SOFTWARE ARCHITECTUUR DOCUMENT

Document sur l'architecture logicielle

Jordy de Vries
S1151166

Toestemming gebruik
Ik geef de opleiding informatica toestemming om dit verslag in geanonimiseerde vorm te gebruiken voor onderwijsdoeleinden: *Ja/~~Nee~~* (doorhalen wat niet van toepassing is).

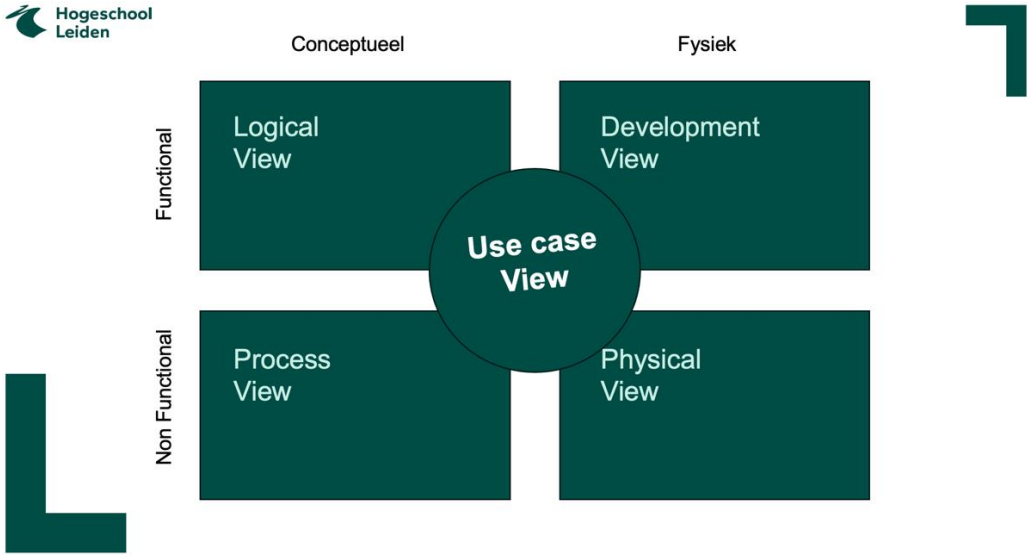
Inhoudsopgave

1. Inleiding

1.1 Doel van dit document

Het Software Architectuur Document (SAD) bevat een uitgebreide architecturale kijk op het systeem WebshopBob ontwikkeld door dhr. Bob. Het beschrijft een aantal verschillende architecturale views van het systeem om zo verschillende aspecten van het systeem te belichten.

Dit document beschrijft de verschillende views op de software architectuur volgens het 4+1 view model. Het 4+1 view model stelt de verschillende belanghebbenden in staat vanuit hun eigen perspectief de invloed van de gekozen architectuur te bepalen.



Het doel van dit document is om het duidelijker te maken voor de belanghebbenden en eventueel de toekomstige developers hoe de feature, in dit geval productvarianten, geprogrammeerd is en later ook geïmplementeerd is in de webshop van dhr. Bob.

Dit document biedt inzicht in alle technische aspecten van de feature. Zo gaat het bijvoorbeeld over alle endpoints die zijn toegevoegd en waar in de feature gebruik van is gemaakt en over de eisen van de endpoints en de data die de endpoints terugsturen zodra een request is ontvangen in de backend/API. Per hoofdstuk wordt er in dit document ingegaan op bepaalde technische aspecten van functionele tot non functionele requirements en van conceptueel tot fysiek.

De feature zal gebruikers de optie bieden om een keuze te kunnen maken tussen verschillende varianten van hetzelfde product. Hiervoor zijn enkele wijzingen gemaakt en toevoegingen gedaan aan de huidige scripts van de webshop om de feature werkend te kunnen krijgen.

In de logical view krijgt u met behulp van klassendiagrammen inzicht in hoe de klassen zijn toegevoegd aan de code en wat er veranderd is aan andere klassen. In de procesview is te zien wat de code doet en data die daarbij wordt verstuurd. De development view geeft vooral developers inzicht in hoe een functie te werk gaat en andere functie/endpoints aan roept. En tot slot geeft de physicalview inzicht in de configuraties van het systeem zoals server details en eventuele domein configuraties

