Accesstoken = i8k0j9xr9uk1bmrlx7j6

# Data naar data server

De data die de arduino genereerd moet naar een database verstuurd worden. Er is gekozen voor een mySQL database die op een losse dataserver zal draaien. De makkelijkste manier om hier data naar te versturen is door het maken van een script op deze server. Deze accepteert een parameter in de URL die het nieuwe aantal bevat. Echter is dit natuurlijk niet veilig omdat zomaar iedereen op deze manier de database aan kan passen. Een andere oplossing kan zijn een extra parameter meesturen waar een key in zit. Wanneer deze correct is kan het script iets aanpassen. Echter het veiligste is om een RESTful API te maken die deze zaken afhandelen. De manier waarop de arduino data zal verzenden is door een Ethernet module. Deze ethernet module zal naar een specifiek URL een request sturen met de juiste api call. Binnen deze api call zal de juiste data staan die wordt aangepast.

Deze API zal plaatsvinden op een LINUX server besturingssysteem. Ook zal hierop de MySQL database draaien waar de api mee zal communiceren. Vanuit buitenaf zal er een request binnen komen bij de server. Deze zal luisteren naar de request en hem doorsturen naar de database. Dat api zal geschreven worden door middel van Javascript, NodeJs als server logic en Express als http routing.

# Dataserver documentatie

Om de product informatie op te slaan wordt er gebruik gemaakt van een dataserver. Deze draait op een raspberry pi die is verbonden aan het netlab. Op deze manier kan er via een vpn verbinding connectie gemaakt worden met de server. Het ip adress van deze server is 5.104.119.149:8000. Op de server draait nodeJS en een distributie van mySQL. Via een package heb ik een nodeJS server gemaakt die luistert op poort 8000. Wanneer er een request binnen komt zal de backend dit afhandelen en routen naar de juiste database query.

Om de server veiliger te maken zijn er een aantal maatregelen getroffen. De eerste maatregel om indringers buiten te houden is een Public/ Private key pair te gebruiken. Dit is een manier om ongewilde gasten buiten te houden die het wachtwoord van een gebruiker weten of het wachtwoord brute forcen. Hoe dit werkt is: op de server staat een bestand genaamd “authorized\_keys” in dit bestand staat een string waarin staat wat voor type het keypair is en een daarna een public key. Deze is gekoppeld aan één specifieke private key die goed beveiligd staat op het device van de gebruiker. Deze key combinatie is aangemaakt op het device van de client door het volgende command

*ssh-keygen -t ecdsa*

Dit command zorgt er voor dat er een public en private keypair word gemaakt. Het ecdsa gedeelte is de manier waarop de encryptie werkt. Hierna word gevraag waar de keys opgslagen moeten worden. Dit is meestal de .ssh map. Ook moet er een naam van het bestand worden opgegeven. Deze is later nog belangrijk voor het verbinden met de server. Als laatste word er gevraagd een wachtwoord op te geven voor het bestand. Deze is ook belangrijk voor de connectie met de server.

Wanner de keys zijn gegenereerd op de client device kan de **public** key worden overgezet naar de server. Dit heb ik gedaan door op de server in het bestand authorized\_keys handmatig de public key te kopieren. Dit gebeurt door het volgende command

*Sudo nano ~/.ssh/authorized\_keys*

Dit command zorgt er voor dat het bestand kan worden aangepast.

Na getest te hebben dat het werkte met keyfile kon het ssh wachtwoord worden uitgeschakeld. Op deze manier kan er enkel doormiddel van een public key worden ingelogd.

Om nu verbinding te maken met de server moet het volgende command worden gebruikt

ssh -i ~/.ssh/superroute\_laptop\_thijmen-ecdsa [pi@5.104.119.149](mailto:pi@192.168.218.103)

Naast ssh beveiliging beschikt de server ook over een firewall die enkel de poorten toe staat die er gebruikt worden. Dit zijn poort 8000 voor de API en poort 22 voor de ssh verbinding. Het toestaan van poorten worden gedaan door het volgende command

*Sudo ufw allow <poort>*

# API documentatie

De data zal vanuit de arduino doormiddel van een Ethernet module naar de dataserver verzonden worden. Deze ethernet module zal een http request sturen naar het ip adress van de dataserver, deze is 5.104.119.149:8000.

