC-021	Het wachtwoord moet minimaal 1 hoofdletter bevatten.	Behaald	
R-7.0	TC-022	Het wachtwoord moet minimaal 1 cijfer bevatten.	Behaald	
R-7.0	TC-023	Het wachtwoordt mag speciale tekens bevatten.	Behaald	
R-7.0	TC-024	De gebruikersnaam is uniek.	Behaald	
R-7.0	TC-025	Het wachtwoord moet overeenkomen met de herhaling van het wachtwoord.	Behaald	
R-7.0	TC-026	Alle velden, behalve het veld 'toevoeging', zijn verplicht.	Behaald	
R-7.0	TC-027	Er wordt een foutmelding getoond als een veld incorrect is ingevoerd.	Behaald	
R-7.0	TC-028	Het formulier wordt niet verzonden naar de server op het moment dat er een veld niet correct of niet is ingevuld.	Behaald	
R-7.0	TC-029	Bij het correct invoeren van alle velden wordt de gebruiker doorverwezen naar het accountoverzicht	Behaald	
R-7.0	TC-030	In het accountoverzicht is het wachtwoord in een wachtwoordhash getoond.	Behaald	
R-7.0	TC-031	De ingevoerde gegevens zijn in de tabellen zichtbaar.	Behaald	
R-7.0	TC-032	De ingevoerde gegevens (behalve het wachtwoord & gebruikersnaam) worden in de database opgeslagen.	Behaald	

5.3 Gegevensverwerking en -normalisatie

In dit hoofdstuk wordt een samenvatting gegeven van de gegevensverwerking en de normalisatie van de database.

5.3.1 Gegevensverwerking

In hoofdstuk 4 van dit technische ontwerp wordt beschreven hoe de gegevens worden ingevoerd, verwijderd of bijgewerkt in de database.

Zoals in de requirementslist (Boogaard A., Diallo A., Van Benthem D., Nieskens I., & Pearson J, 2023) staat vermeld, zal de organisatie gebruik maken van SQLite als databasemanagementsysteem. Binnen dit systeem wordt SQL-editor gebruikt om handmatig queries te schrijven, waardoor gegevens kunnen worden opgehaald(read), toegevoegd (create), bijgewerkt (update) en verwijderd (delete). De beveiligingsmaatregelen voor deze gegevens zullen worden toegelicht in hoofdstuk 6 van dit technische ontwerp.

5.3.2 Normalisatie

Normalisatie is een belangrijk proces in het definiëren van de databasestructuur. Het doel is om redundantie te verminderen; redundantie is het proces waarbij dezelfde gegevens op meerdere plaatsen in een systeem worden opgeslagen. Dit kan problematische zijn in een informatiesysteem, omdat de kans op inconsistentie hierdoor wordt vergroot en het ingewikkelder wordt om gegevens bij te werken. Door normalisatie wordt dubbele opslag van gegevens voorkomen en de relaties tussen entiteiten worden geherstructureerd, waardoor een gestructureerde en efficiënte database wordt gecreëerd.
In hoofdstuk 3 van het functioneel ontwerp (Boogaard et al., 2023) zijn de entiteiten, attributen en primaire sleutels gedefinieerd die in de database zullen worden opgeslagen.

Voor deze database voldoen wij aan de 3e normaalvorm (3NF). Dit betekent dat er geen transitieve afhankelijkheid is. Transitieve afhankelijkheid betekend dat er geen kolom afhankelijk is van een kolom die op zijn beurt afhankelijk is van de primaire sleutel. Met

andere woorden, er zijn geen attributen die afhankelijk zijn van andere niet-sleutelattributen. (appMaster, 2022).

6 Toegankelijkheid

Om er voor te zorgen dat de webshop toegankelijk is voor zo veel mogelijk mensen, maken wij een aantal functies die dit in hand spelen.

6.1 Taal switch

Zoals te lezen is in het plan van aanpak, (Boogaard A., Diallo A., Van Benthem D., Nieskens I., & Pearson J, 2023) Zal het mogelijk zijn om de taal te wisselen tussen Nederlands en Engels.

Dit gaan wij realiseren door twee losse taal documenten te maken die worden ingeladen aan de hand van de taal die de gebruiker kiest. Door te werken losse taal documenten is het relatief makkelijk om later ander talen toe te voegen.

6.2 Zoek functie

Op de website zal het mogelijk zijn om te zoeken naar producten en gebruikers. Het zoeken naar gebruikers zal enkel mogelijk zijn voor de rollen, medewerker en beheerder.