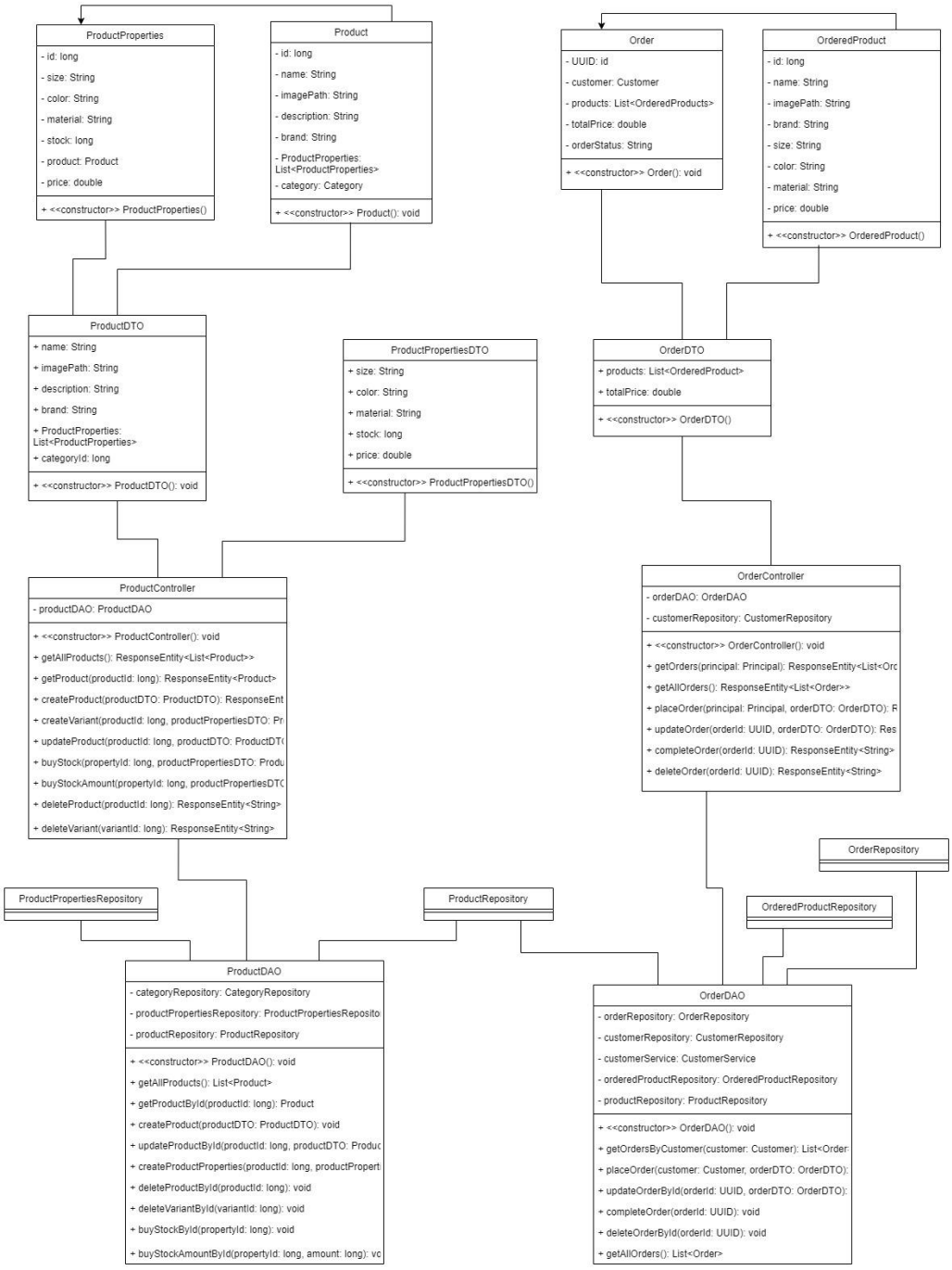
1.2 Referenties

Titel	**Versie**	**Auteur**
INSOFAD_s1151166_requirements	V1	Jordy de Vries
INSOFAD_s1151166_use_case_model	V1	Jordy de Vries
INSOFAD_s1151166_test_plan	V1	Jordy de Vries

2. Logical View

1. Backend diagram

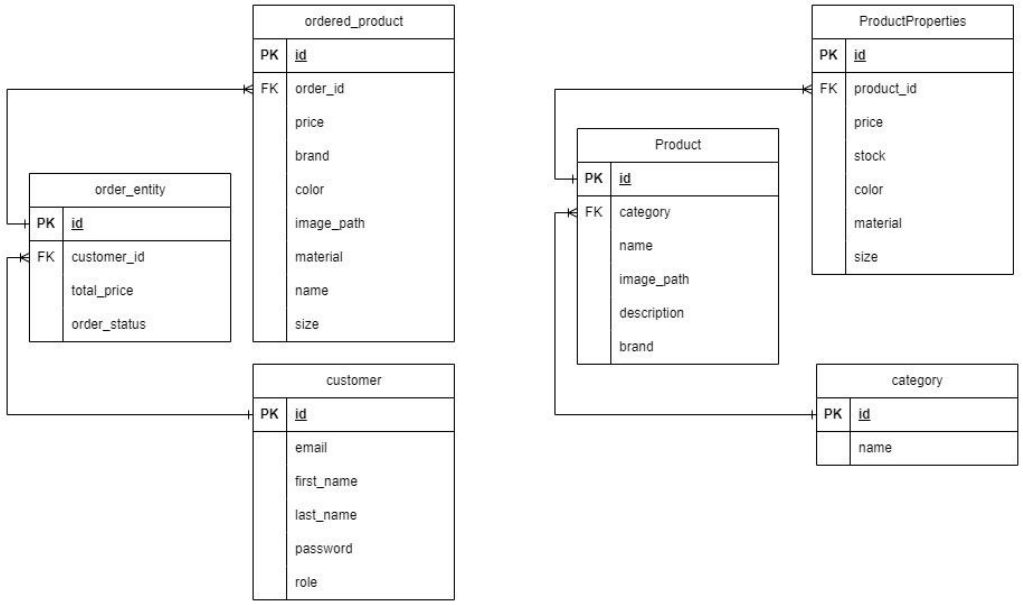
Ik heb voor het maken van de backend een aantal klassen toegevoegd om de feature werkend te kunnen krijgen. Deze klassen hebben ook weer eigen functies en variabelen die met de feature te maken hebben. In het onderstaande schema is precies te zien hoe de klassen naadloos samenkomen en de functionaliteit van de gehele API uitbreiden



