Technisch ontwerp

Technisch ontwerp

Project Devices

Team Selficient - Virtuele realisatie van de werkelijkheid

Versie: 1.2

Begeleider: Rik Jansen

Teamleden:

- Jan Halsema
- Kevin Veld
- Kaj van Meel
- Mart Noten
- Polle Pas
- Dylan Gomez Vazquez

Versiebeheer

0.1	Opstart document
0.2	Invullen hoofdstukken
1.0	Eerste versie
1.1	Wijzigingen t.o.v. eerste verbeteringen
1.2	Toevoegingen FlaSi

Inhoudsopgave

Inleiding

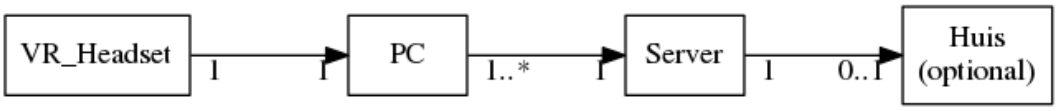
Na aanleiding van het functioneel ontwerp is het technisch ontwerp geschreven. In het technisch ontwerp wordt de data architectuur in beeld gebracht van de back-end services.

Allereerst worden de losse componenten beschreven die in de back-end services zullen worden gebruikt. Sommige onderdelen hiervan verdienen extra toelichting en deze zijn dan ook in dit hoofdstuk besproken.

De gebruikte technieken vormen geen goede basis voor het uitschrijven van de modellen in een ERD of class diagram. Dit komt omdat er niet gebruik is gemaakt van een traditionele relationele database. Toch hebben we het geprobeerd in kaart te brengen om een algemeen beeld te schetsen van hoe de services draaien.

Architectuur

Binnen het project is er een architectuur opgebouwd zodat er een duidelijke verbintenis is tussen de deliverables die zijn bepaald aan het begin van het project.



Devices Systeem Architectuur

Bovenstaande afbeelding laat zien hoe de architectuur is opgebouwd in ons project. Er is een duidelijk afbakening gemaakt in de communicatie tussen de verschillende platformen. Zo kan is er maar een eenrichtings verbinding naar alle andere componenten binnen ons project.

Een VR headset kan alleen in verbinding staan met de PC, dit is zo omdat de VR headset alles laat zien wat gerendered wordt op de PC, de PC verbindt vervolgens met de servers waar alle back end logica staat, deze logica wordt verder in dit document beschreven. Vervolgens maakt de server verbinding met het echte huis. De bedoeling van dit stuk van de verbinding is zodat wanneer er iets in de PC versie van het huis een actie wordt uitgevoerd, zoals een deur die wordt geopend of een lamp die wordt aangedrukt, deze ook werkelijk iets triggered in het fysieke huis.

MongoDB

binnen het project hebben wij de keuze gemaakt om alle data afhandeling op te slaan in MongoDB. MongoDB slaat alle data op in JSON en hierdoor vervallen veel delen in het Technisch ontwerp vanwege hoe JSON wordt opgeslagen.

TCIP en LUA

Voor de verbinding met het huis wordt er gebruikt gemaakt van een TCIP server die gerunned wordt aan de hand van een docker container die LUA scripts afhandeld. Dit subhoofdstuk wordt verder uitgebreid zodra er iets meer duidelijkheid is over de afhandeling hiervan.

HomeLYnk

In het huis wordt er gebruik gemaakt van een HomeLYnk systeem dat is gemaakt door Schneider Electric, dit systeem zorgt voor veel handelingen in het huis. Handelingen zoals het alarm opzetten, automatische berekeningen doen voor het zuinig gebruik van water en verwarmingen etc en lampen bedienen. Dit systeem heeft betrekking met ons omdat HomeLYnk als enige de verbinding kan opzetten met onze server.

NodeJS Server

De webservice is geschreven in NodeJS en die gedraaid wordt in een Docker container. NodeJS is ontworpen voor een snelle setup van webserver en dient daarom goed voor dit project. Het wordt breed gedragen door de community en is geschreven in Javascript[1]. Het dient ter aanspreekpunt van de Virtual Reality game en daarnaast ondersteunt het ook de Angular front-end.