Het zoeken naar producten maakt het makkelijk voor de klant om direct het product te zien wat hij wil. Dit doen wij op twee manieren. Op de winkel pagina is een filter aanwezig, deze filtert op voor gekozen categorieën. Daarnaast is er een zoek pagina, hierin word er op keyboards gezocht in de database.

Op het moment dat woorden die de gebruiker zoekt, niet gevonden kunnen worden. Zullen deze worden opgeslagen zodat de beheerder deze zoek woorden later kan terug vinden en eventueel kan toevoegen aan de keywords.

7 Unittesten

Unittesten in PHP zijn van cruciaal belang voor de kwaliteitsborging van de code. Ze helpen bij het identificeren van bugs op het kleinste, meest geïsoleerde niveau van de applicatie - de individuele functies of methoden. Door functies afzonderlijk te testen, kunnen ontwikkelaars verifiëren of functies correct werken onder verschillende omstandigheden en met verschillende invoer. Dit maakt het gemakkelijker om problemen te lokaliseren en te repareren.

Doormiddel van unittesten kunnen ontwikkelaars een proactieve benadering hanteren om potentiële bugs en fouten te identificeren nog voordat de code naar productieomgevingen wordt uitgerold. Bovendien zorgen unittesten voor een soort documentatie die nieuwe teamleden kunnen gebruiken om te begrijpen wat elke functie zou moeten doen.

Unittests zijn handig omdat ze helpen bij het opsporen van fouten die optreden wanneer nieuwe code wordt toegevoegd of bestaande code wordt gewijzigd. Zo kunnen ontwikkelaars met meer zekerheid aanpassingen maken zonder dat dit de al bestaande werking van de code in gevaar brengt.

Doormiddel van unittesten worden best practices in software-engineering versterkt, zoals modulaire en herbruikbare code, goede documentatie en het naleven van coderingsstandaarden.

In essentie zijn unittesten een onschatbaar instrument voor het creëren en onderhouden van robuuste en hoogwaardige PHP-applicaties. Door code op verschillende niveaus te testen, kunnen ontwikkelaars de betrouwbaarheid en bruikbaarheid verbeteren, wat helpt om te voldoen aan de groeiende eisen van moderne softwareontwikkeling.

7.1 PHPUnit

PHPUnit is een specifiek unittest-framework voor PHP. Het biedt een gestructureerde manier om unittesten te schrijven en uit te voeren. Met PHPUnit kunnen ontwikkelaars testcases opzetten, afhankelijkheden beheren, uitzonderingen testen en de resultaten van hun tests op een overzichtelijke manier presenteren.

Het gebruik van PHPUnit kan helpen bij het verbeteren van de codekwaliteit en het verhogen van het vertrouwen in de stabiliteit van de applicatie. Het stelt teams ook in staat om nieuwe functies en verbeteringen sneller en met minder risico te implementeren.

7.1.1 PHPUnit installeren en gebruiken

Na installatie, hetzij via Composer of door het downloaden van een PHAR-bestand, stelt PHPUnit een ontwikkelaar in staat om unittesten te schrijven en uit te voeren.

In PHPUnit worden tests geschreven als methoden binnen een testklasse. Elke testklasse moet een subklasse zijn van `PHPUnit\Framework\TestCase`. Binnen elke testmethode worden beweringen (assertions) gebruikt om te controleren of de code zich gedraagt zoals verwacht.

Wanneer de tests worden uitgevoerd, biedt PHPUnit gedetailleerde rapporten over welke tests zijn geslaagd en welke zijn mislukt. Het kan ook code coverage rapporten genereren, wat nuttig is om te zien welke delen van de code worden getest. Hiervoor wordt Xdebug gebruikt.

7.1.1.1 Assertion-regels

De unittests worden geschreven met een methode waarbij de verwachte uitkomst vergeleken wordt met de uitkomst die door de test uitgevoerd wordt. Daarvoor kunnen onder andere de vergelijkingen gemaakt worden, ook wel Assertion-rules genoemd, die in Tabel 2 geplaatst zijn (Haperen, 2018).