Een van de Use Cases was dat een klant kon kiezen uit verschillende productvarianten van hetzelfde product. Dit heb ik op de volgende wijze weten te implementeren. De product klasse die boven in het diagram te zien is was al een bestaande klasse. Deze klassen heeft een attribuut extra gekregen namelijk List<ProductProperties>. Deze list is een lijst met mogelijke productvarianten. Ook heb ik in de product klasse de prijs attribuut weggehaald en naar de productproperties klasse gebracht. Dit omdat voor elke productvariant de prijs moet kunnen verschillen. Orders zijn ook aangepast. De orders bevatten nu een lijst van orderedproducts, wat overigens ook een nieuwe klasse is. Deze klasse bevatten alle details van een product. Het is eigenlijk een soort clone van de product klassen maar dan iets minder uitgebreid. Eerst was een relatie tussen product en order gelegd, maar deze relatie was voor de update niet handig. Want zodra een product verwijderd wordt kan een klant het product niet meer terugvinden in zijn orders. Ik heb geen Controller en DAO gemaakt voor productproperties, omdat de product en productproperties klassen heel erg dicht bij elkaar liggen waardoor ik eigenlijk alle functionaliteit kon toevoegen in de productcontroller en productDAO. Bovendien moest de meeste functionaliteit voor productproperties via de product klasse.

2. Database diagram

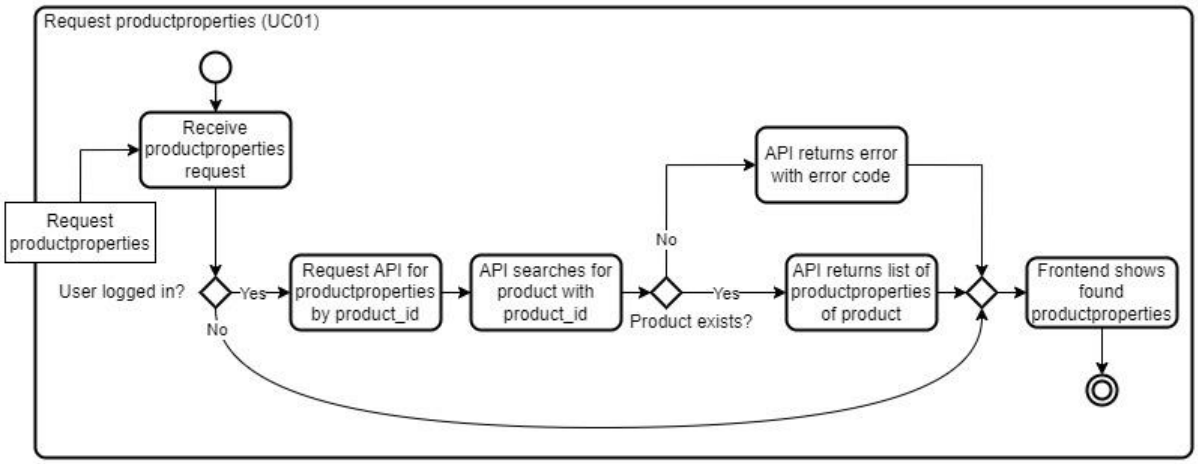
Hieronder is een klassen diagram van de database weergegeven. Het diagram bestaat eigenlijk uit twee kleine diagrammen. Het eerste diagram voor de orders en het tweede diagram voor alle producten en producteigenschappen. Hierin is ook goed te zien hoe er een scheiding zit tussen de orders en de producten. Mocht er in dit geval een product worden aangepast of verwijderd, dan wordt het product niet uit de orders verwijderd en kan een klant gewoon zijn orders blijven inzien. Elke product entiteit kan meerdere productproperties bevatten, maar elke productproperties entiteit is maar gelinkt aan één product entiteit. Dit is met de order, ordered_product en customer entiteiten hetzelfde. Order_entity kan meerdere producten hebben maar is gelinkt aan slechts één customer.



