12-12-2016

Arend-Jan van Hilten & Vincent Kling

KAlsbeek College

Slimme Koelkast

PWS

Inhoud

[Het idee 5](#_Toc469250729)

[De stappen 7](#_Toc469250730)

[Inventarisatie 7](#_Toc469250731)

[Pre-engineering 8](#_Toc469250732)

[Prototyping/ Proof of concept 8](#_Toc469250733)

[Detail engineering 8](#_Toc469250734)

[Vervaardiging 8](#_Toc469250735)

[Assemblage 8](#_Toc469250736)

[Testen & Afname 8](#_Toc469250737)

[Onderzoeken 9](#_Toc469250738)

[Data 10](#_Toc469250739)

[XML 10](#_Toc469250740)

[JSON 11](#_Toc469250741)

[YAML 11](#_Toc469250742)

[AXON 12](#_Toc469250743)

[Conclusie 12](#_Toc469250744)

[Enquetes 13](#_Toc469250745)

[Enquête 13](#_Toc469250746)

[Enquête 2 16](#_Toc469250747)

[Conclusie 18](#_Toc469250748)

[Hardware 19](#_Toc469250749)

[Raspberry Pi 20](#_Toc469250750)

[Knoppen 21](#_Toc469250751)

[Elektronica 21](#_Toc469250752)

[Barcodescanner 25](#_Toc469250753)

[Koelkast 27](#_Toc469250754)

[Thermische Printer 28](#_Toc469250755)

[Andere hardware 28](#_Toc469250756)

[Architectuur 30](#_Toc469250757)

[Software 31](#_Toc469250758)

[Raspberry Pi 31](#_Toc469250759)

[Overview 32](#_Toc469250760)

[Install.py 33](#_Toc469250761)

[Settings.py 35](#_Toc469250762)

[Start.py 35](#_Toc469250763)

[Main.py 35](#_Toc469250764)

[Threads.py 36](#_Toc469250765)

[keyboardThread.py 36](#_Toc469250766)

[Knoppen 36](#_Toc469250767)

[Barcode 37](#_Toc469250768)

[AwakeThread.py 38](#_Toc469250769)

[JobsThread.py 38](#_Toc469250770)

[Jobs.py 39](#_Toc469250771)

[Restart/shutdown 40](#_Toc469250772)

[Tekst 41](#_Toc469250773)

[List 42](#_Toc469250774)

[Barcode 42](#_Toc469250775)

[Server 43](#_Toc469250776)

[Database 44](#_Toc469250777)

[PHP 45](#_Toc469250778)

[Statements 45](#_Toc469250779)

[newItem 47](#_Toc469250780)

[openItem 47](#_Toc469250781)

[delItem 48](#_Toc469250782)

[getItem 49](#_Toc469250783)

[Globals 50](#_Toc469250784)

[Generate UserId 50](#_Toc469250785)

[Active 51](#_Toc469250786)

[titleChange 51](#_Toc469250787)

[Error logging 52](#_Toc469250788)

[App 54](#_Toc469250789)

[Foto’s + uitleg 54](#_Toc469250790)

[Login 55](#_Toc469250791)

[Home 56](#_Toc469250792)

[Menu 57](#_Toc469250793)

[Jobs 58](#_Toc469250794)

[Logs 59](#_Toc469250795)

[Shopping List 60](#_Toc469250796)

[Technische details 61](#_Toc469250797)

[Overview 61](#_Toc469250798)

[Smartfridge API 64](#_Toc469250799)

[SmartfridgeSave.java 66](#_Toc469250800)

[Boodschappenlijstje 68](#_Toc469250801)

[Issues app 69](#_Toc469250802)

[Willekeurige crashes 69](#_Toc469250803)

[Opslaan 69](#_Toc469250804)

[Dubbelen 69](#_Toc469250805)

[Documentatie 70](#_Toc469250806)

[Andere software 70](#_Toc469250807)

[Licenties 71](#_Toc469250808)

[Versiebeheer 71](#_Toc469250809)

[Open Source & Hackster.io 72](#_Toc469250810)

[Verbeteringen 73](#_Toc469250811)

[Conclusie 74](#_Toc469250812)

[Discussie 75](#_Toc469250813)

[Logboek 76](#_Toc469250814)

[Kosten 76](#_Toc469250815)

[Eigen kosten 76](#_Toc469250816)

[Normale kosten 77](#_Toc469250817)

[Bedanken 78](#_Toc469250818)

[Bronnen 79](#_Toc469250819)

[Data onderzoek 79](#_Toc469250820)

[Barcode Scanner 79](#_Toc469250821)

[Install.py 79](#_Toc469250822)

[KeyboardThread.py 79](#_Toc469250823)

[Algemeen 79](#_Toc469250824)

# Het idee

Het idee: Een slimme koelkast….

Maar hoe dan precies?

We willen een computer in een koelkast zetten met een invoerapparaat dat weet welk product er in/uit gaat en een app om de lijst met inhoud op te bekijken.

Groep:

 Arend-Jan van Hilten

 Vincent Kling

Onderwerp:

 Internet Of Things

o Industrial Internet Of Things

 Domotica

Hoe we dit willen gaan uitwerken:

* Eigen domoticaproject die aan het internet hangt
* App
* We dachten aan een slimme koelkast
* Website erbij
* Misschien kunnen we aan een Toon display komen

 Misschien een bezoek aan een paar bedrijven over IIOT of domotica

o VolkerWessel(treinwissels aan het internet)

o Quby(Toon, slimme thermostaat)

 Onderzoek over wat nou de beste manier is om data over het internet te versturen:

* XML
* JSON
* PAIR-VALUE
* Andere manier

Hoofdvraag:

- Hoe komt een slimme koelkast tot stand?

Deelvragen:

- Welke stappen zijn essentieel voor het maken van een slim apparaat?

- Welk formaat is het beste om data te sturen?

- Wat vinden mensen van het idee slimme koelkast?

Relevantie:

Internet of Things is steeds aan het vergroten en verbreden. Steeds meer slimme apparaten worden ontwikkeld en uiteindelijk zullen alle, of in ieder geval heel veel, apparaten verbonden zijn met het internet.

Een beschrijving/ verantwoording van het soort onderzoek en een beschrijving van de methode per deelvraag:

- Welke stappen zijn essentieel voor het maken van een slim apparaat?

Tijdens het onderzoek bekijken we wat het belangrijkste is om het geheel te laten werken.

- Welk formaat is het beste om data te sturen?

Hier gaan we vooronderzoek doen in de bronnen. Daarnaast doen we ons eigen onderzoek en kijken we wat in ons geval het beste uit komt. We kunnen de resultaten vergelijken en zo bedenken of wat in ons geval het beste uit komt ook voor grotere bedrijven handig is.

Methode van dataverzameling:

Tijdens ons onderzoek kunnen we gebruik maken van bestaande bronnen. Echter is het interessanter om zelf kleinere onderzoekjes te doen om te kijken wat in ons geval beter uit komt. Deze uitkomsten kunnen we dan vergelijken met de bronnen.

# De stappen

Voor het maken van een slimme koelkast zijn er veel stappen. Deze stappen nemen wij allemaal door in dit verslag. Tijdens deze stappen komen veel dingen aan bod die tijdens het ontwerpen van een gebruiksproduct ook voorkomen. Hierbij valt te denken aan de mens-machine interactie, ook wel afgekort naar MMI. Bij het ontwikkelen van een product zijn er 8 stappen in de volgende volgorde:

* Inventarisatie
* pre-engineering
* prototyping/proof of concept
* detail engineering
* vervaardiging
* assemblage
* testen & afname.

Voor ons PWS doen we eigenlijk alleen maar de inventarisatie, pre-engineering en prototyping/proof of concept. De latere stappen zijn eigenlijk voor wanneer je een product echt wilt laten produceren.

## Inventarisatie

De eerste stap is de inventarisatie, dit houdt in dat er wordt gebrainstormd over wat de eisen zijn van het product. En naar aanleiding van de eisen wordt er een plan opgezet om te bepalen wat de beste manier is om deze eisen te behalen.

Bij onze inventarisatie hebben we ook als eerste bepaald wat onze slimme koelkast voor een features moet hebben. Hierbij hebben we onder andere gebruik gemaakt van een enquête. Wij hebben bepaald dat onze slimme koelkast op zijn minst een barcode scanner nodig heeft om de producten te scannen. Om te bepalen wat er met de producten gebeurd, zoals erin of eruit, hebben we besloten dat er ook knoppen in moeten komen. Je drukt dan eerst op een knop en dan haal je het product langs de scanner. Daarnaast vonden mensen het handig als er automatisch een boodschappenlijstje geproduceerd kan worden. Om deze feature te versterken hebben wij bedacht dat een thermische printer ook interessant is. Met deze printer kan een boodschappenlijstje geprint worden, zodat mensen iets fysieks hebben om mee te nemen naar de winkel.

Om de barcodescanner, thermische printer en knoppen te kunnen gebruiken voor invoer/uitvoer hebben wij gekozen om een Raspberry Pi te gebruiken.

Daarnaast moet alle data voor het voorraadbeheer opgeslagen worden om ook later weer op te kunnen vragen. Hiervoor hebben wij bepaald dat we een webserver met een database nodig hebben. De webserver krijgt de opdrachten met variabelen van de Raspberry, de server voert dan de opdracht uit en voegt iets toe in de database of leest juist iets uit.

## Pre-engineering

Pre-engineering is het bekijken van bepaalde onderdelen van het te maken product. Hierdoor kan je bepalen wat voor en nadelen kunnen zijn bij deze onderdelen.

Een eerste onderdeel van onze pre-engineering was het onderzoeken op welke manier wij onze data willen versturen.

Daarnaast moeten we door middel van pre-engineering bepalen hoe wij alles aan willen sluiten. Zouden we extra onderdelen nodig moeten hebben waar we eerst niet aan hebben gedacht?

## Prototyping/ Proof of concept

Prototyping/ proof of concept is zoals de naam al zegt het maken van een prototype. Met dit prototype heb je eigenlijk een geheel wat je volledig kan testen. Op basis van deze testen kan er bepaald worden of er weer om de tekentafel gezeten moet worden, of dat het door kan naar de detail engineering. Met een prototype kunnen mensen ook zien wat er daadwerkelijk op de markt gaat komen.

Voor ons PWS is het prototype eigenlijk onze eindfase en zal het ook onze presentatie zijn.

## Detail engineering

Nadat er door middel van een prototype duidelijk is wat er gemaakt moet gaan worden, kan er ook naar de details gekeken worden. Dit gebeurt tijdens het detail engineering. Hier kunnen we naar kijken na het afmaken van ons profielwerkstuk. Als we het open source maken en mensen gaan het daadwerkelijk gebruiken, dan zullen die ook ideeën en code terugsturen. Hierdoor kunnen we de slimme koelkast verbeteren.

## Vervaardiging

Nadat er na de detail engineering groen licht wordt gegeven, kunnen alle onderdelen gefabriceerd worden. Dit onderdeel is onmogelijk voor ons, vooral vanwege het feit dat we een budget van ± 0 hebben en er waarschijnlijk ook een userbase van niet groot is. Het zou wel kunnen dat een bedrijf ons idee zo goed vindt dat ze delen overnemen en dat het uiteindelijk wordt gefabriceerd. Daarnaast zou een project op bijvoorbeeld Kickstarter wel kunnen slagen.

## Assemblage

Al deze losse onderdelen worden daarna, in elkaar gezet zodat er een product is. Wij maken alles in het koelvak van de koelkast zelf. Wanneer een product zoals dit zou worden gemaakt, dan wordt natuurlijk de elektronica in de koelkast gebouwd. Hierom zetten wij de koelkast niet aan, vocht is namelijk niet zo bevorderlijk voor elektronica.

## Testen & Afname

Na het assembleren wordt er nog voor een laatste keer grondig getest. Het wilt nog weleens gebeuren dat een product toch niet goed werkt wanneer het geassembleerd is. Wanneer het product door de tests heen komt, kan het product eindelijk verkocht worden.

# Onderzoeken

Voor dit PWS hebben we drie deelonderzoeken gedaan. Het eerste onderzoek was om te kijken wat de handigste manier is om de vele data te versturen. Het tweede onderzoek is een enquête over wat mensen denken over een slimme koelkast. Als laatste hebben we nog een enquête gehouden over wat mensen nu vinden van het eindresultaat.

Het hoofdonderzoek is natuurlijk het hele proces. We onderzoeken of we echt een slimme koelkast kunnen maken en ook hoe we dan een slimme koelkast kunnen maken. Na het maken kunnen we nog een onderzoekje doen hoe mensen het uiteindelijk echt vinden.

## Data

Voor ons PWS moet er veel data worden verzonden van de Raspberry Pi naar de database, van de app naar de database. Ook moet er veel data opgevraagd worden vanuit de database. Hiervoor moeten we kijken wat de verschillende manieren om data te verzenden zijn. Daarnaast moeten we kijken wat de voordelen of de nadelen van die manieren zijn en ze afwegen. Voor een goede manier is het belangrijk dat ze zo weinig mogelijk karakters nodig hebben om de data te versturen(lengte) en dat ze door zo veel mogelijk programmeertalen ondersteund worden(ondersteuning). We denken dat JSON de beste manier is.

We vergelijken hier 4 methoden, waarvan we er al 3 kenden: XML, JSON, en YAML. Door te googelen naar alternatieven kwamen we AXON tegen.

Voorbeeld case:

Er wordt een geopende fles cola uit de koelkast gehaald en er word een nieuwe in gezet.