## Getting started

Om verbinding te maken met de api heeft de client een acces token nodig. Deze moet worden verwerkt in de url van de api call. Wanneer deze correct is kunnen er verschillende URL’s worden opgevraagd die verschillende data opvragen of aanpassen.

## Use cases

De API heeft een aantal verschillende doelen. Het moet data weergeven binnen de mobiele app. Hier word alle data voor opgevraagd om dit in een lijst weer te geven. Daarnaast moet er een enkel product te zien zijn, Dus de mogelijkheid bestaat doormiddel van een ID een product op te vragen via de api waarna deze data in JSON vorm zal worden weergegeven. En ten derde zal de api gebruikt worden om data aan te passen. Voorraden moeten worden aangepast van specifieke producten. Dit kan worden gedaan doormiddel van een PUT request.

## URI

Alle returns zijn In JSON format en de API is case sensitive

### Get all stores

Deze endpoint is om alle winkels met bijbehorende schappen op te vragen. Return value is

{

“winkelId”:

“winkelNaam”:

“Adres”:

“winkelLogo”:

}

De store id is essentieel omdat deze later in iedere api call weer moet worden toegevoegd om verdere informatie te krijgen.

Er hoeft bij deze endpoint geen body meegestuurd te worden. Enkel hoeft er een GET request gestuurd te worden naar

*http://5.104.119.149:8000/<accesstoken>/*stores

### Get store by id

Deze endpoint is om een bepaalde winkel met zijn details te krijgen. De gebeurt doormiddel van het id mee te geven over de url. Het id is strict een integer en moet aan het einde worden geplaatst.

Verder is het dezelfde endpoint als *Get all stores*. Wanneer er een id word meegegeven word er een winkel terug gegeven indien het id bestaat. Wanneer er niks word meegegeven worden alle winkels terug gegeven.

*http://5.104.119.149:8000/<accesstoken>/*stores/id

De return format is ook hetzelfde als de *Get all stores*.

### Get winkel assortiment

Deze endpoint is om het gehele assortiment van een bepaalde winkel op te vragen. De api geeft dan een lijst met schappen en zijn details en daarbij de details van het product van wat aan dat schap is gekoppeld.

het format is:

{

“winkelId”: int

“schapId”: int

“ProductId”: int

“productNaam”: string

“productAantal”: int

“productPrijs”: double

“error”: bool

“productFoto”: string

“productGewicht”: int

“catergorieNaam”: string

“SchapLocatie”: string

}

Ook hier moet er weer een winkel id in de request header worden meegestuurd. Dit is het id van de winkel waar het assortiment van moet worden opgevraagd.

*http://5.104.119.149:8000/<accesstoken>/* getAssortiment/id

### Put edit product details

Deze endpoint is om de details van een bepaald product te wijzigen. De nieuwe product details worden meegestuurd over de body in JSON format houdende aan een bepaald patroon. Zo kan de naam en de prijs van een product worden aangepast. Om een bepaald product te bewerken moet er een winkel id en een product id worden meegestuurd over de url.

Header:

*http://5.104.119.149:8000/<accesstoken>/*product/winkelId/productId

Body:

{

*Optional* “ProductNaam”: string,

Optional “ProductPrijs”: int

}

Wanneer de bewerking is gelukt zal er een succes bericht terug gegeven worden.

### Post nieuw product

Deze endpoint is om een nieuw product aan het assortiment toe te voegen. Dit gebeurt door alle benodigde waarden van het product in te vullen. Weer in de body in JSON formaat. In de header moet weer een winkel id worden meegeven om het product aan een bepaalde winkel te koppelen.