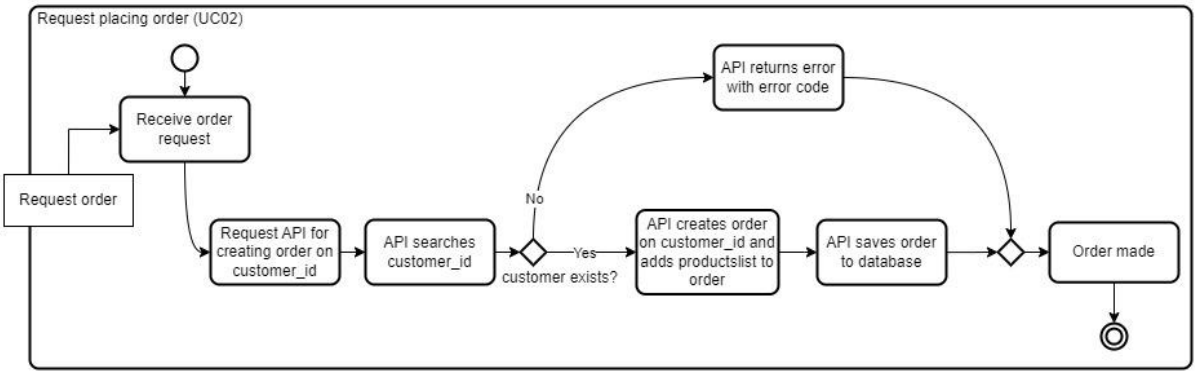
3. Process View

Hieronder zijn van vijf Use cases een globale werking te zien. Het laat zien hoe de frontend van een functie een request maakt, de backend met de request om gaat en wat de backend terug stuurt en wat de frontend en eventueel een database die betrokken is doet. Voor elke Use Case is in diagram te zien om welke Use Case het gaat.

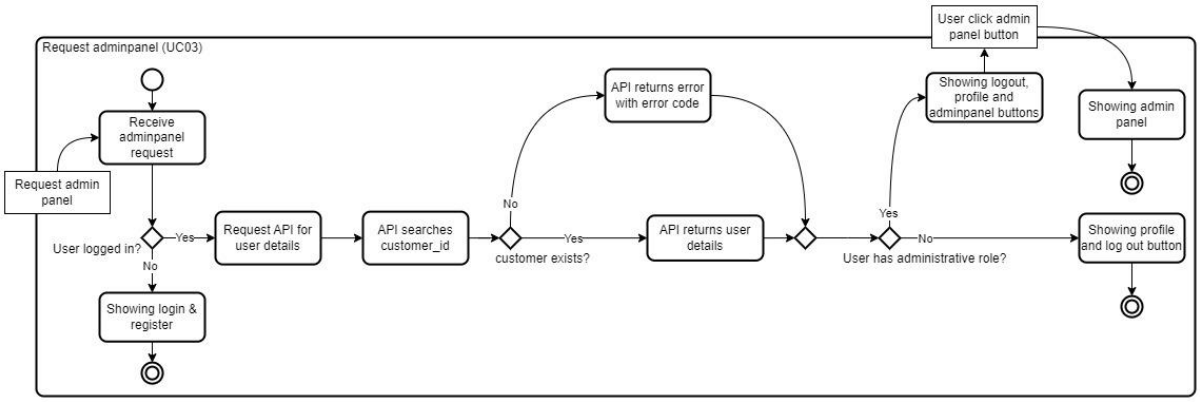
Als eerst is hieronder de werking van UC01 weergegeven. Dit is de Use Case die gaat over het tonen van producteigenschappen zodra er op de frontend op een product wordt geklikt.



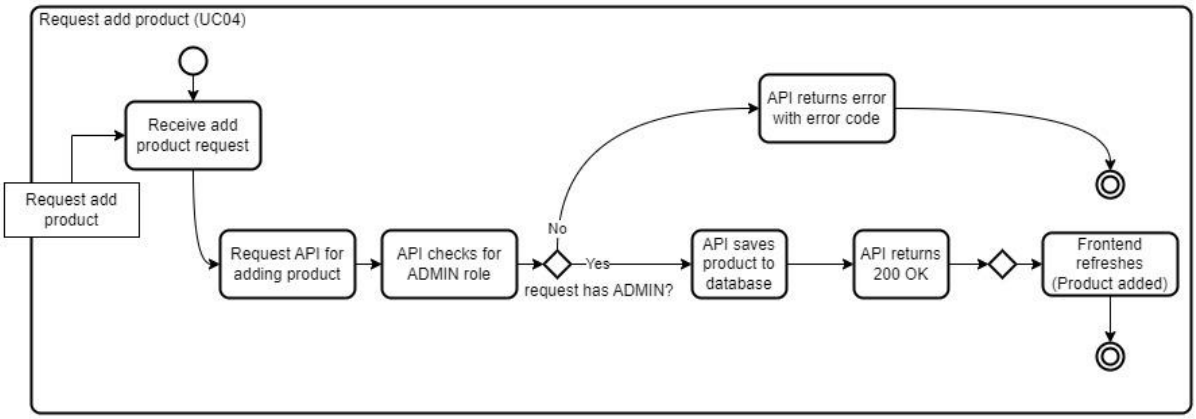
Als tweede is de Use Case UC02 gegeven. Dit is een Use Case die gaat over het plaatsen van een order.