### XML

<?xml version="1.0" encoding="UTF-8"?>  
<koelkast\_data>  
<product>  
 <name>Cola</name>

<inout>In</inout>  
  <geopend>N</geopend>  
</product>

<product>  
  <name>Cola</name>

<inout>Out</inout>  
  <geopend>Y</geopend>  
</product>  
</koelkast\_data>

Voordelen:

* Leesbaar

Nadelen:

* Groot

### JSON

{"koelkast\_data":[  
    {"Name":"Cola", "InOut":"In", "Geopend":"N"},  
    {"Name":"Cola", "InOut":"Out", "Geopend":"Y"}  
]}

Voordelen:

* Leesbaar, ook voor iemand die er een klein beetje verstand van heeft
* Compact

Nadelen:

* Onoverzichtelijk bij te veel arrays en objecten

### YAML

Koelkast:

In:

- Product: {name: Cola, geopend:N}

Out:

- Product: {name: Cola, geopend:Y}

Voordelen:

* Leesbaar, ook voor iemand die er een klein beetje verstand van heeft
* Compact
* Geen aanhalingstekens

Nadelen:

* Kan geen tabs gebruiken, maar moet 4 keer spatie doen.

### AXON

koelkastdata

product{name:"Cola" InOut:"In" geopend:"N"}

product{name:"Cola" InOut:"Out" geopend:"Y"}

Voordelen:

* Leesbaar, ook voor iemand die er een klein beetje verstand van heeft
* Compact
* Geen komma’s

Nadelen:

* Deze method is nieuw en daardoor nog niet zo ver ontwikkeld (versie 0.9)
* Niet met PHP mogelijk

Bij deze voorbeelden worden er maar 2x 3 waardes verzonden, waarbij de naam van het product. In werkelijkheid wordt de barcode verzonden. Door gebruik te maken van 2 tabellen kunnen we deze aan elkaar “koppelen”. Als wel op de site of in de app iets willen uitlezen, vragen we de barcode op, met deze barcode kunnen we in de tweede tabel de informatie zien over deze producten.

### Conclusie

Op het eerste gezicht leek AXON leuk om te gebruiken omdat het een combinatie is van meerdere manieren om data te versturen en omdat het voor ons allebei nieuw is. Helaas is AXON (nog) niet te gebruiken met PHP. Het makkelijkste in ons geval is om JSON te gebruiken, omdat het in verhoudingminder nadelen heeft en in de meeste programmeertalen is te gebruiken.

Dus het antwoord op onze tweede deelvraag (Wat is het beste formaat om data te versturen?) is JSON.

## Enquetes

### Enquête

Voor dit PWS hebben we ook een enquête gemaakt, met een aantal vragen over een slimme koelkast.

Als eerste is het natuurlijk interessant waar mensen aan denken bij het horen van een slimme koelkast. Over het algemeen denken mensen gelijk aan het bijhouden van wat er in de koelkast zit. Er zijn ook mensen die denken dat een slimme koelkast zuiniger is met stroom. Wat ook heel veel voor komt is het bijhouden van houdbaarheidsdatums, zodat je een berichtje krijgt als een product bijna over datum is.

Mensen zien over het algemeen wel het gemak in van een slimme koelkast, afhankelijk van wat ze denken dat een slimme koelkast is kregen wij antwoorden zoals: het is duurzaam, meer inzicht over je koelkast, het maakt boodschappen doen praktischer en je gooit minder voedsel weg wat over de datum raakt.

Mensen zien ook nadelen in een slimme koelkast. Zo wordt er gezegd dat een slimme koelkast vrij duur zal zijn, maar er wordt ook een ethische kwestie aan het licht gebracht. Mensen zijn er van bewust dat misschien de privacy op het spel staat. Er wordt hier vooral gedacht aan het feit dat op basis van de inhoud er bepaald kan worden van wat voor een persoon de koelkast is, maar ook dat een hacker kan zien of mensen thuis zijn op basis van de inhoud van de koelkast.

Daarnaast hebben we vier vragen gesteld over functies, waarvan wij dachten dat die belangrijk zijn. Deze vragen waren te beantwoorden met een getal van 0 tot en met 10, waarbij 0 helemaal niet graag is en 10 is heel erg graag.

Uit de gegevens blijkt dat mensen het vooral belangrijk vinden dat er bijgehouden wordt wat er in de koelkast zit, dit in combinatie met een app waar ze dit kunnen zien en ook kunnen zien wat ze zouden moeten bestellen/kopen. Het printen van barcodes voor speciale producten zoals fruit vonden mensen een minder belangrijke functie.

Bijhouden wat er in de koelkast zit scoort gemiddeld een 8,07

Boodschappenlijstjes printen scoort gemiddeld een 6,89

Barcodes printen voor speciale producten zoals fruit scoort gemiddeld een 4,80

Een app die vertelt wat er in de koelkast zit en weet wat er nog besteld moet worden scoort gemiddeld een 7,92

Bij de overige opmerkingen blijkt dan mensen aardig enthousiast zijn. Er zijn zelfs mensen die wel een slimme koelkast van ons zouden willen kopen.

### Enquête 2

Als tweede enquête hebben wij aan mensen gevraagd wat ze nu uiteindelijk vinden van het eindresultaat. Aangezien wij bij deze enquête niet zo lang hebben gewacht op reacties zijn er niet zo veel gegevens als bij de vorige enquête.

We hebben weer gevraagd wat mensen vinden van de functionaliteit om bij te houden wat er in de koelkast zit.

Dit keer scoort dit item gemiddeld een 7,67, wat iets lager is dan bij de eerste enquête.

Ook hebben we weer gevraagd wat mensen ervan vinden dat er automatisch een boodschappenlijstje geprint kan worden.

Dit keer scoort dit item gemiddeld een 7,80. Dit item scoort juist weer iets hoger dan bij de vorige enquête.

De laatste functionaliteit waar naar wij de mening hebben gevraagd is het printen van een barcode voor speciale producten zonder barcode.

Deze functie scoort nu gemiddeld een 6,93. Dit is ineens een stuk hoger dan bij de eerste enquête.

Omdat we nu ook een app hebben, hebben we ook aan de mensen gevraagd wat ze vinden van de app.

Over het algemeen zijn mensen wel tevreden met de app. Bijna iedereen vindt het er netjes en zakelijk uit zien. Ze zeggen dat het er juist netjes uit ziet omdat er niet te veel poespas in zit: “Less is more”.

### Conclusie

Over het algemeen (eerste en tweede enquête samen) zijn mensen dus positief gezind over het idee van een slimme koelkast, wat dus onze laatste deelvraag beantwoordt.

# Hardware

Een belangrijk onderdeel van ons onderzoek is het onderdeel hardware. In dit hoofdstuk leggen we uit welke onderdelen we hebben gekozen en nog belangrijker waarom. Ook laten we zien hoe we die hardware gebruiken en samengesteld hebben.

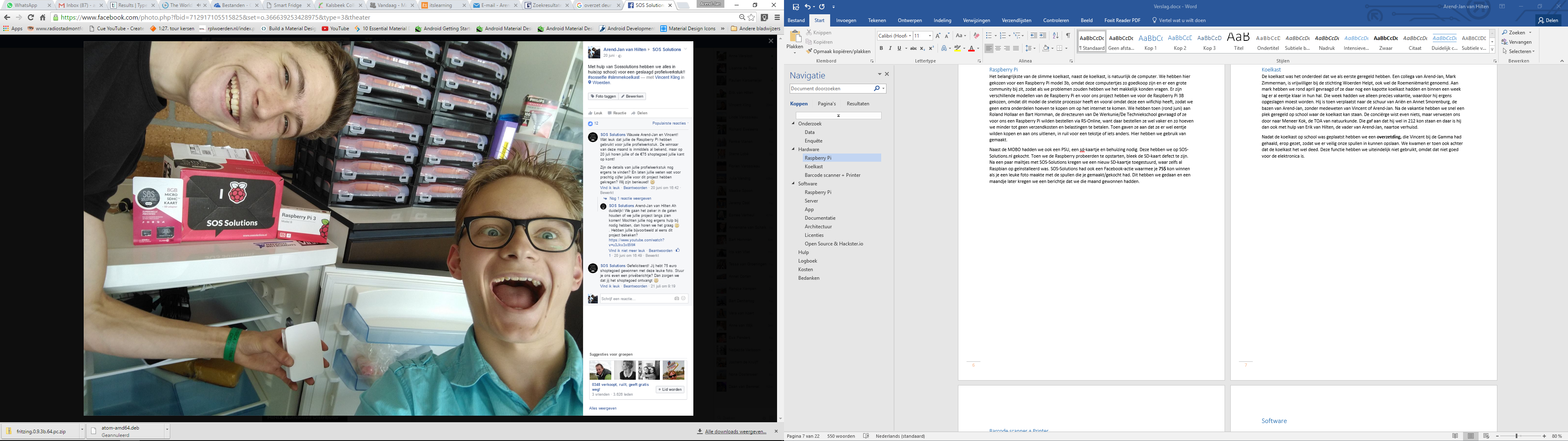
****

## Raspberry Pi

Het belangrijkste van de slimme koelkast, naast de koelkast, is natuurlijk de computer. We hebben hier gekozen voor een Raspberry Pi model 3b, omdat deze computers zo goedkoop zijn en er een grote community bij zit, zodat als we problemen zouden hebben we het makkelijk konden vragen. Er zijn verschillende modellen van de Raspberry Pi en voor ons project hebben we voor de Raspberry Pi 3B gekozen, omdat dit model de snelste processor heeft en vooral omdat deze een wifichip heeft, zodat we geen extra onderdelen hoeven te kopen om op het internet te komen. We hebben een Raspberry Pi te leen gekregen van Roland Hollaar en Bart Hornman van de Werkunie.

Naast de MOBO hadden we ook een voeding, een sd-kaartje en behuizing nodig. Deze hebben we bij SOS-Solutions besteld.

Bij SOS-Solutions hebben we 75 euro tegoed gekregen met deze foto:



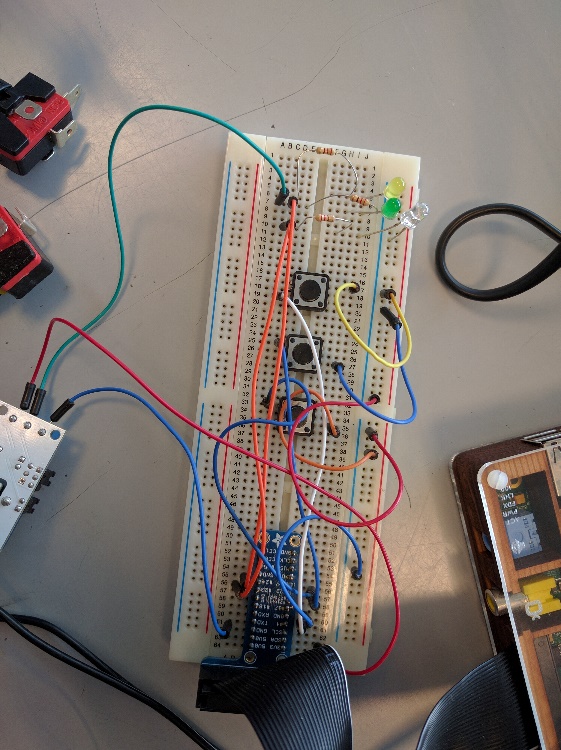
Met dit shoptegoed hebben we eind augustus o.a. een USB-TTL kabel gekocht, zodat we met een laptop de terminal kunnen besturen, zonder wifi te gebruiken. Daarnaast hebben we ook 3 arcade knoppen gekocht, die in onze koelkast zijn gemonteerd. Deze knoppen hebben we gekozen vanwege de nerdfactor en ook vanwege de grootte.

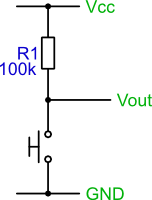
## Knoppen

De knoppen die we hebben gekozen zijn 3 arcade knoppen met ingebouwde verlichting. We hebben voor de kleuren rood, groen en blauw gekozen. Deze kleuren corresponderen met de actie die de gebruiker maakt: rood staat voor eruit halen, groen voor een nieuw item erin doen en blauw voor een geopende verpakking. Deze knoppen hebben we op het bovenste plankje van de deur geplaatst, omdat we denken dat op deze manier het gemakkelijker is om even op de knop te drukken dan als hij diep verstopt is achterin. We hebben met een speedboor van Meneer Kok 3 gaten geboord, maar vanwege het feit dat de binnenkant van de deur een beetje schuin en bol loopt, zitten de moeren niet helemaal strak en kunnen de knoppen een beetje bewegen.

### Elektronica

Met alleen maar fancy knoppen in een koelkastdeur kan je niet zo veel, dus moesten we de knoppen aansluiten aan de Raspberry Pi. Hiervoor hebben we eerst een testopstelling gemaakt en getest. Deze testopstelling bestond uit een Raspberry Pi 1B, een ouder model, waarvan het niet zo erg zou zijn als deze kortsluiting kreeg of kapotging, wat gelukkig niet is gebeurd. Deze Raspberry Pi was met een bandkabel verbonden aan een breadboard, zodat we met draadjes van alles konden uitproberen.

Op dit breadboard hebben we o.a. 3 knopjes gezet en 3 ledjes, omdat dat samen dan een knop is. Toen we dat werkend hadden qua goede pinout en een klein beetje code hebben we alles naar de knoppen overgezet. Dit ging niet helemaal makkelijk. De leds in de knoppen waren gemaakt voor maximaal 12 volt terwijl de Raspberry Pi maar 3.3 volt kan leveren en zelfs met een transistor, wat moeilijk is om te maken, zou het maximaal op 5 volt komen, wat bijna niet zichtbaar is bleek uiteindelijk. We hebben toen een 12 volt voeding genomen (dat stond erop, maar de voltmeter gaf ~9 volt aan) en 3 relais geleend bij De Werkunie. Deze relais hebben we aangesloten en samen met de voeding gaven ze wel genoeg power om de leds in de knoppen te laten branden.