Tabel 2 Overzicht veelgebruikte assertions

assertTrue($x)	Fail if $x is false
assertFalse($x)	Fail if $x is true
assertNull($x)	Fail if $x is set
assertNotNull($x)	Fail if $x not set
assertIsA($x, $t)	Fail if $x is not the class or type $t
assertNotA($x, $t)	Fail if $x is of the class or type $t
assertEquals($x, $y)	**Fail if $x == $y is false**
assertNotEquals($x, $y)	Fail if $x == $y is true
assertWithinMargin($x, $y, $m)	Fail if abs($x - $y) < $m is false
assertOutsideMargin($x, $y, $m)	Fail if abs($x - $y) < $m is true
assertIdentical($x, $y)	Fail if $x == $y is false or a type mismatch
assertNotIdentical($x, $y)	Fail if $x == $y is true and types match
assertReference($x, $y)	Fail unless $x and $y are the same variable
assertClone($x, $y)	Fail unless $x and $y are identical copies
assertPattern($p, $x)	Fail unless the regex $p matches $x
assertNoPattern($p, $x)	Fail if the regex $p matches $x

Voor een compleet overzicht van de meer dan 80 assertions:
https://docs.phpunit.de/en/11.1/assertions.html#

7.1.2 Unittest voorbeeld

Een voorbeeld van een unittest voor een PHP-functie die controleert of een gegeven string een geldig e-mailadres is:

Eerst wordt de e-mailvalideringsfunctie gemaakt:

```
<?php
function isValidEmail($email) {
    return filter_var($email, FILTER_VALIDATE_EMAIL) !== false;
}
?>
```

En een unittest met PHPUnit voor deze functie:

```
<?php
use PHPUnit\Framework\TestCase;

class EmailTest extends TestCase {
    public function testIsValidEmail() {
        // Test voor een geldig e-mailadres
        $this->assertTrue(isValidEmail('test@example.com'));

        // Test voor een ongeldig e-mailadres
        $this->assertFalse(isValidEmail('test@@example'));
    }
}
?>
```

8 Beveiliging en autorisatie

Beveiliging en autorisatie zijn van belang om de beschikbaarheid, integriteit en vertrouwelijkheid te waarborgen. Daarnaast zijn ze nodig om informatie van klanten te beschermen tegen potentiële bedreigingen. Door verschillende gebruikersrollen te gebruiken binnen het systeem wordt er gewaarborgd dat alleen personen met de juiste bevoegdheid toegang hebben tot bepaalde delen van het systeem. Daarnaast wordt er door gebruik te maken van versleuteltechnieken gezorgd dat informatie op de juiste manier versleuteld wordt.

8.1 Toegangscontrole

De verschillende gebruikersrollen binnen het systeem zijn beheerder, verkoopmedewerker, klantenservice en klant. In het functioneel ontwerprapport (Boogaard et al., 2023) hoofdstuk 6.2 in tabel 23: Menustructuur 'Sippr.' webshop staat aangegeven welke rol toegang heeft tot welke delen van het menu, en welke rechten hierbij horen. Door autorisatie op verschillende niveaus toe te wijzen, wordt ervoor gezorgd dat informatie alleen door de juiste gebruiker wordt ingezien en gewijzigd.

Om deze gebruikersrollen veilig te houden, worden verschillende mechanismen voor authenticatie en autorisatie gebruikt. De identiteit van de gebruiker wordt geverifieerd door gebruik te maken van alfanumerieke wachtwoorden van een lengte van 8 tekens of meer.

8.2 Gegevensversleuteling

Om ervoor te zorgen dat data op de juiste manier versleuteld is, wordt er gebruik gemaakt van SSL/TLS-encryptie. Zoals beschreven in paragraaf 3.1 van dit document, zorgen SSL-certificaten ervoor dat de gegevens die worden overgedragen tussen de gebruiker en de server versleuteld zijn. Let's Encrypt zal worden gebruikt om deze certificaten te verkrijgen. De gebruiker zal dit zien doordat de URL van de website begint met "https://". Dit wordt ook aangegeven met een hangslot in de zoekbalk.

Voor het versleutelen van de database wordt gebruik gemaakt van verschillende ingebouwde functies in SQLite. Hierbij worden verschillende gebruikersrollen met bijbehorende rechten gedefinieerd. Daarnaast heeft SQLite een ingebouwde interface voor encryptie waarvan gebruik zal worden gemaakt.

Wachtwoorden van gebruikers worden handmatig gegenereerd, maar moeten voldoen aan eerdergenoemde eisen. Daarnaast moet een gebruiker met een rol anders dan klant jaarlijks zijn of haar wachtwoord vernieuwen.

9 Implementatieplan

9.1 Fasering van implementatie

Zoals aangegeven in het functioneel ontwerprapport (Boogaard et al., 2023) zal de implementatie in één stap plaatsvinden, waarbij de webshop als geheel wordt opgeleverd.

9.2 Training en documentatie

Het ‘Sippr.’ team zal worden getraind in het gebruik van de webshop. Het 'Sippr.' team zal worden getraind in het gebruik van de webshop. Voor elke rol, waaronder Beheerder, verkoopmedewerker en klantenservice, zal specifieke training worden gegeven (Boogaard et al., 2023).