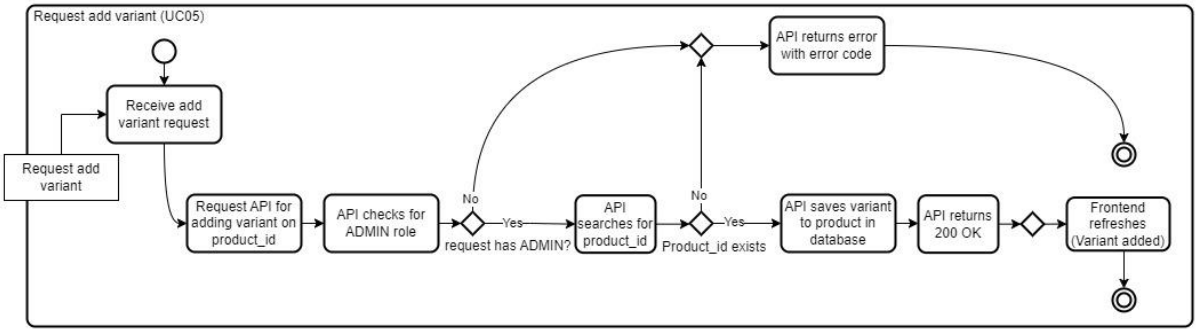
Dan als derde Is de Use Case UC03 uitgewerkt. Dit is de Use Case die gaat over het administrator paneel. Dit is overigens ook een wat complexere Use Case die namelijk ook met authenticatie werkt op verschillende niveaus van frontend tot backend.



Als vierde Use Case is UC04 uitgewerkt. Dit is een Use Case die gaat over het toevoegen van een product.



En tot slot, de vijfde Use Case. Deze Use Case, UC05, gaat over het toevoegen van een productvariant. Hiervoor wordt ook authenticatie vereist en er moet al een product voor de productvariant zijn aangemaakt.

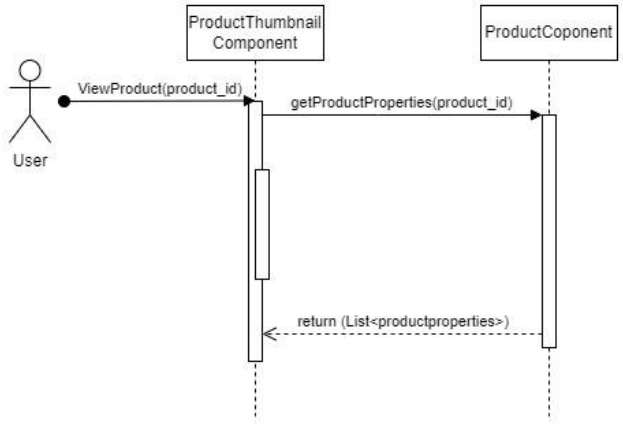


4. Development View

In dit hoofdstuk zijn de fysieke en functionele invullingen van het softwaresysteem beschreven. Dit hoofd stuk is vooral bedoeld voor eventuele developers die nog meer uitbreiding gaan bieden aan dit project. In dit hoofdstuk wordt met sequentie diagrammen duidelijk gemaakt hoe het systeem te werk gaat met verschillende functies en welke functies worden aangeroepen. De sequentie diagrammen zijn per Use Case uitgewerkt en bevatten elke een korte beschrijving om de volgordelijkheid duidelijk te maken.

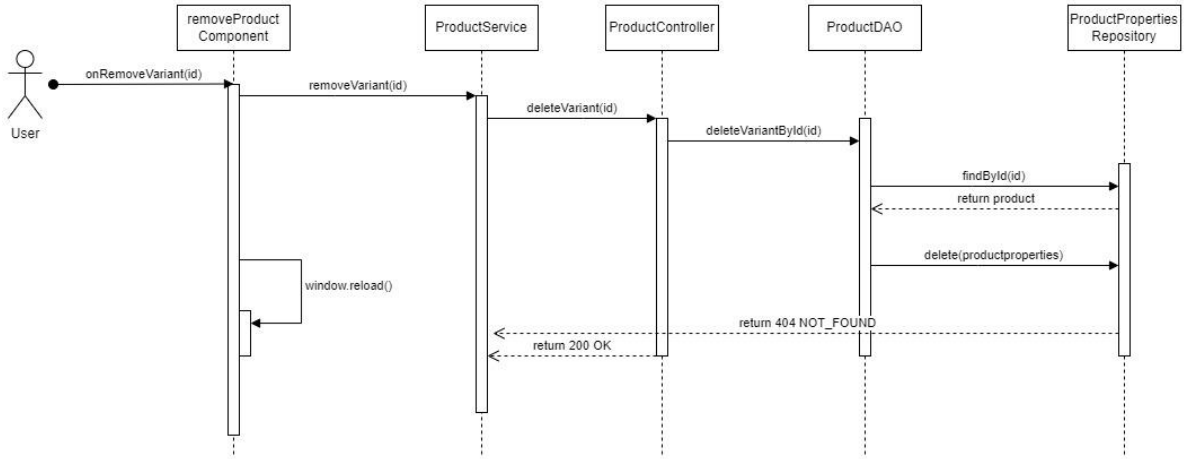
1. USC01

Dit sequentie diagram laat zien hoe de productvarianten op de website getoond worden. Het gehele proces bevindt zich alleen in de frontend. Er komt dus geen API van te pas. Het proces begint en eindigt ook in de ProductThumbnailComponent.