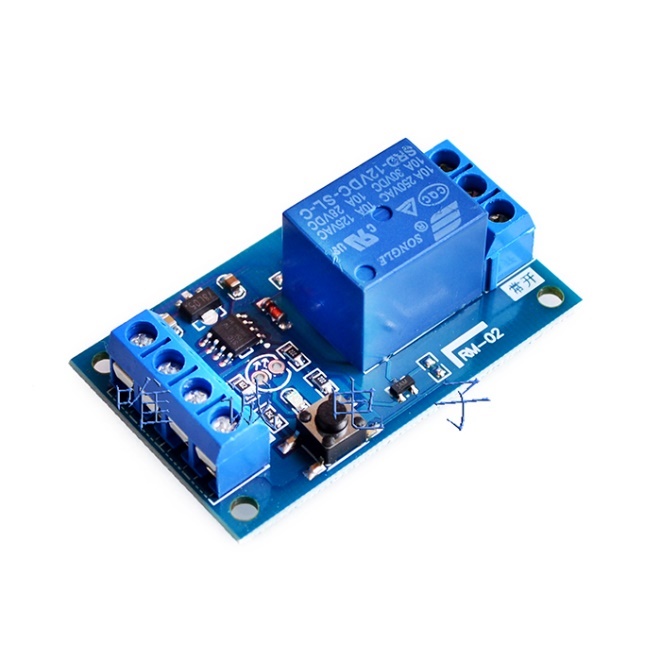
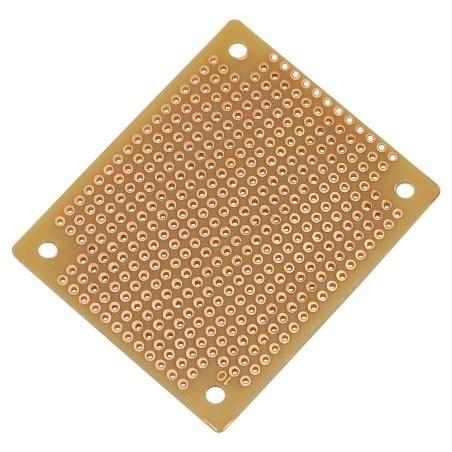
Bij de knoppen hebben we gebruik gemaakt van ingebouwde pullupweerstanden, dat betekent dat er standaard een klein stroompje door een weerstand loopt die aan de input en de Vcc, 3,3 volt, is gekoppeld, waardoor de input de hele tijd een logische 1 meet. Als de knop ingedrukt wordt dan wordt het signaal naar beneden ‘getrokken’: omdat de weerstand zo hoog is, loopt er weinig stroom, ook al wordt het signaal ‘kortgesloten’. De pin krijgt dan een logische 0 binnen. In de Raspberry Pi, de Broadcom chip, zit deze weerstand al ingebakken, die alleen in werking gesteld moet worden. Dit is erg makkelijk te doen door de regel

te gebruiken in python. Dit stelt gelijk de ‘pin’ knopIn (variabele integer) in als een input.

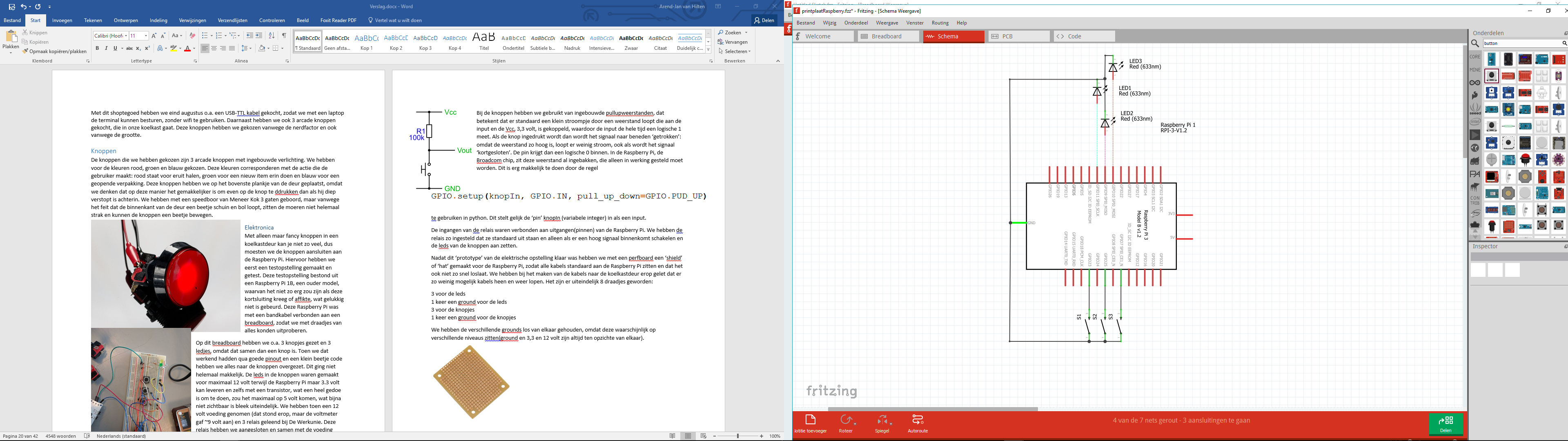
De ingangen van de relais waren verbonden aan uitgangen(pinnen) van de Raspberry Pi. We hebben de relais zo ingesteld dat ze standaard uit staan en alleen als er een hoog signaal binnenkomt schakelen en de leds van de knoppen aanzetten.

Nadat dit ‘prototype’ van de elektrische opstelling klaar was hebben we met een perfboard een ‘shield’ of ‘hat’ gemaakt voor de Raspberry Pi, zodat alle kabels standaard aan de Raspberry Pi zitten en dat het ook niet zo snel loslaat. We hebben bij het maken van de kabels naar de koelkastdeur erop gelet dat er zo weinig mogelijk kabels heen en weer lopen. Het zijn er uiteindelijk 8 draadjes geworden:

3 voor de leds  
1 keer een ground voor de leds  
3 voor de knopjes  
1 keer een ground voor de knopjes

We hebben de verschillende grounds los van elkaar gehouden, omdat deze waarschijnlijk op verschillende niveaus zitten(ground en 3,3 en 12 volt zijn altijd ten opzichte van elkaar). 

Heel erg vereenvoudigd, dus zonder relais, voedingen (en mogelijk met de verkeerde pinnen), ziet het schema er zo uit:



We wilden nog een printplaat laten etsen, maar vanwege de lange levertijd van meer dan een maand hebben we dit niet gedaan.

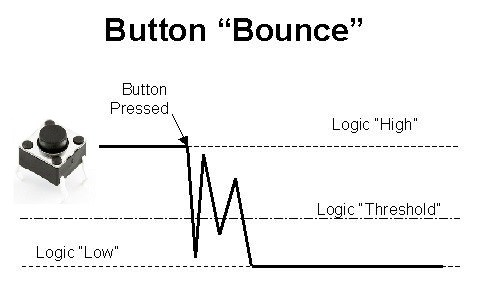
De code op de Raspberry Pi is als volgt opgesteld:

Initializeer de 3 uitgangen  
Initializeer de 3 ingangen met pullup weerstand  
zolang altijd:

Check knop 1  
Als knop 1:  
 zet de globale variabele in op knop 1  
Check knop 2  
Als knop 2:  
 zet de globale variabele in op knop 2  
Check knop 3  
Als knop 3:  
 zet de globale variabele in op knop 3  
Als huidige waarde anders is dan vorige waarde:  
 zet led van knop {vorige waarde} uit  
 zet led van knop {huidige waarde} aan  
wacht 0,1 seconde

We hebben die wachttijd van 1 seconde erin gezet, zodat de andere code ook gebruik kan maken van de processor. We waren op een keer aan het testen en alles draaide heel erg langzaam, het uitlezen van de scanner duurde meer dan 10 seconden en het ophalen van de jobs ging ook niet echt snel. We hebben toen geprobeerd om die onderdelen sneller te maken door overbodige code te verwijderen en te optimaliseren. Pas na heel wat testen en gedoe kwamen we erachter dat deze thread heel erg veel CPU innam, waardoor alles minder CPU kreeg en langzamer draaide. We hebben hier heel erg van geleerd, dat je niet per se naar de onderdelen die langzaam draaien moet kijken, maar ook naar de onderdelen die juist te snel draaien. We hebben voor het getal 0.1 gekozen, omdat het dan nog vaak genoeg gepollt wordt, 10 keer per seconde, dus het is moeilijk om zo snel te drukken dat de Raspberry Pi het niet kan meten. In de code en ook in het schema hoeven we ons geen zorgen te maken over het bouncen van het signaal, want het aantal knopdrukken maakt niet uit en mede door de wachttijd is de bounce er al uit voor het begin van de volgende loop.

Een button bounce er wanneer de knop niet meer ingedrukt wordt, maar de geleiders nog een beetje bewegen en nog een beetje doorlaten. Je krijgt door een bounce nooit een mooie blokgolf.

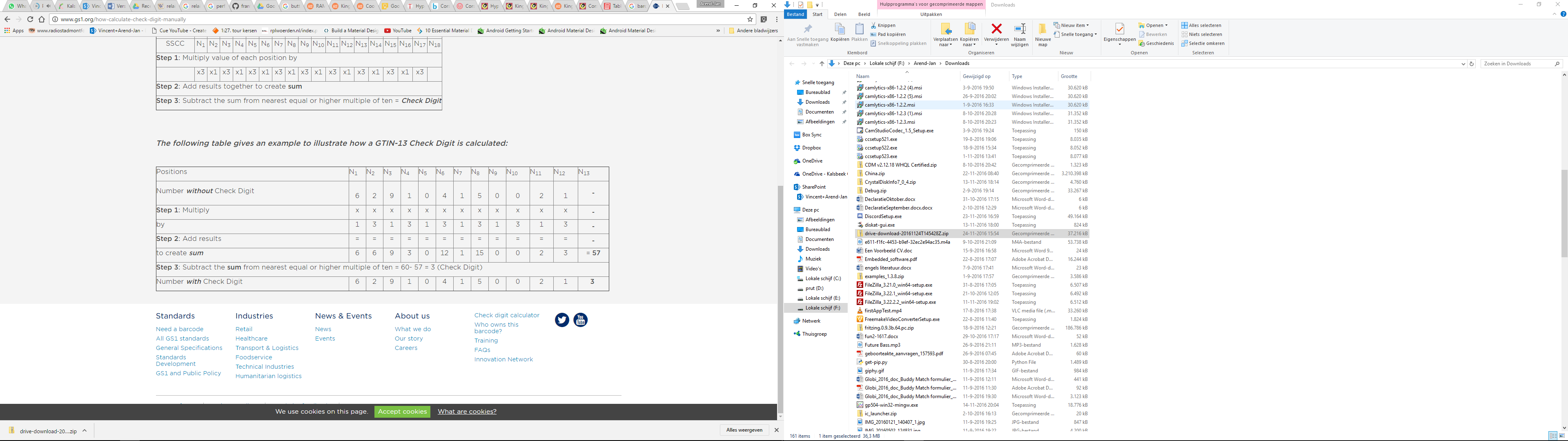


## Barcodescanner

Nu weten we wel welke richting het item opgaat (er in/uit/open), maar we weten nog niet welk item het is. Het is niet handig om te weten hoeveel items, dus zonder naam, je hebt, dus moet er iets komen wat een nummer, id of naam van een item krijgt. We hebben gekeken naar barcodes en RFID. Een barcode is een aantal streepjes met witruimte ertussen, die een scanner uit kan lezen en daar een nummer vanaf kan lezen. RFID maakt gebruik van een ontvanger en een ‘actief’ zendertje, die door elektromagnetische fluctuatie (uit een spoel in de ontvanger) aangaat en door het signaal kort te sluiten berichten naar de ontvanger stuurt. Deze kan dan ook berichten naar het zendertje sturen. RFID wordt veel gebruikt in bijvoorbeeld kledingwinkels en elektronicawinkels, waar de items toch iets duurder zijn dan een pak melk. Deze RFID-zendertjes, ook wel tags genoemd, zijn erg duur ten opzichte van de prijs van normale supermarktitems, ze zijn gemiddeld voor tussen de 20 cent en 1 euro per stuk te koop (10.000 tags). Omdat deze tags zo duur zijn worden ze dus nog niet gebruikt in supermarkten en worden nog de ‘gratis’ barcodes gebruikt. We hebben dus gekozen voor een barcodescanner.

Er zijn verschillende soorten barcodescanners: gewone scanners die alleen maar een lijntje als een laser gebruikt, maar ook omni-directionale scanners. Die gebruiken een draaiende spiegel om barcodes uit elk mogelijke oriëntatie te scannen. We hebben voor eentje uit de laatste categorie gekozen, omdat het zo vele malen makkelijker is om een product te scannen, waardoor alles makkelijker gaat, dus gebruiksvriendelijker is. De barcodescanner identificeert zich als een toetsenbord, zodat alle computers met een usb poort makkelijk gebruik kunnen maken van deze scanner. Achter elke code stuurt hij een \n, dus een enter.

Die code kan in verschillende formaten en lengtes geprint worden. We hebben gekeken naar een aantal verpakkingen en zijn tot de ontdekking gekomen dat het over het algemeen allemaal EAN13 getallen zijn, dat betekent:



Er zijn 12 getallen, de eerste paar horen bij een land, de volgende horen bij een producent, die dan de laatste getallen kan kiezen voor al zijn producten. In totaal kunnen er 12 getallen gekozen worden en het laatste getal is een controlegetal. Hiervoor is een speciaal algoritme, die de barcodescanner weet en ook actief gebruikt, want alleen als de code klopt stuurt hij die code door naar de computer. De oneven getallen(1-geindexeerd) worden opgeteld en de even getallen worden opgeteld en maal 3 gedaan. Hier komt een getal uit, bijvoorbeeld 57. Het controlegetal is dan (10-(som%10))%10, dus eigenlijk hoeveel stapjes naar het volgende tiental. Het controlegetal is 0 als het al een tiental is. Deze formule gebruiken we ook in de app voor het genereren van een barcode. Het enige verschil is dat we een volstrekt willekeurig getal maken. Hier kunnen we later verbeteringen in maken dat het begint met de reeks 20…., want dit is een ‘country’code speciaal voor in winkels, dus tijdelijke items, zoals fruit en kaas.