Flask Server

De Flask server zal binnen in het Selficient huis zorgen voor een connectie met de buitenwereld via het internet. Deze zorgt dat er door onze eigen apparatuur kan worden verbonden met de HomeLynk. Het stelt ons ook in staat om de MySQL database van de applicatie uit te lezen.

Docker

Om de onderhoudbaarheid en overdraagbaarheid van het project te vergroten is er voor gekozen om de uiteindelijke hosting van de server en haar database te regelen via Docker containers. Dit vergroot de overdraagbaarheid omdat Docker containers makkelijk verwisselbaar zijn tussen servers.Het vergroot de onderhoudbaarheid omdat Dockers genieten van grote aanpassingsmogelijkheden en ook makkelijk runtime gewijzigd kunnen worden. Het zou later uitgebreid kunnen worden naar een *CI/CD*[2] *omgeving*.

Angular

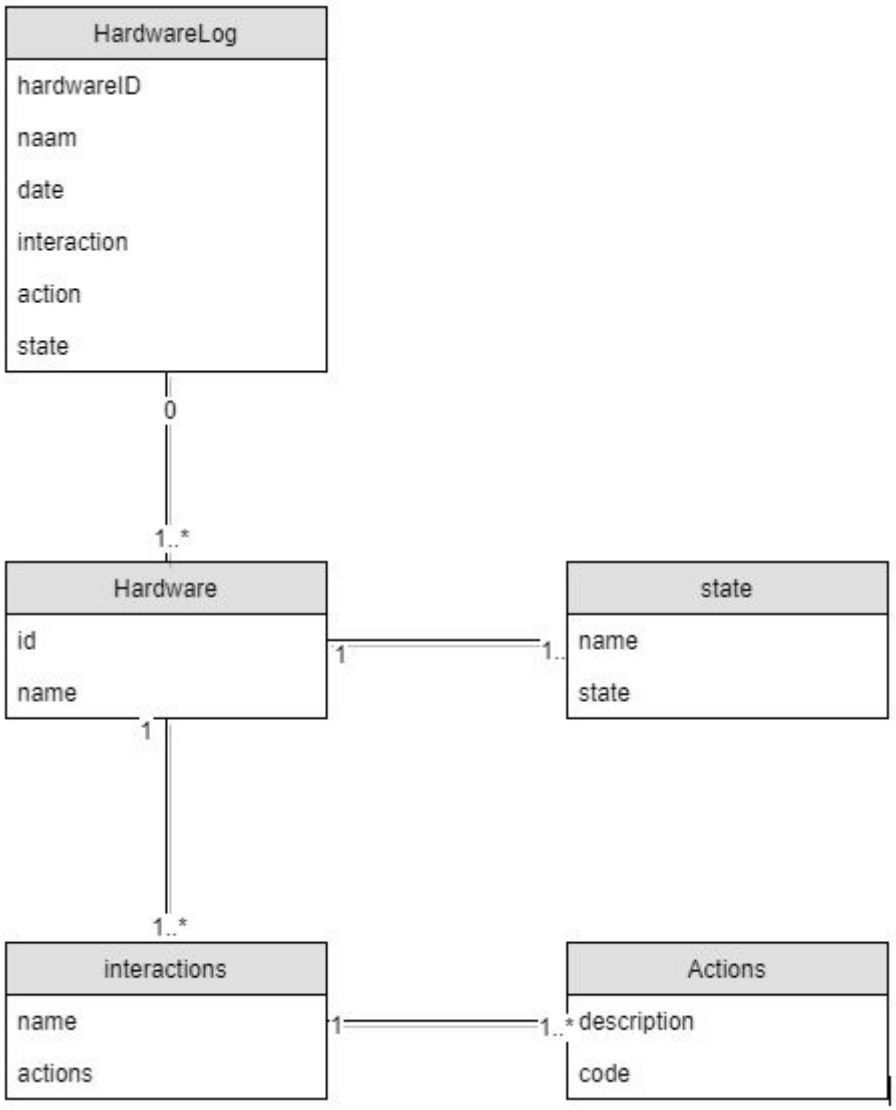
De Angular front-end applicatie dient voor het tonen van de data die is opgeslagen op de centrale server. Het communiceert hiervoor alleen met de NodeJS server.