Header:

*http://5.104.119.149:8000/<accesstoken>/*addProduct/winkelId

Body:

{

“ProductNaam”: string,

“ProductPrijs”: int,

“ProductFoto”: string

}

Wanneer dit is toegevoegd zijn de aantal en gewicht velden leeg. Deze worden gevuld wanneer het product daadwerkelijk aan een schap word gekoppeld. Dan word er dmv de admin app berekend hoe zwaar het product is en hoeveel er op het schap staan.

### Post product koppelen

Deze endpoint is om een product aan een schap te koppelen. Wat er gebeurt is dat het product id in de schap tabel word geplaats. Dan kan via die manier data opgevraagd worden over het schap. In de header zal het winkel id worden meegestuurd om de schappen en producten op te vragen. En dan zal in de body het schap id en het product id meegegeven worden.

Header:

*http://5.104.119.149:8000/<accesstoken>/*koppelProduct/winkelId

Body:

{

“schapId”: int,

“productId”: int

}

### Delete product ontkoppelen

Let op, deze endpoint is het zelfde als het koppelen. echter is dit een Delete en de ander een Put

Om een product te ontkoppelen van een schap word het product id weer weg gehaald uit de corresponderen schap tabel. Dan komt het product weer te staan bij losse producten en het schap te staan bij de lijst losse schappen. \*zie Get losse producten en Get losse schappen

In de header zal het winkel id en het schap id meegestuurd worden. Bij deze is geen body nodig omdat er dit keer geen product id nodig is. Het product id word gewoon weggehaald bij het schap.

Header: *http://5.104.119.149:8000/<accesstoken>/*koppelProduct/winkelId/schapId

### Get losse producten

Deze endpoint is om een lijst te krijgen met producten die niet aan een schap zijn gekoppeld. Deze gegevens kunnen gebruikt worden om een product juist wel aan een schap te koppelen. Om deze lijst op te vragen is er alleen een winkel id nodig. Deze kan meegegeven worden in de header.

Header

Header: *http://5.104.119.149:8000/<accesstoken>/*losseProducten/winkelId

Output:

{

“ProductId”: int

“productNaam”: string

“productPrijs”: double

“productFoto”: string

}

### Get losse schappen

Deze endpoint is om een lijst van schappen te krijgen waar geen product aan is gekoppeld. Dit kan handig zijn om er een nieuw product aan te koppelen. Deze gegevens worden wederom opgevraagd dor enkel een winkel id in de header mee te sturen.

Header

Header: *http://5.104.119.149:8000/<accesstoken>/*losseSchappen/winkelId

Output:

{

“schapId”: int

“categorieNaam”: string

“schapLocatie”: string

}

### Get product details

Let op! Deze endpoint is het zelfde als het updaten van een product. Alleen is het hier GET en daar PUT

Deze endpoint is om details op te vragen van een bepaald product. Dit gebeurt weer door het winkel id en het product is in de url te zetten. Op deze manier kan er per winkel gezocht worden naar een product.

Header

*http://5.104.119.149:8000/<accesstoken>/*product/winkelId/productId

Output:

{

“ProductId”: int

“productNaam”: string

“productAantal”: int

“productPrijs”: double

“error”: bool

“productFoto”: string

“productGewicht”: int

}

### Put update product voorraad

Deze endpoint is om de voorraad van een product aan te passen. Deze word enkel gebruikt door het schap en dus de ESP32. Ook hier moet weer een winkel id en een product id meegestuurd worden. Echter draagt de ESP32 al bij default het winkelId waar deze aan is gekoppeld bij dus dit gebeurt automatisch. In de body moet de aangepaste voorraad worden meegestuurd.

Header:

*http://5.104.119.149:8000/<accesstoken>/*updateVoorraad/winkelId/productId

Body:

*{“Aantal”: <Aantal>}*

### Get categoriën

Deze endpoint is om alle categoriën op te halen. Er hoeven geen parameters mee gegeven te worden.

Header:

*http://5.104.119.149:8000/<accesstoken>/*categorien

Output:

{

“categorieId”: int

“categorieNaam”: string

}