In het hoofdstuk software>scanner gaan we erop in hoe we de code in ons programma binnenkrijgen en gebruiken.

## Koelkast

De koelkast was het onderdeel dat we als eerste geregeld hebben. Een collega van Arend-Jan, Mark Zimmerman, is vrijwilliger bij de stichting Woerden Helpt, ook wel de Roemeniëmarkt genoemd. Aan mark hebben we rond april gevraagd of ze daar nog een kapotte koelkast hadden en binnen een week lag er al eentje klaar in hun hal. We hebben snel een plek geregeld op school waar de koelkast kon staan. De koelkast kon 212 staan en daar is hij dan ook met hulp van Erik van Hilten, de vader van Arend-Jan, naartoe verhuisd.

Nadat de koelkast op school was geplaatst hebben we een overvalsluiting**,** die Vincent bij de Gamma had gehaald, erop gezet, zodat we er veilig onze spullen in kunnen opslaan. We kwamen er toen ook achter dat de koelkast het wel deed. Deze functie hebben we uiteindelijk niet gebruikt, omdat dat niet goed voor de elektronica is vanwege condens. In de koelkast hebben we een stekkerdoos geplaatst, door in de zijkant van de koelkast een gat te boren en er een andere stekker op te schroeven.



## Thermische Printer

We wilden naast de app voor uitlezen van de koelkast nog een uitvoer van data. Hier hebben we voor een thermische printer gekozen, vanwege de maat van het papier, dit is gewoon bonnetjespapier, dus makkelijk mee te nemen. Een ander voordeel van de printer is dat het een klein ding is, dus niet een hele printer in een koelkastje, maar een kastje van nog geen 20\*10\*10 cm. Ook zijn thermische printers makkelijk aan te sturen via USB (hierover meer in het hoofdstuk software) en het papier is goedkoop, wat weer fijn is voor ons als we aan het testen zijn.

De printer hebben we via eBay, net zoals de barcodescanner, besteld met als enige criteria dat hij niet al te duur moest zijn.

Voor het papier zijn we bij Van Lint in Woerden langsgegaan, die 125 meter papier voor €1,40 te koop heeft, dus net iets meer dan een cent per meter.

De bedoeling is dat de printer teksten, boodschappenlijstjes en barcodes kan printen. Onze printer kan zonder extra software teksten printen.



## Andere hardware

Naast de Raspberry Pi hadden we ook andere hardware nodig om onze software op te maken of te draaien. Op school konden we niet gebruik maken van de computers, omdat ze te weinig opslaggeheugen hebben en veel te langzaam zijn om op te programmeren. Gelukkig kregen we van systeembeheer een laptop om op te werken voor de reüniewebsite. Deze laptop hebben we ook gebruikt voor het maken van code tijdens tussenuren. Omdat de laptop en de Raspberry Pi beide op hetzelfde wifinetwerk zitten konden we via SSH zeer gemakkelijk communiceren met de ‘koelkast’.

Naast deze laptop hadden we ook een eigen laptop meegenomen, dit was een laptop gesponsord door National Instruments. Deze laptop hebben we veel voor erbij gebruikt en voor de connectie met de Raspberry Pi.

De laptop van school was jammer niet zo krachtig dat we daar Android Studio op konden draaien en de Ubuntu-laptop zeker niet, dus hebben we alle code voor de App thuis gemaakt. Het voordeel was dat testen gemakkelijk te doen was.

Naast laptops hadden we ook een server nodig. Arend-Jan was al klant van Strato, dus de keuze was makkelijk gemaakt. We hebben een subdomein gemaakt: pws.svshizzle.com, waar we alles op hebben gezet. Vincent heeft een ftp-account gekregen om toegang te krijgen hiertoe.

## Architectuur

In het onderstaande schema is te zien hoe de architectuur van ons PWS in elkaar steekt.

C:\Users\10261\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Pws Diagram.pngDe Raspberry Pi krijgt input van de 3 Arcade buttons en de barcode scanner. De Raspberry Pi kan ook weer dingen uitprinten via de thermische printer, daarnaast staat de Raspberry Pi in contact met de webserver.

De webserver staat zorgt ervoor dat er handelingen, in opdracht van de Raspberry Pi of de App, gedaan kunnen worden met de database en hier eventueel een response geven aan de App of aan de Raspberry Pi. Omdat we de server niet op de Raspberry Pi zetten is de App altijd te gebruiken, zelfs al staat de koelkast uit. Een ander voordeel aan deze gescheiden opstelling is dat de App overal te gebruiken is, wat niet het geval is als de server op de Raspberry Pi staat. De App is dan niet te gebruiken buiten het netwerk omdat de Raspberry Pi dan niet te bereiken is. (Buiten beschouwing gelaten de mogelijkheid van port forwarding, maar dit is niet mogelijk voor mensen zonder enige kennis of netwerken zoals het schoolnetwerk.)

# Software

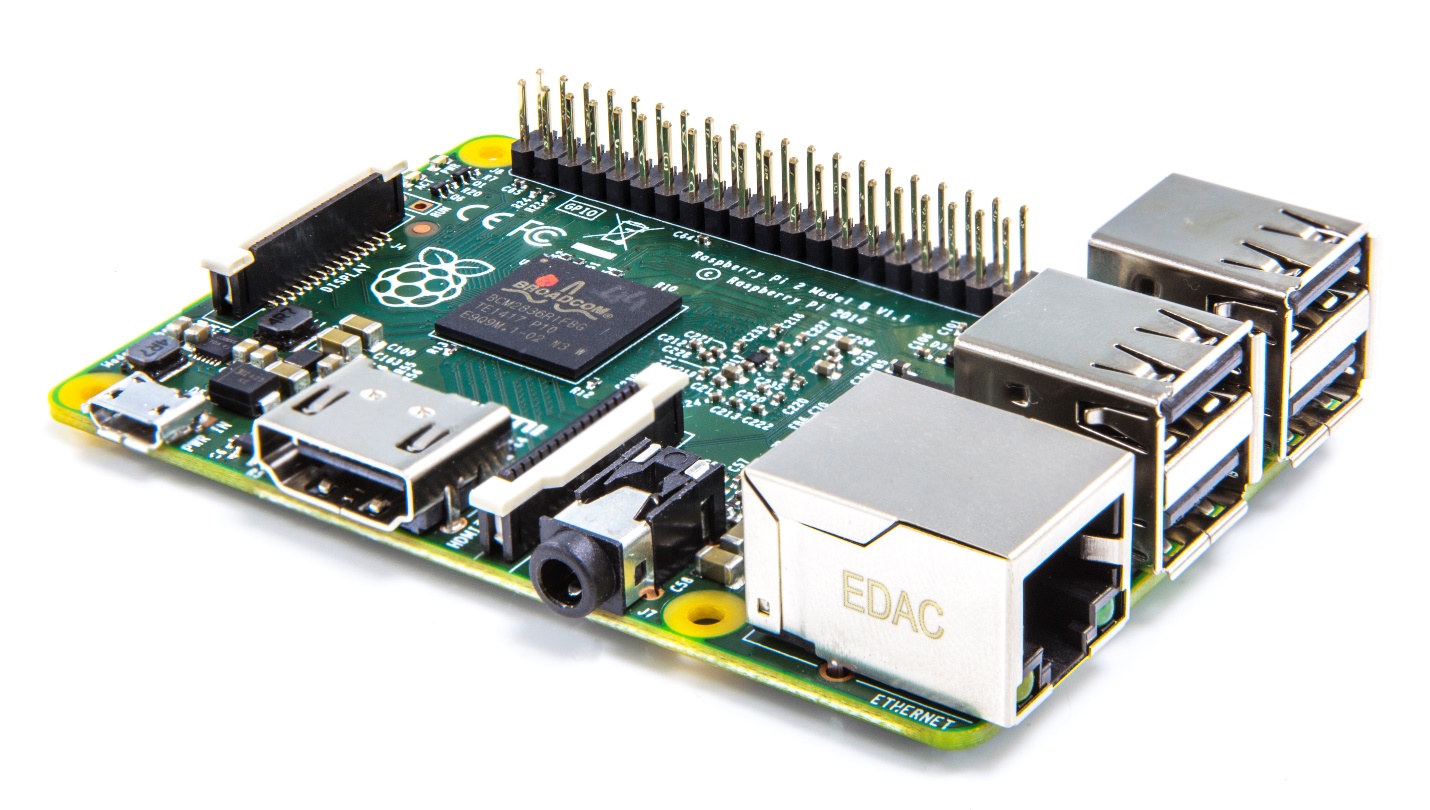
De software is het onderdeel waar we het meeste mee bezig geweest zijn, omdat al die hardware zonder code alleen maar als presse-papier te gebruiken is.

We hebben alle code in het bijgesloten bronnenboek gedaan, zodat het er makkelijk naast gelegd kan worden en dit verslag niet ellenlang (nog langer) wordt door alle code.

## Raspberry Pi

De Raspberry Pi is de computer die we in de koelkast hebben. Hier hebben we een versie van Debian, Raspbian, op draaien, dat is de standaard/aanbevolen Linuxversie. Deze installatie houden we up-to-date door geregeld sudo apt-get update && sudo apt-get upgrade te gebruiken. Voor de zekerheid hebben we ook het wachtwoord aangepast, zodat hij niet in een domoticabotnet wordt ingelijfd en Amerikaanse DNS-servers gaat DDOSSen.

Voor de software van onze koelkast hebben we gekozen voor Python, dit is een zeer gemakkelijke taal om te schrijven en te leren, maar nog overzichtelijker is dan PHP, dus zonder gekke naamconventies en meerdere functies voor hetzelfde. Daarnaast is Python gelijk te draaien op Linux, want de Python interpreter is standaard geïnstalleerd, waardoor we geen bash-scripts hoeven te schrijven voor het installeren van Python of onze code. Achteraf gezien konden we toch beter een bash-script gebruiken om de code te installeren, omdat de meerderheid van de installatiestappen eigenlijk commando’s zijn.



### Overview

De code op de Raspberry Pi heeft als doel om de hele tijd te pollen of er een nieuwe opdracht(job) is en of er een nieuwe barcode gescand is. Dit alles moet zo snel mogelijk en ook zonder elkaar te hinderen. De beide onderdelen mogen dus niet blokkerend zijn, zodat ze niet de thread blokkeren en dat ze dan hun opdracht uitvoeren en dan pas verder gaan. We hebben dus gekozen voor een multithread oplossing: beide delen draaien afzonderlijk van elkaar. Toen kwamen we bij het probleem dat wanneer de Raspberry Pi een opdracht krijgt om iets te printen of te doen, dat hij dan niet doorgaat met pollen van de webserver. Hier hebben we dan ook gekozen voor mulithreading: elke opdracht krijgt zijn eigen thread en sluit die Thread dan af als het klaar is. We kwamen hier ook wel de fout tegen dat we pas een thread moeten laten afsluiten als het onderdeel klaar is, anders hebben we een memory-leak. Dit kwamen we tegen toen we een keer aan het testen waren en het RAM-gebruik steeds meer werd en uiteindelijk het hele programma crashte.

We hebben alles zo netjes mogelijk bij elkaar gestopt, zodat niet onderdelen van de jobs bij de barcode delen komen.

├── includes

│   ├── awakeThread.py

│   ├── \_\_init\_\_.py

│   ├── jobs.py

│   ├── jobsThread.py

│   ├── keyboardThread.py

│   └── threads.py

├── \_\_init\_\_.py

├── IP.php

├── main.py

├── printjobs

│   └── printjobs.txt

├── sendIP.py

├── settings.py

└── start.py

### Install.py

[Pagina 6-7]

Dit stuk code installeert alle software, van onze code, tot python plugins en de cronjob. Het start voornamelijk commando’s, die ook in de terminal ingevoerd kunnen worden, en wacht totdat er eentje klaar is en start dan de volgende. Dat wachten is nodig, omdat sommige onderdelen resources nodig hebben van eerdere commando’s, zoals apt-get (hiermee installeer je een applicatie) en pip (hiermee kan je python-packages installeren).

De plugins die we installeren zijn:

* Curl
* Pip
  + pyBarcode
  + python-crontab

Naast deze makkelijke installatie van onze eigen software is er nog wel een moeilijkere installatie te doen. Dit is de installatie van de printersoftware. Dit komt omdat in elke Linux-versie de printer automatisch als een USB-apparaat herkend wordt, zo hebben we het ook eerst thuis getest op een Ubuntu 16.04 computer, maar na heel veel testen en niet werken en googlen blijkt dat er op Raspbian, de linux-versie speciaal voor de Raspberry Pi, dit onderdeel niet in de kernel zit, dus zouden we de kernel opnieuw moeten compileren, wat niet te doen is. Daarom hebben we een tutorial gevolgd die een thermische printer installeert in het printerprogramma van Linux, waar printers normaal vanuit aangestuurd worden, genaamd CUPS. Deze tutorial is te volgen op scruss**.**

De pseudocode van install.py ziet er ongeveer zo uit:

Check sudo ( zonder sudo recht kunnen we niks installeren )  
Update de pakketbestanden  
Installer curl, pip  
upgrade pip (de versie op de raspbian server is niet de nieuwste versie)  
installeer pyBarcode in pip  
installeer python-crontab in pip  
maak een folder smartfridge, hier komt alle code in  
Download met curl alle code van github in een tarball  
unzip tarball naar een tijdelijke folder  
kopier alleen de code voor de Raspberry Pi naar de smartfridge folder  
kopier start.py van de smartfridge folder naar de huidige folder (folder erboven)  
verwijder de tijdelijke bestanden en de tarball  
verander de toegansrechten van de code naar ‘iedereen mag alles doen’ (chmod 777)  
maak de crontab  
verander indien nodig het instellingenbestand

Een crontab is een opdracht aan Linux die op bepaalde tijdstippen uitgevoerd moeten worden, zoals bijvoorbeeld elke 5 minuten of elke dag. Er is ook een instelling om het elke keer dat de Raspberry Pi opstart de code te laten uitvoeren. Die laatste instelling gebruiken we om bij het opstarten gelijk onze code te laten draaien. De code is:

@reboot python /home/pi/start.py > /home/pi/backup.txt 2>&1

Het gedeelte achter start.py maakt een log bestand aan waar alle tekst die ‘geprint’ is vanuit de code(dus niet vanaf de printer). Deze backup.txt is ervoor om te checken of er iets mis gaat.