[1] Verwijzing van https://dzone.com/articles/what-are-benefits-nodejs
[2] Continuse Integration of Continous deployment

Databases

ERD MongoDB

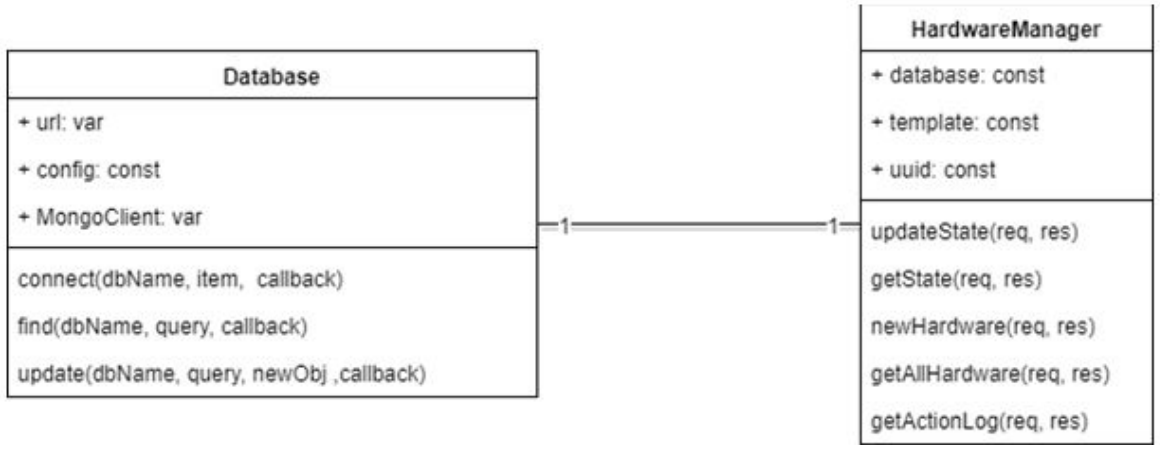
Dit schema toont de relatie tussen de tabellen. Omdat in de praktijk een No-SQL database wordt gebruikt moet dit schema met een schuin oog worden gelezen. Het vertelt ons in ieder geval de verschillende items van een entiteit en de relatie tussen de entiteiten.



Webservice NoSi

De webservice is onderverdeeld in de database en de API, hernoemd naar HardwareManager. Deze heeft verschillende services die betrekking hebben tot het onderhandelen over de data tussen de aanvragen en de database.

Diagram



Details

<table>
<tr><th>Entiteit</th><th></th></tr>
<tr><td>database.js</td><td>
<table>
<tr><th>Attribuut</th><th>Omschrijving</th></tr>
<tr><td>url</td><td>Bevat een geconstrueerde url bestaande uit ‘url:port/scheme’</td></tr>
<tr><td>config</td><td>Bevat de url, port en scheme gegevens van de database.</td></tr>
<tr><td>MongoClient</td><td>Bevat een instantie van de MongoClient.</td></tr>
</table>
<table>
<tr><th>Function</th><th>Omschrijving</th></tr>
<tr><td>connect</td><td>Maakt dmv MongoClient connectie met de database waarin hij vervolgens een database object retourneerd.</td></tr>
<tr><td>insert</td><td>Doet een insert in een collectie afhankelijk van de gegeven dbName en item.</td></tr>
<tr><td>find</td><td>Vind een specifiek stuk hardware en geeft dit terug.</td></tr>
<tr><td>update</td><td>Update een item afhankelijk van de gegeven dBName, query en newObj.</td></tr>
</table>
</td></tr>
</table>

hardwaremanager.js

Attribuut	*Omschrijving*
database	Bevat het database object
template	Bevat hardware gegevens als json format. Zie ERD in het hoofdstuk database.
uuid	Bevat een uuid object waarin snel en simpel een RFC4122 uuid gegenereerd kan worden.

Function	*Omschrijving*
updateState	Update de state van de hardware en voegt een nieuwe log toe.
getState	Geeft de state terug van een stuk hardware.
newHardware	Voegt een nieuw stuk hardware toe
getAllHardware	Geeft alle hardware terug
getActionLog	Geeft alle action logs terug.