Tijdens de trainingen zullen er ook digitale documenten worden gedeeld, per rol met uitleg over de webshop. Hier staan alle handelingen nogmaals beknopt in beschreven. De documenten zijn bedoeld als handleiding, het personeel kan deze gebruiken op het moment dat bepaalde functies niet meer helder zijn, of als er nieuw personeel wordt ingewerkt. Mochten er later updates verschijnen die functies van de webshop zullen aanpassen. Zal het document worden aangepast.

9.3 Risicobeoordeling en -beheer

Zoals beschreven in het vooronderzoek rapport (Boogaard et al., 2023) wordt er rekening gehouden met drie grote risico’s:

Technische storingen (server uitval enz....)	**Preventie**: Back-upsystemen
Beveiligingsinbreuken (cyberaanval)	**Preventie**: Implementeer geavanceerde beveiligingsprotocollen, waaronder SSL-certificaten, MFA en de database zal versleuteld worden opgeslagen.
Fouten in de software (bugs)	**Preventie**: Regelmatig uitvoeren van tests en audits
	Implementeer kwaliteitscontrole procedures tijdens de softwareontwikkeling

Devs2B zal ervoor zorgen dat er zo veel mogelijk wordt gedaan om deze risico's te beperken. Uiteindelijk is het echter aan 'Sippr.' om te bepalen of Devs2B doorgaat met het beheren en updaten van de webshop.

Bibliografie

appMaster. (2022, okt 30). *wat-is-datanormalisatie*. https://appmaster.io/: https://appmaster.io/nl/blog/wat-is-datanormalisatie

Aweda, Z. I. (2022, 9 maart). *How to Test PHP Code With PHPUnit*. Retrieved 15 december 2023, from freeCodeCamp: https://www.freecodecamp.org/news/test-php-code-with-phpunit/

Boogaard A., Diallo A., Van Benthem D., Nieskens I., & Pearson J. (2023). *Plan van aanpak.* Breda: Devs2b. https://sippr.nl/pdf/Plan_van_aanpak_Sippr_220FDD-DO1_3_DEF.pdf

Boogaard, A., Diallo, A., van Benthem, D., Nieskens, I., & Pearson, J. (2023). *Functioneel ontwerprapport.* Breda: Devs2b.

Boogaard, A., van Benthem, D., Diallo, A., Nieskens, I., & Pearson, J. (2023). *Vooronderzoek rapport.* Devs2B.

Brinkman-Davis, S. (2012, 4 december). *Essence of MVC* . Retrieved 5 december 2023, from Essence and Artifact : https://www.essenceandartifact.com/2012/12/the-essence-of-mvc.html

Cloudflare. (z.d.). *What is SSL?* Retrieved 14 december 2023, from Cloudflare: https://www.cloudflare.com/learning/ssl/what-is-ssl/

Haperen, M. v. (2018, 31 mei). *Unit testen: Presentatie [Powerpoint-pressentatie].* Retrieved 2 mei 2024, from Brightspace Avans: https://brightspace.avans.nl/d2l/le/lessons/142614/topics/1225426

Osman, J. (2023, 23 maart). *CRUD Operations - Wat is CRUD?* appmaster.io: https://appmaster.io/nl/blog/crud-operations-wat-is-crud

ostezer, & Drake, M. (2022, 9 maart). *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. Retrieved 13 december 2023, from DigitalOcean: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

Patric. (2023, 23 februari). *A Beginner's Guide to PHPUnit*. Retrieved 14 december 2023, from Medium: https://pguso.medium.com/a-beginners-guide-to-phpunit-writing-and-running-unit-tests-in-php-d0b23b96749f

Rabobank. (z.d.). *Rabo OnlineKassa*. Rabobank: https://www.rabobank.nl/bedrijven/betalen/klanten-laten-betalen/rabo-smart-pay/rabo-onlinekassa

The PHP Group. (z.d.). *Intro PDO*. Retrieved 13 december 2023, from PHP: Introduction manual: https://www.php.net/manual/en/intro.pdo.php

w3schools. (z.d.). *HTML*. w3schools: https://www.w3schools.com/html/

w3schools. (z.d.). *Javascript*. w3schools: https://www.w3schools.com/js/

Warmer, J., & Kleppe, A. (2011). *Praktisch UML.* Pearson Education Benelux.

Website Rating. (2023, 22 mei). *What is apache server?* Retrieved 13 december 2023, from Website Rating: https://www.websiterating.com/nl/web-hosting/glossary/what-is-apache-server/