### Settings.py

[Pagina 8]

Dit bestandje heeft de instellingen opgeslagen in variabelen, dus deze hoeven dan niet uit een csv of een ander bestand gelezen te worden. Als er ergens een instelling nodig is, dan importeert de code dit bestand en dan zijn de instellingen makkelijk op te halen.

De standaard instellingen zijn:

url **=** "http://pws.svshizzle.com/api/" *#The url of the api*

userId **=** "" *#Your own userId*

interval **=** **5** *#interval of the getjobs function*

Deze instellingen zijn makkelijk aan te passen en ook nog leesbaar.

Install.py kan deze instellingen zelf instellen, door dit bestand te verwijderen en het hele bestand te schrijven met de juiste waardes.

Omdat iedereen zelf zijn eigen api kan hosten op zijn server kan je ook de server instellen van de api. Deze url wordt overal gebruikt, er zijn geen urls hard ingebakken in de code.

Ook kan iedereen zijn eigen userId gebruiken, zodat mensen hun eigen slimme koelkast kunnen hebben op onze servers.

### Start.py

[Pagina 8]

Dit stukje code komt in de home folder (meestal, het kan anders zijn als install.py niet in de home folder is gestart. Hier gebruiken we voor de eenvoud de home folder) van de Raspberry, ook wel aangeduid als **~** , wat normaal op /home/pi/ neerkomt. Dit kleine stukje code start code van main.py.

**from** smartfridge **import** main

main**.**start**()**

### Main.py

[Pagina 8]  
Deze code staat in **~**/smartfridge/ en roept een functie op die de threads starten. Eerst was het de bedoeling dat je dit programma aan moest roepen, maar je kan alleen een bestand importeren als die minimaal 1 folderlaag lager is dan het hoofdprogramma. Bij ons is dat start.py. We moesten dit dus aanpassen omdat keyboardThread.py settings.py opvraagt en dat is in dezelfde folderlaag als main.py, dus werkt het niet. We weten niet precies waarom dit zo is en denken dat dit door een gedachtekronkel van Guido van Rossum, de bedenker, komt.

### Threads.py

[pagina 9]

Deze code start stuk voor stuk het initialiseergedeelte van de afzonderlijke threads. Dit kan niet als een los programma gestart worden.

### keyboardThread.py

[Pagina 9-11]

Nu begint de echte code + het leukste deel!  
KeyboardThread.py is het belangrijkste van alle bestanden, want deze leest de knoppen en barcodescanner uit en stuurt de code door naar de server.

#### Knoppen

De knoppen worden uitgelezen in de functie buttonThread (def staat voor functie …). Deze functie wordt aangeroepen aan het einde van start(). Er wordt een thread gemaakt door middel van de functies

thread **=** Thread**(**target**=**buttonThread**,** args**=())**

thread**.**start**()**

Deze maken een thread aan die de functie buttonThread aanroept zonder parameters. Deze functie heeft een globale variabele (state) die door de barcodescanner thread uitgelezen wordt.

De thread draait steeds dezelfde code, hieronder te zien, maar dan met 3 verschillende variabelen. De waarde van GPIO.input() wordt vergeleken met False omdat de knoppen een pullup weerstand hebben, waardoor ze normaal een True uitlezen en als het signaal naar beneden wordt getrokken een False uitlezen. We hebben een extra time.sleep() toegevoegd zodat niet de lampjes en vooral de relais niet heel erg snel knipperen als alle knopjes tegelijkertijd ingedrukt worden.

**if** GPIO**.**input**(**knopIn**)** **==** False**:**

**print** "knopjeIn"

state **=** "add"

time**.**sleep**(0.2)**

Na deze checks hebben we een stukje code gemaakt die kijkt of de waarde van state is veranderd, en zo ja de vorige lamp uitzet en de volgende aanzet. Dit voorkomt dat er teveel wordt geschakeld, want Raspbian is geen real-time os, waardoor de input-output meer tijd kosten dan bij een echte real-time processor + os, zoals een microcontroller.

**if(**state**!=**statePrev**):**

GPIO**.**output**(**lights**[**statePrev**],** GPIO**.**LOW**)**

GPIO**.**output**(**lights**[**state**],** GPIO**.**HIGH**)**

statePrev **=** state

lights is een array met alle 3 de ledpinnen gekoppeld aan een string (state).

#### Barcode

De barcodescanner identificeert zich als een toetsenbord, dus dat is makkelijk zou je denken, maar dat was het niet helemaal. Omdat de code gestart wordt via SSH of crontab is de barcodescanner niet in die thread. Dit is hetzelfde verschijnsel als je SSHt naar een computer waar iemand mee bezig is en wat diegene typt zie je in je terminal, dat wil je ook niet. Vanwege dit verschijnsel moesten we een soort van keylogger maken die uitleest wat er wordt ‘ingetypt’ door de barcodescanner. Hiervoor hebben we veel op het internet gezocht en uiteindelijk op het Raspberry Pi forum gekomen waar iemand een barcodescanner uit kon lezen met behulp van een python keylogger[[1]](#footnote-2). Deze code hebben we genomen en een beetje aangepast.

De pseudocode van de barcodeThread ziet er zo ongeveer uit:

* Open het ‘toetsenbord’bestand (hier schrijft Linux alle keypresses naartoe)
* Zolang altijd:
  + Haal een barcode op (scancode functie, met bestand variabele)
    - Lees een aantal bytes
    - Voor elke byte:
      * Kijk of het gelijk is aan een return (40)
        + Break, stuur code terug
      * Anders voeg het karakter dat bij die byte hoort aan een string variabele uit een lookup table
  + Start een nieuwe thread met de barcode en de status van de knoppen
    - Deze thread vraagt de pagina itemChange op met de userId, barcode en status op.

We hebben voor een extra thread oplossing gekozen omdat het internet soms zo langzaam is dat er al een volgende code is gescand voordat de vorige is verzonden naar de server. Een verbetering die we hierin zouden kunnen maken is het checken voor errors, zodat zelfs als het internet het niet doet, het opgeslagen wordt en dat ze verzonden worden als het internet het wel weer doet.

Een ander punt om de stabiliteit te verbeteren is dat we maar een keer het ‘toetsenbord’bestand openen. Dit bestand is een soort van buffer: vanaf het punt dat we het openen komt alles in het geheugen te staan, data van voor het openen staat er dan niet meer in. We lezen de buffer stukje voor stukje uit. We openen het bestand maar een keer en gebruiken dat object de hele tijd. We doen dit omdat het kan zijn dat tussen het sluiten en het opnieuw openen van het bestand er een barcode ingevoerd kan worden en dat die dan maar half of helemaal niet uitgelezen wordt. Dit zou voor ongemak bij de gebruiker zorgen. Omdat we het bestand nooit sluiten komen er nooit halve codes binnen en hoeven we dus ook niet de codes te controleren of de codes wel kloppen.

### AwakeThread.py

[Pagina 12]

AwakeThread stuurt elke n (deze kan je instellen in settings.py) seconden een berichtje naar de server (sendActive.php) met de userId.

### JobsThread.py

[Pagina 12-13]

JobsThread is het een-na belangrijkste stukje code dat op de Raspberry Pi staat. Deze thread haalt de jobs op en maakt de threads aan die die jobs afhandelen.

Jobs zijn opdrachten die de computer moet doen, zoals afsluiten, barcode printen, tekst printen, opnieuw opstarten of een lijst uitprinten. Als de code Open-Source is kan iedereen zijn eigen job toevoegen door in een array zijn/haar functie te plaatsen en die functie te definiëren. Op de server checken we niet of het type job wel echt bestaat, dat doet de Raspberry Pi, zodat mensen niet de server hoeven te installeren als ze een extra job willen. Extra jobs zouden muziekjes, natuurlijk van DjVinnii, afspelen of een foto maken kunnen zijn.

De pseudocode van jobsThread ziet er als volgt uit:

* Haal getJobs.php op
* Decodeer de JSON tot een array
* Doorloop de array:
  + Markeer de job gedaan door checkJob te sturen naar de server.
  + Maak een thread aan voor elk item (jobs.py)
* Wacht n seconden(Settings.py)

We hebben gekozen voor deze aanpak, omdat op deze manier de job eerst wordt gemarkeerd en daarna pas gedaan wordt. Dit is beter dan de andere kant op, eerst doen dan markeren, want als de opdracht het afsluiten van de Raspberry Pi is, dan is hij al afgesloten voordat hij hem gemarkeerd heeft. Als dit zou gebeuren zou deze job voor eeuwig bestaan en gedaan worden, wat niet echt het gewenste resultaat is. Het enige nadeel aan deze aanpak is dat een opdracht, bijvoorbeeld een printopdracht, voor het echte uitsturen van het commando afgekapt wordt, omdat er ook een afsluitopdracht is ontvangen. De kans is heel erg klein dat dit zou kunnen gebeuren en we hebben hier ook niets tegen gedaan. Het kan wel zo zijn dat er pas iets uit de printer komt rollen als de Raspberry Pi weer opgestart is na het ontvangen van een afsluitopdracht. Dit komt omdat onze software niet de printer direct aanspreekt, maar via de eerdergenoemde CUPS, die dan de printopdracht pas uitvoert na opstarten. Hier kunnen we niet zo veel aan doen, omdat CUPS de ene keer iets uitprint in 5 seconden en de andere keer er een halve minuut over doet.

Een ander voordeel aan al deze verschillende threads is dat als een thread voor het printen van een barcode crasht vanwege een error en de exception van die error niet wordt opgevangen dat alleen die thread crasht en niet de hele koelkast. Dit is natuurlijk niet helemaal de bedoeling, maar als vangnet wel fijn.

Een verbetering in de code zou kunnen zijn dat hij bij excepties van het markeren niet de job gaat doen, maar overslaat en de volgende keer doet, zodat een job niet meermaals gedaan wordt.

### Jobs.py

[Pagina 14-15]

Jobs.py voert echt de opdrachten uit in tegenstelling tot JobsThread die ze alleen maar ophaalt en afvinkt en een thread maakt.

**def** parseJob**(**jsonX**):***#this will call your function!*

**if** jsonX**[**"Type"**]** **in** options**:**

options**[**jsonX**[**"Type"**]](**jsonX**)**

**else:**

**print** "Not added to the dictionary options"

Deze functie wordt gestart door JobsThread.py als het gemarkeerd is. options is een lijst van de namen(Type) van de opdrachten gekoppeld aan de naam van de functie. In python kan je gebruik maken van lijstcomprehensie, dat betekent dat je een lijst(array) van functies kunt maken en een functie uit die lijst kunt starten door lijst[n](parameters) aan te roepen. Door deze functie hoef je geen switch te gebruiken, wat het gemakkelijker maakt voor andere gebruikers om een functie toe te voegen.

Op het moment hebben we deze opdrachten:

options **=** **{**

"restart" **:** restart**,**

"barcode" **:** qrCode**,**

"list" **:**listFunc**,**

"text" **:** text**,**

"update" **:** update**,**

"shutdown" **:** shutdown

**}**

Deze opdrachten zullen we deels bespreken, omdat sommige niet meer gebruikt worden, zoals update, of hetzelfde zijn als andere functies.

#### Restart/shutdown

Deze functies versturen een commando via de terminal, waardoor de Raspberry Pi opnieuw opstart of afsluit.

**def** restart**(**job**):**

subprocess**.**Popen**([**"sudo"**,** "reboot"**]).wait()** *#Sends sudo restart to the terminal, which will restart the RPi*

Dit stuurt “sudo reboot” naar de terminal. Shutdown stuurt “sudo halt” naar de terminal.

We hebben hier een wait() achter gezet, zodat er geen memory-leak komt en het ram uiteindelijk volloopt. Dit niet zal gebeuren met deze commando’s, maar voor de consistentie doen we het toch.

#### Tekst

Text is de makkelijkste functie voor het printen van tekst, hij maakt er een string van(voor als het een getal is of iets anders geks) en stuurt die string dan naar de functie printCommand.

Deze functie slaat de tekst op in een klein .txt bestandje in de folder printjobs. We doen dit omdat de string een accolade of een dubbele accolade kan bevatten, waardoor het commando “sudo lpr ‘tekststring hierzo ‘ ” niet zou kunnen werken en crashen. Daarnaast is er te volgen wat er mislukt is tijdens het printen, bijvoorbeeld als de Raspberry Pi opeens uitvalt of afsluit, dan blijft het bestand bestaan en dan kunnen ze nog een keer geprint worden. Normaal worden de bestanden verwijderd na het printen, daarom hebben we ook de .wait() functie op het commando gezet, zodat het bestand nog niet is verwijderd als het nog niet is geprint. We verwijderen het bestand als het succesvol is geprint, omdat anders de ‘harde schijf’ van de Raspberry Pi langzaam volloopt. In de database komt ook altijd een log van wat er is geprint, dus die is altijd te checken.

De naam van het tekstbestandje is altijd de tijd op de milliseconden precies van wanneer de opdracht wordt gestart. Als er meerdere opdrachten tegelijkertijd binnenkomen, dan worden ze toch als andere bestanden opgeslagen, omdat ze altijd iets verschillen qua opstarttijd.

lpr is het commando om via de terminal teksten of plaatjes uit te printen. Ze worden dan via het CUPS systeem geprint.