HTTP GET/POST request

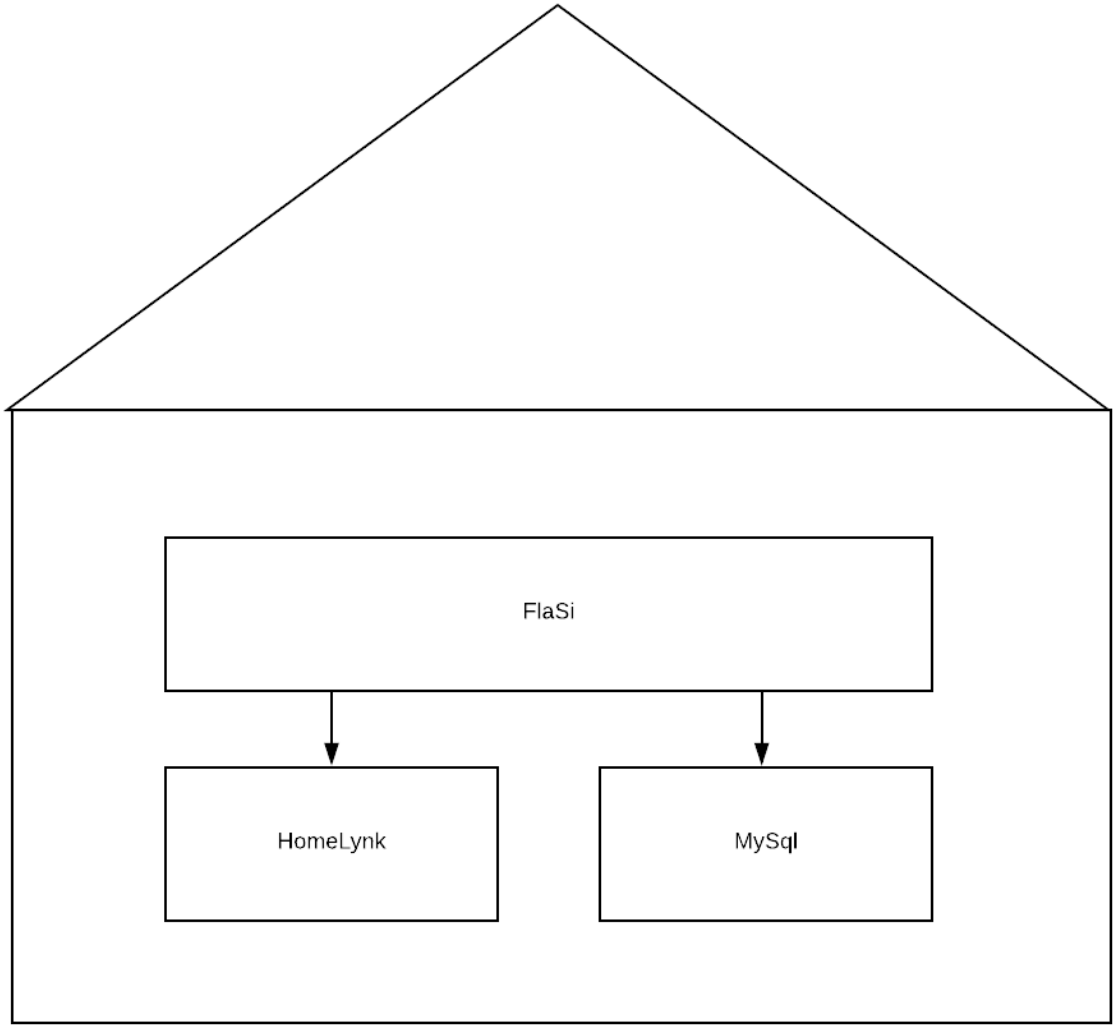
In de onderstaande tabel worden de endpoints omschreven die beschikbaar zijn binnen de API.

Endpoints

url	*type*	*Omschrijving*
/	get	Response is een error message in html/text
/updatestate	post	Roept de updateState function aan in hardwareManager. Response is, wanneer de hardware bestaat, is een succes message.
/getstate	post	Roept getState function aan in hardwareManager. Response is, wanneer de hardware bestaat, is een succes message.
/getallHardware	get	Roept getAllHardware function aan in hardwareManager. Returned een jsonObject.
/getactionlog	get	Roept getActionLog function aan in hardwareManager. Returned een jsonObject.
/new	post	Roept newHardware function aan in hardwaremanager. Response is, wanneer de hardware bestaat, is een succes message.