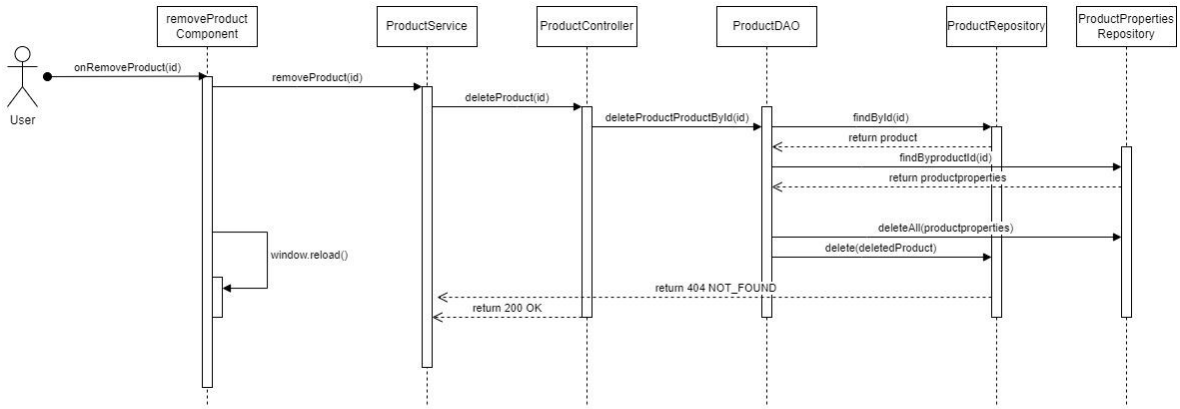
2. USC07

Dit sequentie diagram beschrijft hoe de request van het verwijderen van een productvariant in werking gaat. Het begint bij de RemoveProductComponent, er wordt een request gemaakt in de service naar de backend en de backend kijkt of het product bestaat en zo ja dan verwijderd de backend het product. Mocht het product niet bestaan, dan wordt er een error gegooid.



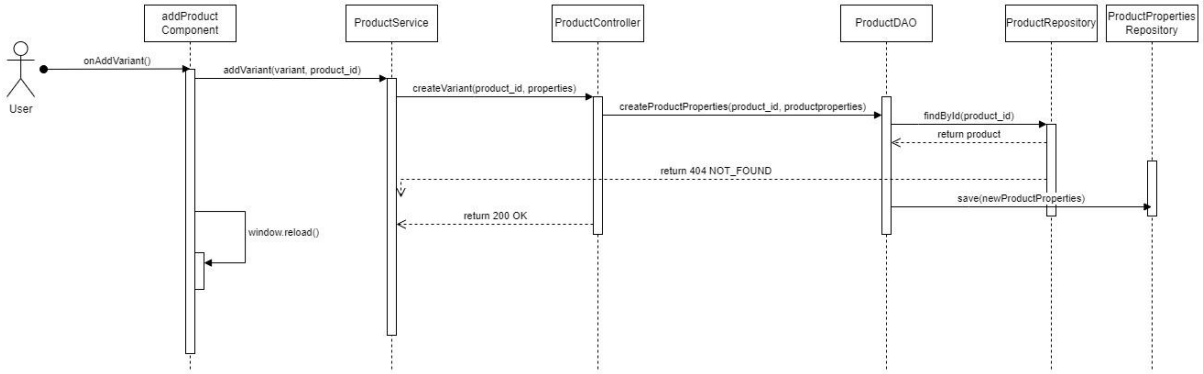
3. USC06

Dit sequentie diagram beschrijft hoe de request van het verwijderen van een product in werking gaat. Het begint bij de RemoveProductComponent, er wordt een request gemaakt in de service naar de backend om een product te verwijderen. De backend kijkt of het product bestaat. Zo ja, dan verwijderd de backend het product. En slaat de backend de verandering op in de database Bestaat het product echter niet, dan wordt er een error gegooid.