**def** printCommand**(**text**):** *#This is the function that is called with a string containing a text that should be printed.*

filename **=** os**.**path**.**dirname**(**os**.**path**.**realpath**(**\_\_file\_\_**))** **+** "/../printjobs/" **+** getTimeStamp**()** **+** ".txt"

filex **=** open**(**filename**,** 'w+'**)**

filex**.**write**(**text**)**

filex**.**close**()**

command **=** "sudo lpr '"**+** filename **+** "'"

**print** command

subprocess**.**Popen**(**command**,** shell**=**True**).**wait**()**

os**.**remove**(**filename**)**

#### List

Er komt een array(json) aan items binnen met de naam van het product of wat er dan ook op een regel moet in het object “Title”. Deze objecten (eigenlijk alleen maar stringen) worden samen met een streepje en een nieuwregel(\n) aan elkaar geplakt. Deze grote lijststring wordt dan ook naar printCommand gestuurd en uitgeprint.

**def** listFunc**(**job**):** *#Prints the list of the items you should buy!!!*

output **=** ""

liste **=** json**.**loads**(**job**[**"List"**])**

**for** item **in** liste**:** *#iterates through the list*

output **+=** "-" **+** item**[**"Title"**]** **+** "\n"

printCommand**(**output**)**

#### Barcode

De barcodefunctie, die we per ongeluk qrCode hebben genoemd, was de moeilijkste functie om te implementeren. Deze functie maakt eerst een plaatje en stuurt dan een commando met de bestandsnaam van dat plaatje. Na het printen van de barcode wordt het bestand niet verwijdert, omdat deze toch overschreven wordt zodra er een nieuwe barcode wordt geprint. Hierbij nemen we dus aan dat er niet twee codes tegelijk binnenkomen. Hier zouden we dus ook gebruik kunnen maken van de tijd en daar bestandsnamen op aan kunnen passen.

**def** qrCode**(**job**):**

**import** barcode

**from** barcode.writer **import** ImageWriter

EAN **=** barcode**.**get\_barcode\_class**(**'EAN'**)**

ean **=** EAN**(**job**[**"Code"**],** writer**=**ImageWriter**())**

ean**.**save**(** os**.**path**.**dirname**(**os**.**path**.**realpath**(**\_\_file\_\_**))** **+** "/../printjobs/ean13\_barcode"**)**

command **=** "lpr -o fit-to-page " **+** os**.**path**.**dirname**(**os**.**path**.**realpath**(**\_\_file\_\_**))** **+** "/../printjobs/ean13\_barcode.png"

subprocess**.**Popen**(**command**,** shell**=**True**).**wait**()**

*#now print this thing!*

## Server

Zoals al te zien was in de architectuur van ons hele PWS is te zien dat de webserver centraal staat in het geheel. Op de webserver staan allerlei PHP-scripts. Deze scripts voeren eigenlijk alle acties uit, die van belang zijn voor onze slimme koelkast. De app of de Raspberry Pi sturen een reeks parameters naar een bepaald PHP-script. We hebben ervoor gekozen om zo veel mogelijk voor iedere actie een apart script te maken. Het voordeel hiervan is dat alles een stuk overzichtelijker is, maar ook wanneer er een fout zit in een script zit werkt er maar een klein stukje niet in plaats van alle code. Ook staat er op de webserver een database, hierin komt alle informatie die we later weer moeten gebruiken.

We hebben voor PHP gekozen omdat dit de enige programmeertaal/scripttaal is die onze server/host ondersteunt. Hierdoor hoefden we geen andere host uit te zoeken of onze eigen server te draaien. PHP is het meest gebruikt en draait dus ook op de meeste servers, waardoor het voor de mensen die ons project willen aanpassen ook makkelijk is om te testen en in productie te zetten.

De filetree van de server ziet er als volgt uit:

├── api

│   ├── defaultOutput

│   │   └── getjobs.php

│   ├── include

│   │   ├── checkJobId.php

│   │   ├── checkUserId.php

│   │   ├── delClosed.php

│   │   ├── delItem.php

│   │   ├── delOpen.php

│   │   ├── getItem.php

│   │   ├── log.php

│   │   ├── newItem.php

│   │   └── openItem.php

│   ├── addJob.php

│   ├── ajlogtest.php

│   ├── contains.php

│   ├── createItem.php

│   ├── getActive.php

│   ├── getJobs.php

│   ├── getLog.php

│   ├── itemChange.php

│   ├── markJob.php

│   ├── resetLog.php

│   ├── sendActive.php

│   ├── serverCheck.php

│   ├── titleChange.php

│   └── userIdCheck.php

├── css

│   ├── bootstrap.css

│   ├── bootstrap.min.css

│   └── one-page-wonder.css

├── custom

│   ├── generate.php

│   ├── printtext.php

│   └── readme.txt

├── fonts

│   ├── glyphicons-halflings-regular.eot

│   ├── glyphicons-halflings-regular.svg

│   ├── glyphicons-halflings-regular.ttf

│   ├── glyphicons-halflings-regular.woff

│   └── glyphicons-halflings-regular.woff2

├── img

│   ├── banner.png

│   ├── plaatje1.png

│   └── plaatje2.png

├── js

│   ├── bootstrap.js

│   ├── bootstrap.min.js

│   └── jquery.js

├── php

│   ├── db.php

│   ├── mailtest.php

│   ├── settings.php

│   └── start.php

├── generateUserId.php

├── index.php

└── text.txt

### Database

Voordat we de database hebben gebouwd, hebben we eerst een ERD (Entity Relationship Diagram) gemaakt. Een ERD is een diagram om het concept voor een database al duidelijk te maken. In een ERD kan je gemakkelijk zien wat er in een tabel moet komen, maar ook relaties tussen bepaalde tabellen zien. De structuur van de database ziet er als volgt uit:



In dit ERD is goed te zien dat in iedere tabel de userId weer terugkomt. Hiermee kunnen er meerdere mensen gebruik maken van dezelfde database, ieder record is zo gekoppeld aan een persoon. Doordat we voor alles een userId gebruiken, kunnen we ook makkelijk gebruiker gerelateerde problemen opsporen.

Ook is te zien dat wij hebben gekozen om twee log tabellen te maken: één tabel voor de normale log en één tabel voor de errorlog.

Omdat we gebruikmaken van de barcode om producten te identificeren is het makkelijk om later uit te breiden naar het op afstand bestellen, want zo worden ze ook bij de Jumbo uit elkaar gehouden. Het enige wat we waarschijnlijk dan moeten toevoegen is een kolom inde Products tabel waar het aantal te bestellen artikelen in komt, in plaats van het gebruiken van een algoritme om de Log uit te lezen. Het koppelen van een receptendatabase of website is daarentegen moeilijker, omdat de naam die we van de Jumbowebsite halen, meestal te lang is of met te veel informatie. Voor elk product zou er dan een trefwoord moeten komen, zodat bijvoorbeeld “Echte Enkhuizer Jodekoeken Choco 18 Koeken 363g” aangepast wordt naar “Jodenkoek”. Met deze trefwoorden zou een receptendatabase wel doorzoekbaar worden. Het enige nadeel hieraan is dat er voor elk product er een trefwoord gemaakt moet worden en deze moeten dan door de gebruiker zelf gemaakt worden, omdat die trefwoorden niet op de Jumbowebsite staan en omdat er ook zelf toegevoegde producten in de koelkast kunnen zitten.

### PHP

Zoals dus al eerder gezegd staan er op de webserver een groot aantal PHP-scripts. Veel van deze scripts zijn eigenlijk voor een groot deel op dezelfde manier opgebouwd, maar we gaan een aantal interessante stukjes script uitleggen.

##### Statements

Het belangrijkste van de PHP is om data in de database te zetten of om juist data uit de database op te vragen. Hiervoor hebben wij ervoor gekozen om gebruik te maken van PDO (PHP Database Objects) in plaats van mySQL of mySQLi. Eigenlijk was de keuze alleen tussen mySQLi en PDO. mySL moet je eigenlijk niet meer gebruiken in verband met de veiligheid van heel de webserver/database en bovendien wordt mySQL niet meer onderhouden. Daarnaast hebben we voor PDO gekozen omdat PDO meer soorten databases ondersteund en mySQL ondersteund alleen mySQL databases.

Wij vragen veel op uit onze database, maar zetten een ook veel in. Een voorbeeld van het opvragen van informatie uit de database ziet er als volgt uit:

$stmt **=** $conn**->**prepare**(**'SELECT \* FROM products WHERE (closed > 0 OR open > 0) AND userId = ?'**);**

$stmt**->**execute**(array(**$userId**));**

Dit is een stukje code uit contains.php. Dit PHP-script vraagt op wat er in de koelkast zit. Dit specifieke statement vraagt alles uit op uit de products tabel waar het aantal geopende producten groter is dan nul of het aantal gesloten producten is groter dan nul. Echter wordt niet alles opgevraagd, want de userId moet overeen komen met de gegeven userId.

Eerst wordt er een variabele gemaakt met de statement zelf. Hier hoeven nog geen variabelen in gevuld te worden, zoals hier het userId. Daarna wordt het statement uitgevoerd met de variabelen.

In dit geval is er maar één variabele in het statement, maar dit kunnen er heel veel zijn. Omdat er heel veel variabelen kunnen zijn, gebruiken we in de execute een array met variabelen die gebruikt worden.

Een voorbeeld van informatie die in de database wordt gezet is bij de addJob.php:

$stmt **=** $conn**->**prepare**(**"INSERT INTO jobs (userId, type, barcode, text, list, status) VALUES (?,?,?,?,?,?)"**);**

$stmt**->**execute**(array(**$userId**,** $type**,** $barcode**,** $text**,** $list**,** "new"**));**

Ook hier wordt eerst een variabele voor het statement gemaakt. In dit geval worden userId, type, barcode, text, list en status in de jobs tabel geplaatst. Hier is goed te zien dat een array voor de variabelen makkelijk is. Als we bij het uitvoeren van het statement geen array gebruiken zal het stukje code er als volgt uit zien:

$stmt **=** $conn**->**prepare**(**"INSERT INTO jobs (userId, type, barcode, text, list, status) VALUES (?,?,?,?,?,?)"**);**

$stmt **->** bind\_param**(**$userId**,** $type**,** $barcode**,** $text**,** $list**,** "new"**);**

$stmt**->**execute**();**

Dit scheelt voor één statement maar één regel code, maar in het geheel hebben we veel statements die uitgevoerd moeten worden en loopt dit dus heel erg op.

##### newItem

[Pagina 40-41]

newItem.php is het script dat word aangeroepen wanneer er een gesloten product in de koelkast wordt gezet. Omdat wij geen database van een supermarkt hebben gekregen hebben wij ervoor gekozen om een zelf opbouwende database te maken, dit komt terug in newItem.php. Dit script moet eerst kijken of het product wat in de koelkast gaat, al in de koelkast staat om zo dubbele rijen te voorkomen in de database.

Eerst maken we dus gebruik van een check statement:

$checkstmt **=** $conn**->**prepare**(**"SELECT barcode FROM products WHERE barcode = ? AND userId = ?"**);**

$checkstmt**->**execute**(array(**$code**,** $userId**));**

Hier wordt de barcode opgevraagd uit de tabel products, wel moeten de barcode en userId overeenkomen met de invoer.

Dan moet er gecontroleerd worden of er al een product bestaat voor deze gebruiker en met de ingevoerde barcode. Dit doen we met een if-else statement.

**if(**$checkstmt**->**rowCount**()** **>** **0){**

*//Doe +1 bij aantal van product*

$upstmt **=** $conn**->**prepare**(**"UPDATE products SET closed = closed + 1 WHERE barcode = ? AND userID = ?"**);**

$upstmt**->**execute**(array(**$code**,** $userId**));**

**}** **else** **{**

*//Voeg product toe*

$desc **=** getTags**(**$code**);**

$addstmt **=** $conn**->**prepare**(**"INSERT INTO products (barcode, closed, description, userId) VALUES (?,1,?,?)"**);**

$addstmt**->**execute**(array(**$code**,**$desc**,**$userId**));**

**}**

Voor de leesbaarheid zijn hier alle try-catch statements er uitgehaald, deze staan wel in de code.

Op basis van het aantal rijen uit het checkstatement wordt bepaald welk statement hier gebruikt moet worden. Al is het aantal rijen groter dan nul, wat betekent dat het product al bestaat, dan wordt alleen het aantal gesloten producten verhoogd met één. Is het aantal rijen gelijk aan, of kleiner dan 0 dan moet er een nieuwe rij gemaakt worden.

##### openItem

[Pagina 41]

openItem.php maakt de hoeveelheid open producten één hoger. Wat wel belangrijk is om te weten is dat we er met openItem ervan uit zijn gegaan dat mensen er eerst een product uit halen om er later weer (open) in te doen, of juist niet want het product is dan op. Eerst wordt het product uit de koelkast gehaald door op de rode knop te drukken. Als de producten gesloten terug gaan kan er op de groene knop gedrukt worden en als de producten geopend terug in de koelkast gaan moet er op de blauwe knop gedrukt worden.

##### delItem

[Pagina 37-38]

Om een product uit de koelkast te halen hebben we het script delItem.php. Een belangrijk punt uit dit script is het feit dat er ook gecontroleerd moet worden of er een product open is. Met onze logica gaan eerst alle open producten uit de koelkast en daarna pas de gesloten producten.