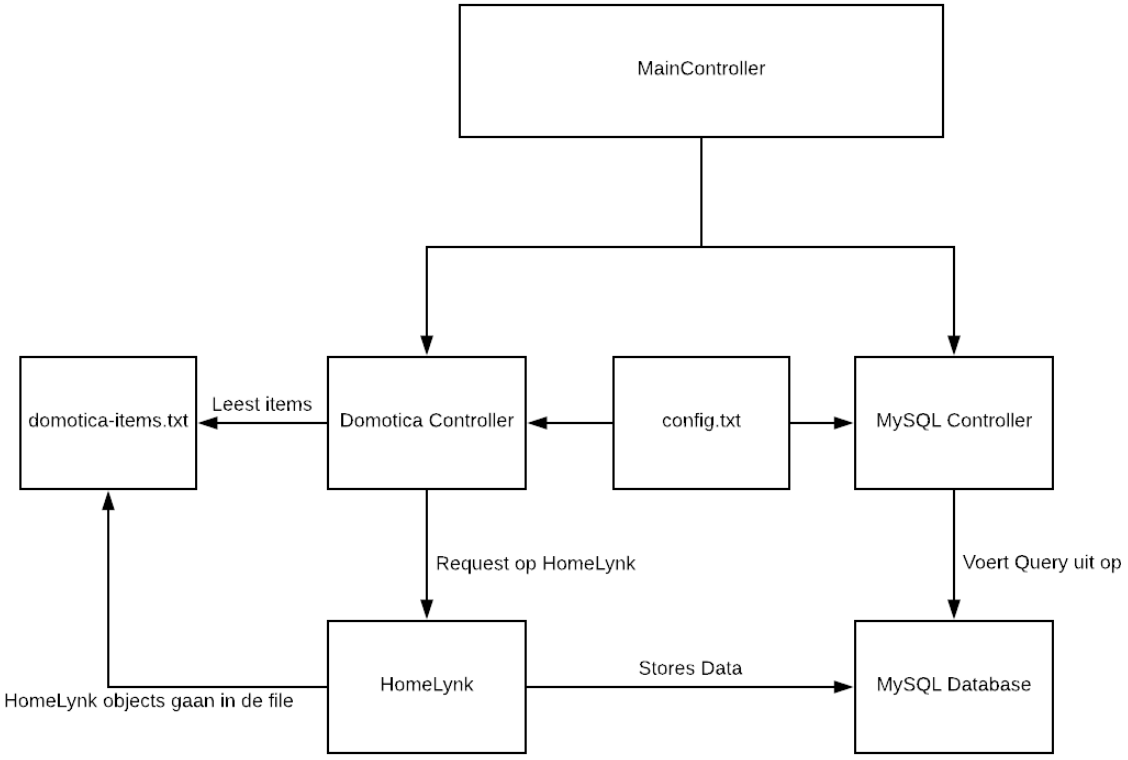
Webservice FlaSi

De FlaSi staat los van de architectuur van de HomeLynk. Het dient als een abstractieniveau op de HomeLynk en haar datastructuren. Het is zich daarom niet bewust van eventuele onderliggende datastructuren of logica. Deze datastructuren en logica liggen bij de partijen waarmee de FlaSi praat (de HomeLynk en haar database)



Details

Omdat de FlaSi zich niet bewust is van onderliggende technologieën wordt er gebruik gemaakt van een duidelijke seperation of concerns. De benodigde informatie om de diensten te laten werken moeten handmatig in de .txt files binnen in de API gezet worden.



HTTP GET/POST request

De requests komen allemaal binnen bij de MainController. Vervolgens worden deze gesplitst naar de categorie zoals bepaald in de url (/domotica gaat naar de domotica controller en /mysql gaat naar de MySQl controller).

Deze controller worden voorzien van de benodigde data door de config files. Vanuit daar weten ze waar ze hun aanvraag moeten doorzetten en vangen ze eventuele fouten af om zo een duidelijke foutmelding af te kunnen geven.