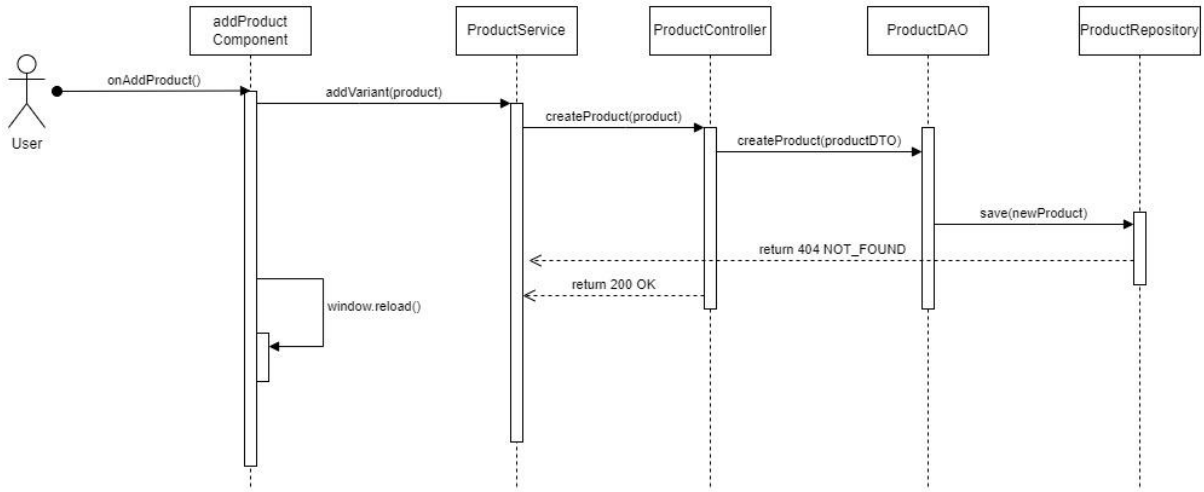
4. USC05

In dit diagram wordt komt tot stand hoe een productvariant kan worden toegevoegd aan de database. Het begint in de addproductcomponent. Vervolgens zal er een request worden gemaakt in de productservice naar de backend. De backend ontvangt de request om een productvariant toe te voegen. En de backend ontvangt hierbij ook gelijk de data voor het product via de request. De productRepository zoekt voor het product met het ID waaraan de variant moet worden toegevoegd en de productPropertiesRepository maakt een nieuwe variant aan met een uniek ID en slaat deze op in de database. Tot slot wordt er een 200 OK status code geretourneerd



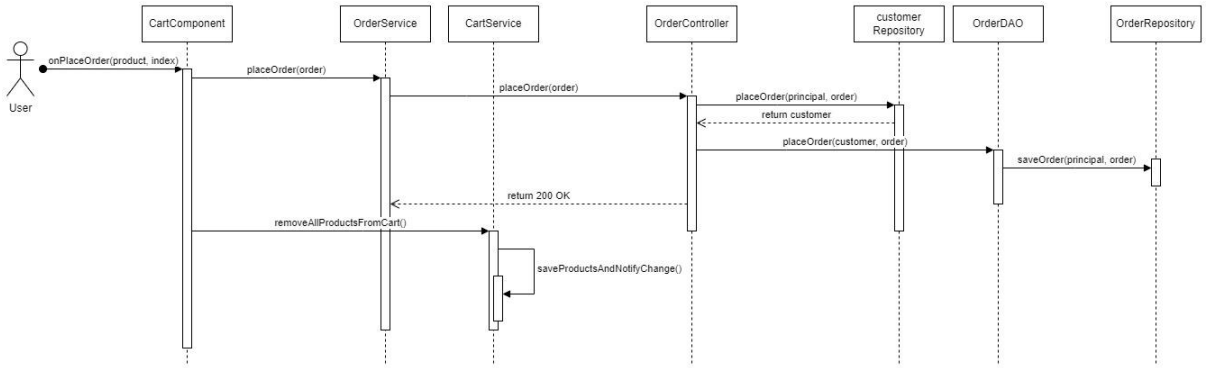
5. USC04

In dit diagram wordt duidelijk gemaakt hoe de producten aan de database worden toegevoegd. Het begint in de addProductComponent. In de productservice wordt een request aangemaakt. De productcontroller ontvangt de request met de data voor het product en stuurt de aanvraag door naar de DAO. Daar word een product samengesteld en in de repository wordt het product opgeslagen.



6. USC04

Tot slot het laatste sequentiediagram. Dit sequentiediagram beschrijft het order proces. Zodra de gebruiker op bestellen klikt in cartcomponent wordt er in orderservice een request gemaakt naar de ordercontroller. De order controller zoekt vervolgens naar de gebruiker in de database. En slaat de order met de producten op in de database op het ID van de gebruiker. Vervolgens wordt in de cartcomponent ook de cartservice aangeroepen. En in de cartservice wordt een functie aangeroepen die het winkelwagentje leegt en de cartcomponent refreshed.