Als eerst wordt het aantal open en gesloten producten uit de database opgevraagd:

$countstmt **=** $conn**->**prepare**(**"SELECT closed, open FROM products WHERE barcode = ? AND userId = ?"**);**

$countstmt**->**execute**(array(**$code**,** $userId**));**

Deze waardes worden daarna gebruikt om te kijken of er open producten zijn en zo niet of er dan wel gesloten producten zijn. Op basis hiervan wordt de voorraad van de open producten of gesloten producten verminderd met één. Dit ziet er als volgt uit:

$ding **=** $countstmt**->**fetch**();**

$closed **=** $ding**[**'closed'**];**

$open **=** $ding**[**'open'**];**

**if** **(**$open **>** **0){**

*//Doe -1 bij open product*

$delstmt **=** $conn**->**prepare**(**"UPDATE products SET open = open - 1 WHERE barcode = ? AND userId = ?"**);**

**}** **else** **{**

**if(**$closed **>** **0)** **{**

*//Doe -1 bij closed product*

$delstmt **=** $conn**->**prepare**(**"UPDATE products SET closed = closed - 1 WHERE barcode = ? AND userId = ?"**);**

**}** **else** **{**

errorLogging**(**basename**(**$\_SERVER**[**'PHP\_SELF'**]),** $params**,** $userId**,** "Closed = 0 or < 0"**);**

**die();**

**}**

**}**

**try** **{**

$delstmt**->**execute**(array(**$code**,** $userId**));**

**}**

**catch** **(**PDOException $e**){**

errorLogging**(**basename**(**$\_SERVER**[**'PHP\_SELF'**]),** $params**,** $userId**,** $e**);**

**die();**

**}**

Naast delItem.php hebben we ook nog delOpen.php en delClosed.php. Deze twee scripts zijn speciaal voor de app.

##### getItem

[Pagina 39]

We willen natuurlijk weten welke producten er in de koelkast zitten. Hiervoor hebben wij eerst geprobeerd om een database van een supermarkt te krijgen. Wij hebben aan de Jumbo, Albert Heijn, Aldi en de Lidl gevraagd of wij een database konden krijgen waar op zijn minst de barcodes met bijbehorende productnamen in staan, zodat wij dit in onze database kunnen gebruiken in de products tabel. Helaas hebben wij deze niet gekregen en hebben wij besloten om dan maar de Jumbo website te gebruiken.

In de PHP bestaat al een functie genaamd get\_meta\_tags. Dit is een functie die de meta tags opvraagt van een site. Een meta tag is een stukje informatie over de informatie op de site. Het stukje code wat wij hebben gebruikt voor het opvragen van de meta tag genaamd description ziet er als volgt uit:

**function** getTags**(**$code**){**

$tags **=** get\_meta\_tags**(**'http://www.jumbo.com/zoeken?SearchTerm=' **.** $code**);**

$desc **=** $tags**[**'description'**];**

**return** $desc**;**

**}**

Hier worden eerst alle meta tags van de Jumbo site gehaald, die volgt uit de zoekfunctie met de barcode. Van alle tags maken we alleen een variabele voor de omschrijving van het product. Deze omschrijving wordt dan teruggestuurd om verder te gebruiken.

##### Globals

Alle PHP-scripts die een actie uitvoeren dat direct te maken heeft met de voorraad, maken gebruik van een globale variabele. Deze scripts zijn: delClosed.php, delItem.php, delOpen.php, newItem.php en openItem.php. Dit heeft te maken met het loggen van de acties, in combinatie met het maken van een boodschappenlijstje.

Aan het eind van deze drie scripts staat de volgende regel:

$GLOBALS**[**'doLog'**]** **=** "y"**;**

Op deze manier krijgt de globale variabele doLog de waarde y, maar als het script eerder wordt gestopt omdat er een fout voordoet blijft deze n. In itemChange.php wordt vervolgens pas iets gelogd als de globale waarde doLog y is.

Wij ontdekten een bug dat er wel een actie gelogd werd wanneer je een product uit de koelkast haalt, maar de voorraad waarde op nul stond. Op deze manier dacht de app dat er dan meer producten gehaald moeten worden om weer op de maximale voorraad te komen. (Let op! Wij gebruiken niet de voorraad voor het boodschappenlijstje, maar de app bepaald doormiddel van de log tabel in de database wat de hoogste voorraad was.) Met de globale variabele is dit probleem opgelost.

##### Generate UserId

[Pagina 42 - 43]

Wij willen dat ons project gebruikt kan worden door meerdere mensen. Hiervoor moet er in de database onderscheid zijn tussen de verschillende gebruikers. Om dit te doen kunnen mensen heel eenvoudig een eigen userId aanmaken. Op de webpagina generateUserId.php is een simpel invoerveld voor een emailadres en een knop. Wanneer er op de knop wordt gedrukt, nadat er een emailadres is ingevoerd, word het PHP-script generate.php opgevraagd. Dit script genereert een userId en stuurt vervolgens een mailtje.

Het genereren van een userId ziet er als volgt uit:

$userId **=** uniqid**();**

Hier wordt de variabele userId gemaakt doormiddel van de standaard functie uniqid. Deze functie genereert een unieke string gebaseerd op de microtijd, de huidige servertijd in microseconden.

##### Active

[Pagina 22-23 en 31-32]

Wij willen ook graag kunnen zien wanneer de Raspberry Pi voor het laatst online is geweest, zodat we in de app kunnen zien of de koelkast online of offline is. Hiervoor hebben wij twee PHP-scripts: sendActive.php en getActive.php. In het kort gezegd zet sendActive.php de tijd van het moment dat het script wordt opgevraagd in de database. getActive.php leest deze waarde weer uit.

De Raspbery Pi vraagt af en toe sendActive.php op, zo wordt de ActiveTime iedere keer geüpdatet. Het script bepaalt de tijd door middel van de functie date. Dit ziet er als volgt uit:

$tijd **=** date**(**"Y-m-d H:i:s"**);**

Door middel van alles wat binnen de aanhalingstekens staat bepalen wij het format van de tijd. De Y geeft het jaar in vier getallen. De m geeft de maand in twee getallen. De d geeft de dag van de maand in twee getallen. De H geeft het uur in 24-uurs format en met een eventuele 0 aan het begin. De i geeft het aantal minuten met eventuele 0 aan het begin. De s geeft het aantal seconden met eventuele 0 aan het begin. De – en : geven het teken aan dat tussen de getallen moet komen. Wanneer er een tijd opgevraagd wordt door het script komt dit er dus als volgt uit te zien: “2016-12-07 14:30:01”.

De app vraagt getActive.php op en krijgt de ActiveTime terug. Met deze tijd kan de app bepalen of de Raspberry Pi uit staat of juist aan. Als de laatste activeTime langer geleden is dan een **ingestelde** **tijd**, dan staat de Raspberry Pi uit. Op deze manier wordt er niet gelijk gedetecteerd dat de Raspberry uit staat, maar dit duurt even. De Raspberry kan namelijk niet de hele tijd dit script uit blijven voeren, dan is er geen ruimte meer over om andere python scripts uit te voeren op de Raspberry Pi.

##### titleChange

[Pagina 32-33]

Soms scant iemand een product wat ze niet bij de Jumbo hebben. Op dat moment vindt de Jumbo website dat product niet en komt er als titel van dat product Jumbo Groceries in de database, wat niet handig is als je verschillende producten hebt die deze titel krijgen. Daarom moet ook de titel van een product in de app aangepast worden. Hiervoor hebben wij het script titleChange.php gemaakt.

Het belangrijkste uit dit script is het statement om de titel te updaten. Deze ziet er als volgt uit:

$open2stmt **=** $conn**->**prepare**(**"UPDATE products SET description = ? WHERE barcode = ? AND userId = ?"**);**

$open2stmt**->**execute**(array(**$title**,** $code**,** $userId**));**

Hier wordt de description in de tabel products geüpdatet waar de barcode en userId over een komen met de invoer. Op deze manier wordt Jumbo Groceries dus gewijzigd naar iets wat de gebruiker zelf invoert.

##### Error logging

[Pagina 39-34]

Soms kan een query niet goed uitgevoerd worden. Of een persoon vergeet een belangrijke parameter mee te sturen. Daarom hebben wij bepaald dat zulke fouten allemaal in een aparte tabel gezet worden, dit is makkelijker voor het oplossen van problemen. Zowel de meegestuurde parameters in de PHP als query’s worden gecontroleerd.

De query’s worden gecontroleerd door een try-catch. Dit wil zeggen dat de PHP probeert een query uit te voeren en als dit niet lukt vangt hij de error en wordt het hele script gestopt.

De parameters in de JSON worden allemaal gecontroleerd door middel van de isset functie en de empty functie. Op deze manier wordt er gekeken of de parameter niet leeg is, maar ook dat dit parameter ook wel wordt meegestuurd.

Voor het daadwerkelijk loggen van de fout hebben we het PHP-script log gemaakt met de functie errorLogging (in de log.php staat ook de normale log functie).

**function** errorLogging**(**$script**,** $params**,** $userId**,** $e**){**

**require\_once(**dirname**(\_\_FILE\_\_).**"/../../php/start.php"**);**

*//Functie parameteres doorgooien naar JSON voor in DB*

$params **=** json\_encode**(array(**'script' **=>** $script**,** 'params' **=>** $params**,** '$userId' **=>** $userId**));**

*//Functie parameteres doorgooien naar JSON voor in error return*

$error **=** json\_encode**(array(**'script' **=>** $script**,** 'params' **=>** $params**,** 'userId' **=>** $userId**,** 'error' **=>** $e**));**

$conn **=** db**();**

**try{**

$logstmt **=** $conn**->**prepare**(**"INSERT INTO errorLogging (time, script, params, userId, error) VALUES (?,?,?,?,?)"**);**

$logstmt**->**execute**(array(**date**(**'Y-m-d H:i:s'**),**$script**,**$params**,**$userId**,**$e**));**

**}**

*//Hier niet errorLogging want anders oneindige loop*

**catch** **(**PDOException $e**){**

**die** **(**"An error occured during error logging:S $e"**);**

**}**

$arr **=** **array(**'error' **=>** $e**);**

**echo** json\_encode**(**$arr**);**

**}**

De code van deze functie is al redelijk uitgelegd binnen de PHP zelf door middel van comments. Wanneer de functie wordt aangeroepen ziet het er als volgt uit:

errorLogging**(**basename**(**$\_SERVER**[**'PHP\_SELF'**]),** $params**,** $userId**,** $e**);**

Wanneer de errorLogging functie wordt aangeroepen door een script worden er een aantal parameters mee gegeven.

Als eerste wordt de naam van het script waar de fout in voor doet mee gegeven. De naam van dit script kan worden gegeven door basename($\_SERVER[‘PHP\_SELF’]).

Daarna worden ook alle parameters van het script mee gegeven, zo kunnen we op een later tijdstip zien welke parameters zorgen voor een probleem.

Ook wordt het userId meegegeven. Dit doen we omdat we dan kunnen zien wie er voor een probleem veroorzaakt. Dit probleem kan een normaal probleem zijn, maar ook iemand die probeert een lek te vinden in onze slimme koelkast.

Als laatste wordt ook de foutmelding mee gegeven. Door terug te kijken wat de foutmelding is, kunnen we gerichter tussen de parameters kijken om te zien wat het probleem is. Het kan ook zijn dat de fout komt omdat bijvoorbeeld de database uit de lucht is, dit kunnen wij op deze manier ook gemakkelijk zien.

## App

De app is het onderdeel om de koelkast mee uit te lezen op een Androidtelefoon. We kunnen nog een website maken om hem uit te lezen op een computer of een iPhone. We hebben alleen een app gemaakt voor Android, omdat we alleen daarvoor kunnen ontwikkelen. We hebben namelijk (nog) geen Mac. Daarnaast hebben de meeste mensen een Android telefoon, vooral in onze ‘doelgroep’, de hackers en programmeurs. We hebben de hele app in native Java geschreven, dus er zit niet nog een laag tussen waardoor de app langzamer wordt. In dit hoofdstuk zullen we eerst wat foto’s laten zien van alle mogelijke schermen met uitleg wat ze doen. Daarna gaan we in op de speciale onderdelen en truckjes die we hebben toegepast om de app zo te maken dat hij ook nog aanpasbaar is voor iemand anders. De minimale versie van Android is 5.0(SDK 21).

We hebben onze app ook in de Play Store gezet, zodat de drempel voor mensen lager is om ons project over te nemen en ermee te gaan spelen/werken.

Alle teksten zijn in het Engels, omdat we het hele project open source maken moet het dus ook voor iedereen te gebruiken zijn. Het kost niet heel erg veel moeite om een andere taal toe te voegen.

### Foto’s + uitleg

Alle schermafbeeldingen zijn genomen op een Nexus 6P met Androidversie 6.1. We hebben ook met deze telefoon getest en ontwikkeld, dus hierop werkt het zeker. Nieuwere/andere Androidversies hadden we niet tot onze beschikking, maar het moet werken op andere telefoons.