Paths

De paths zijn ook altijd te vinden via de url van de API op het /ui pad.
Bijvoorbeeld: http://127.0.0.1:8080/ui/

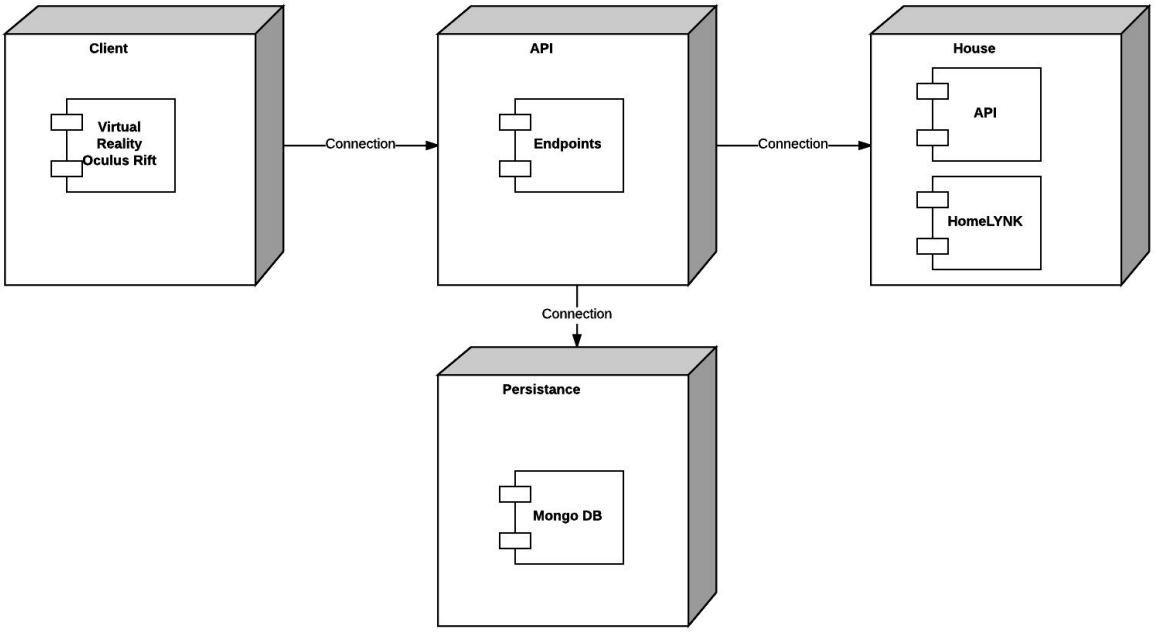
Endpoints

url	*type*	*Omschrijving*
/	get	Response is een error message in html/text
/domotica	get	Haalt alle items op van deze specifieke HomeLynk installatie op
/domotica/{domoticaid}	get	Haalt de specifieke informatie op van het object met {domoticaid}
/domotica/{domoticaid}	post	Activeert een specifieke domotica object met {domoticaid}

/query/execute	post	Queries that what is in the query parameter
		
		Parameter format:
		[
		{
		"name": "query",
		"type": "mysql-query",
		"example": "",
		"value": "select * from aankoop limit 50"
		}
		]

Deployment model

Het deployment model geeft weer op welke nodes de verschillende onderdelen van de applicatie draaien. Elke node heeft een fysieke locatie toegewezen gekregen en daarbij horen verschillende clients die draaien bij deze node. Zo kan opgemaakt worden dat bijvoorbeeld in het huis zowel de API moeten draaien op een eigen cliënt, als een eigen HomeLYNK systeem.



Data flow diagram

Het principe om de data vanuit Unity door te laten stromen naar derde partijen voor hergebruik wordt getoond in het volgende data flow diagram.

Vanuit Unity komt data binnen die verwerkt moet worden door de API. Deze moet vervolgens twee verschillende kanten op:

1. FlaSi: Het aansturen van de domotica sensoren in het echte huis die overeenkomen met de data die komt vanuit Unity
2. Naar de database om opgeslagen te worden voor analyse en verbeteringen ten opzichte van het huidige ontwerp van het huis