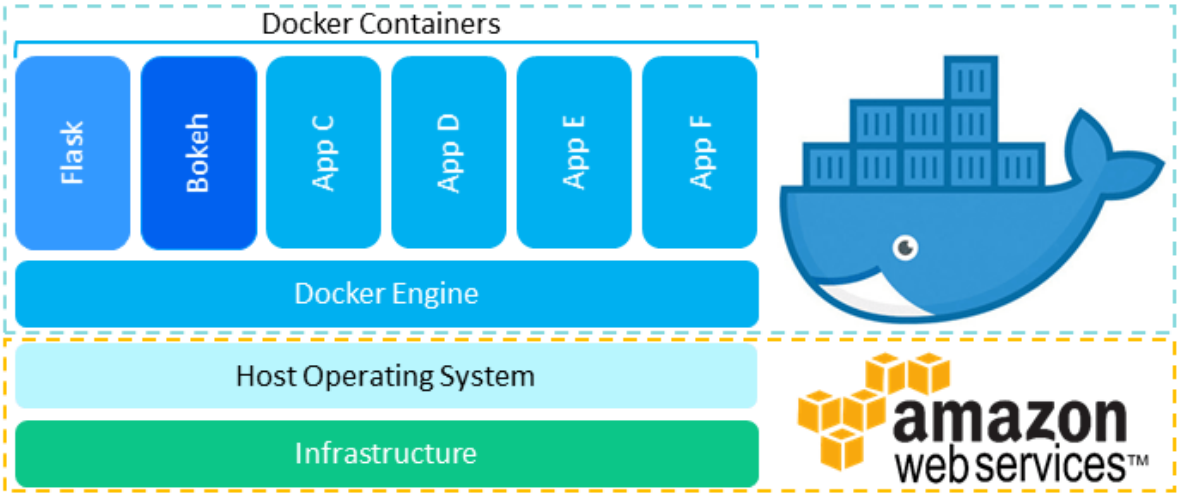


5. Physical View

1. Docker

De gehele webshop inclusief database is gedockerized. Dit houdt in dat elk proces (frontend, backend, database…) in een container draait en dus niet direct op de server/werk ruimte. Het gebruik van Docker verhoogt de efficiëntie en verlaagd de kans de kans op falen van een van de systemen door een bepaald environment na te maken waar de software ongetwijfeld in kan draaien. Dit helpt gelijk ook tegen het welbekende '*it works on my machine*' probleem doordat de software niet direct in Windows of MacOS draait maar een soort virtuele machine (VM). Voor het draaien van de software kon ook een daadwerkelijke VM gebruikt worden, echter is daarvan het nadeel dat de VM eerst moet worden ingesteld wat overigens enige expertise vereist op het gebied van hardware. Ook zal de VM een deel van het processor vermogen en het ramgeheugen pakken wat constant voor de VM beschikbaar zal blijven.

Verder is er ook gebruik gemaakt van docker-compose. Docker-compose laat alle processen in een pod draaien. Het zorgt er ook voor dat je met één druk op de knop alle processen in de juiste volgorde kan opstarten en afsluiten. Het hele plaatje is makkelijk te visualiseren met deze afbeelding



Voor de website is gebruik gemaakt van vier Docker containers een voor de frontend, een voor de backend, een voor de database en als laatst nog een voor het PGadmin systeem. Verder is er ook nog een Docker Compose bestand waar alle Docker containers met één click in Docker desktop mee kunnen worden opgestart.

2. HTTP/HTTPS

Voor de datatransfer tussen de frontend en de backend worden HTTP requests verstuurd. Dit zijn requests die data op verschillende manieren van een frontend naar een backend kunnen brengen. Ik heb voor de backend een aantal HTTP endpoints toegevoegd. In onderstaand tabel kan u precies zien wel requests dat zijn en wat ze doen.

Basis URL path	

http:///localhost:8080/api/				
URL	Method	Data terug te verwachten	Status	Data Type
products/properties/{productId}	POST	Product was succesfully created.	200	String
products/properties/{productId}	PUT	Product was bought	200	String
products/properties/{productId}/{amount}	PUT	Product was bought	200	String
products/variant/{variantId}	DELETE	Product was succesfully deleted	200	String
account	GET	Customer JSON-data	200	JSON

3. PORTS

In de Docker Compose wordt een soort LAN aangemaakt (Local Area Network) waar alle services (containers) met elkaar data over kunnen uitwisselen. Het is mogelijk om een port in de container of de Docker Compose open te zetten zodat je er ook vanaf je eigen PC of met een ander netwerk mee kan praten. De individuele container hebben ook allemaal hun eigen configuraties en porten. Deze zijn te zien in de tabel hieronder. Er is gelijk ook te zien met welke porten elke port verbonden is.

Proces	PORT	Listen Port	Talks to
<YOUR MACHINE>	-	-	4200
Frontend	4200:4200	-	8080
Backend	8080:8080	-	5432
PGadmin	8000:80	80	5432
Postgres	5432:5432	-	-

4. Database

Aan de database is niets veranderd. Voor de database wordt nog steeds hetzelfde PGadmin systeem gebruikt en de database zelf is ook nog steeds een PostgreSQL database. De database heeft wel nieuwe entiteiten gekregen om de feature werkend te krijgen. De database en het PGadmin systeem draaien allebei ook in een Docker container.

5. Links

De GitHub links zijn hieronder te vinden. Ik heb voor het grootste deel van dit project gebruik gemaakt van GitHub omdat ik graag wel gebruik wilde maken van twee verschillende apparaten (PC en laptop). Ik heb op de laatste dag nog wel alle code ook naar de GitLab gepushed.

Aanbevolen (GitHub)
Frontend: JordyDevrix/INSOFAD_frontend: Software Advanced frontend (github.com)
Backend: JordyDevrix/INSOFAD_backend: Software Advanced backend (github.com)
Docker & Docs: JordyDevrix/INSOFAD_docker_and_docs (github.com)

GitLab
Frontend: Jordy de Vries / INSOFAD_frontend · GitLab (inf-hsleiden.nl)
Backend: Jordy de Vries / INSOFAD_backend · GitLab (inf-hsleiden.nl)
Docker & Docs: Jordy de Vries / docker_and_docs · GitLab (inf-hsleiden.nl)