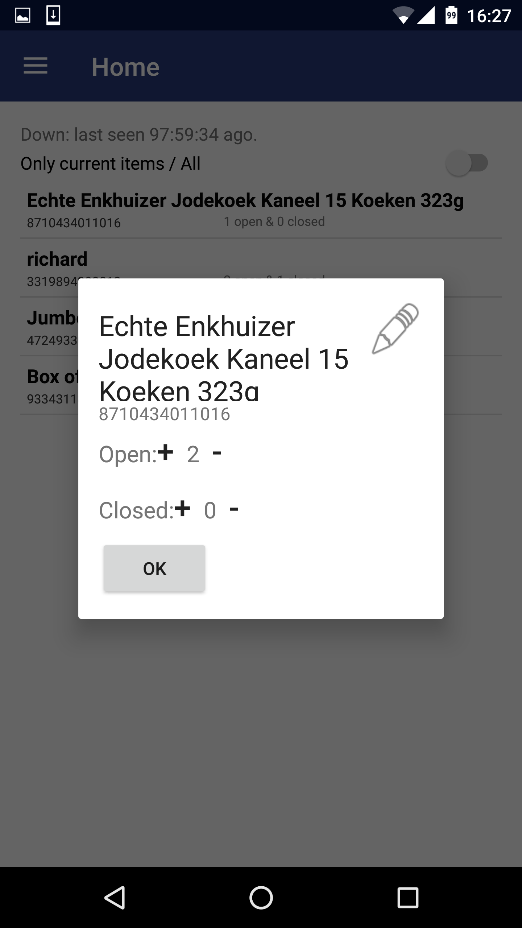
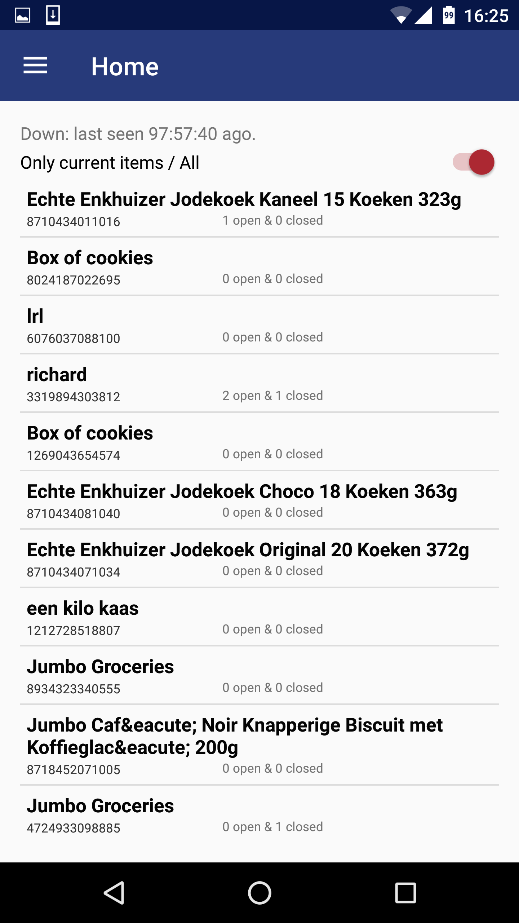
#### Login

Als de gebruiker de app voor het eerst opstart wordt deze direct doorgestuurd naar de loginpagina. Hier kunnen ze hun userId en mogelijk een eigen url van hun server invullen. De url wordt dan gecontroleerd en daarna ook de userId. Als ze het goed hebben ingevuld worden ze doorgestuurd naar het hoofdscherm. Eenmaal ingelogd krijgen ze de loginpagina nooit meer te zien.

Deze variabelen worden opgeslagen en altijd uitgelezen als er iets van de server opgehaald moet worden.

#### Home

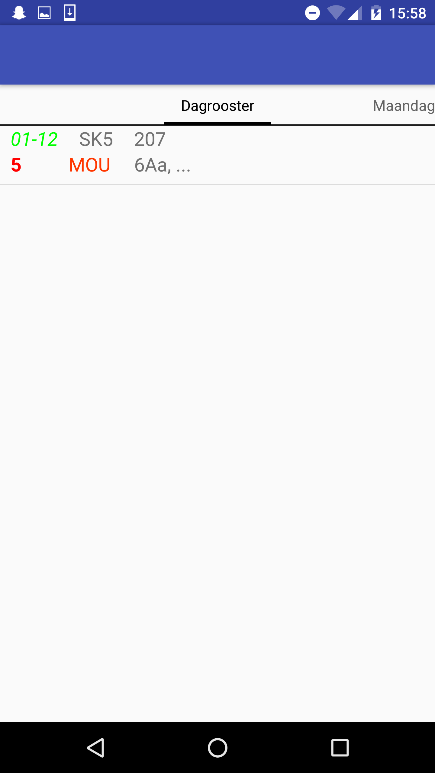
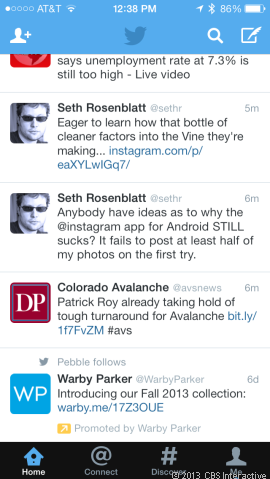
Het homescherm is het eerste scherm dat je ziet na het inloggen of het als je de app opstart en je bent al ingelogd. Hier kan je gelijk zien wat je in je koelkast hebt en of je koelkast nog online is. De koelkast wordt online gezien als hij de afgelopen 10 minuten nog een berichtje heeft gestuurd naar de server. Alleen items waar minimaal 1 van in de koelkast staat, waarbij het niet uitmaakt of het een gesloten of een open item is, krijg je te zien. We hebben daarom ook een swipe toegevoegd, waardoor je alle items kan zien, ook de items die niet in de koelkast zitten. We hebben dit toegevoegd zodat je items kan toevoegen die volgens de server niet in de koelkast zitten, anders kon je niet op die items klikken. Als je op een item klikt krijg je een schermoverlay te zien met de details van dat product en kan je de titel alsmede de open en gesloten aantallen aanpassen.



#### Menu

Om tussen de verschillende pagina’s te gaan hebben we voor een menu gekozen, om precies te zijn een hamburgermenu. We hebben voor dit soort menu gekozen omdat we nog niet wisten hoeveel pagina’s we zouden maken of dat er nog komen, dus konden we niet een menu zoals in de Facebook app gebruiken of de Twitter app, waar de hoofdmenu’s boven respectievelijk onder zitten. We hadden ook voor één pagina kunnen kiezen met ‘tabbladen’, zoals in de Kalsbeek Rooster app, maar we dachten dat het dan overzichtelijk zou worden, vanwege nog een balk en omdat de pagina’s geen onderling verband hebben.

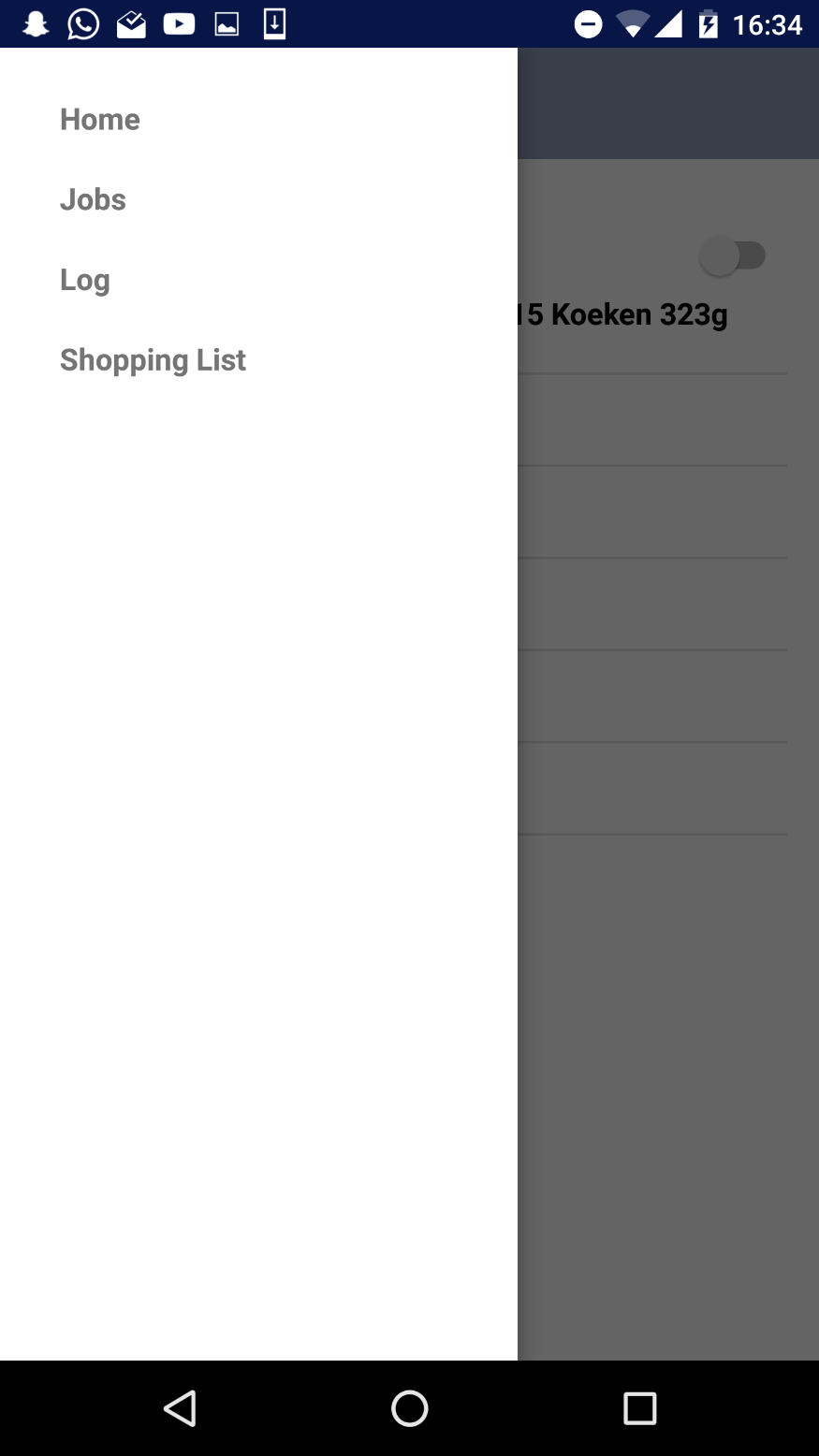




Tabbladen aan de bovenkant in de Kalsbeek Rooster app

Twitter's menu aan de onderkant met 4 plaatjes

Facebook menu aan de bovenkant met de 4 plaatjes



Het menu van onze app

#### Jobs

Op de jobspagina kan je een opdracht naar de koelkast sturen. We hebben hier alle mogelijke opties voor jobs die we zelf hebben bedacht, behalve de lijst. Deze functie kunnen we later nog toevoegen, maar ergens anders hebben we deze functionaliteit al. Alleen kan je daar niet de tekst aanpassen. Als je op een van de vier knoppen drukt komt er een dialoog te zien met een tekst van dat hij aan het verzenden is. Deze dialogen zijn niet te annuleren, iets wat we nog kunnen toevoegen**.** Als het gelukt is, of juist niet, komt er een kleine melding onderin het scherm zichtbaar en weet de gebruiker of het gelukt is. Als je op print barcode klikt wordt er een willekeurige barcode geprint en er wordt een item toegevoegd aan de database met de ingevulde naam (geen aantallen).

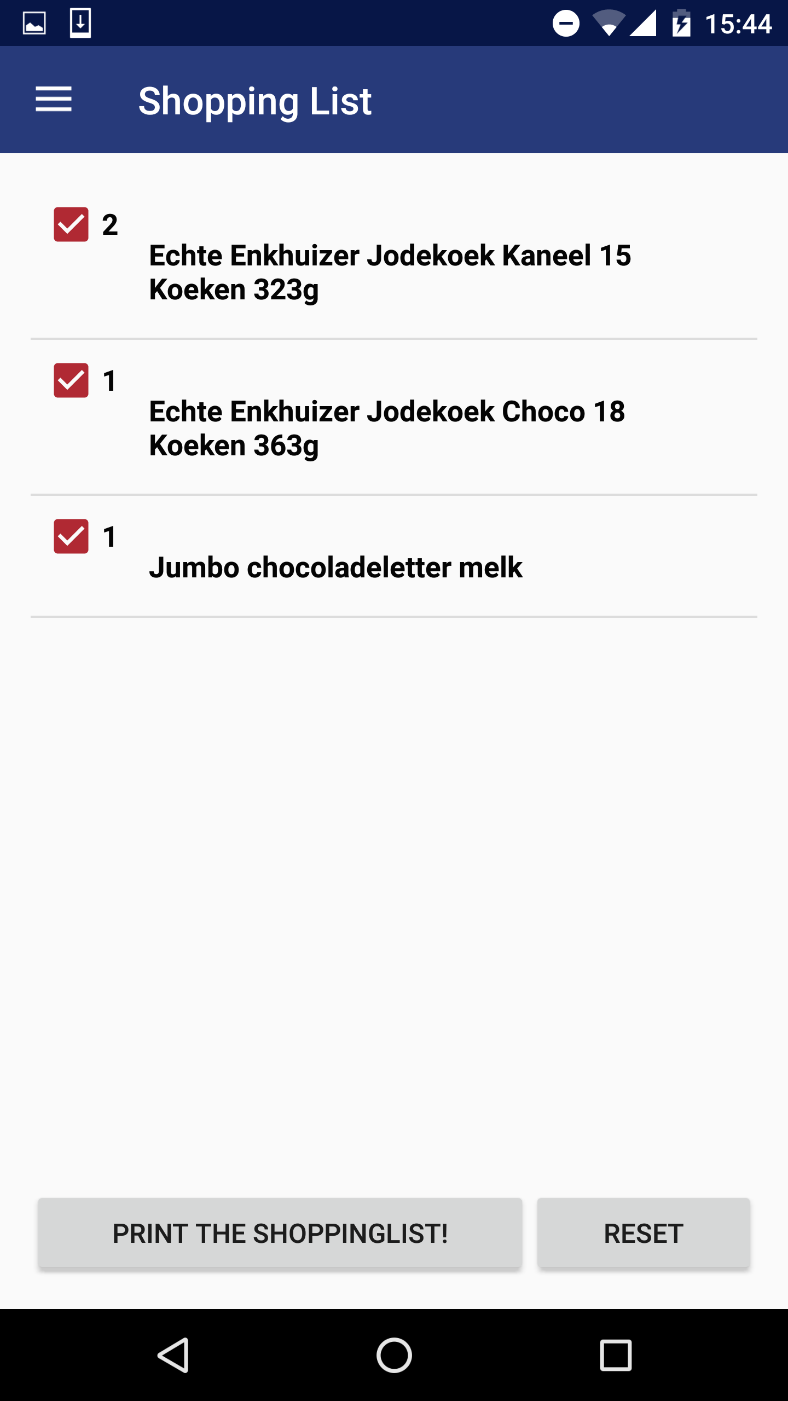


#### Logs

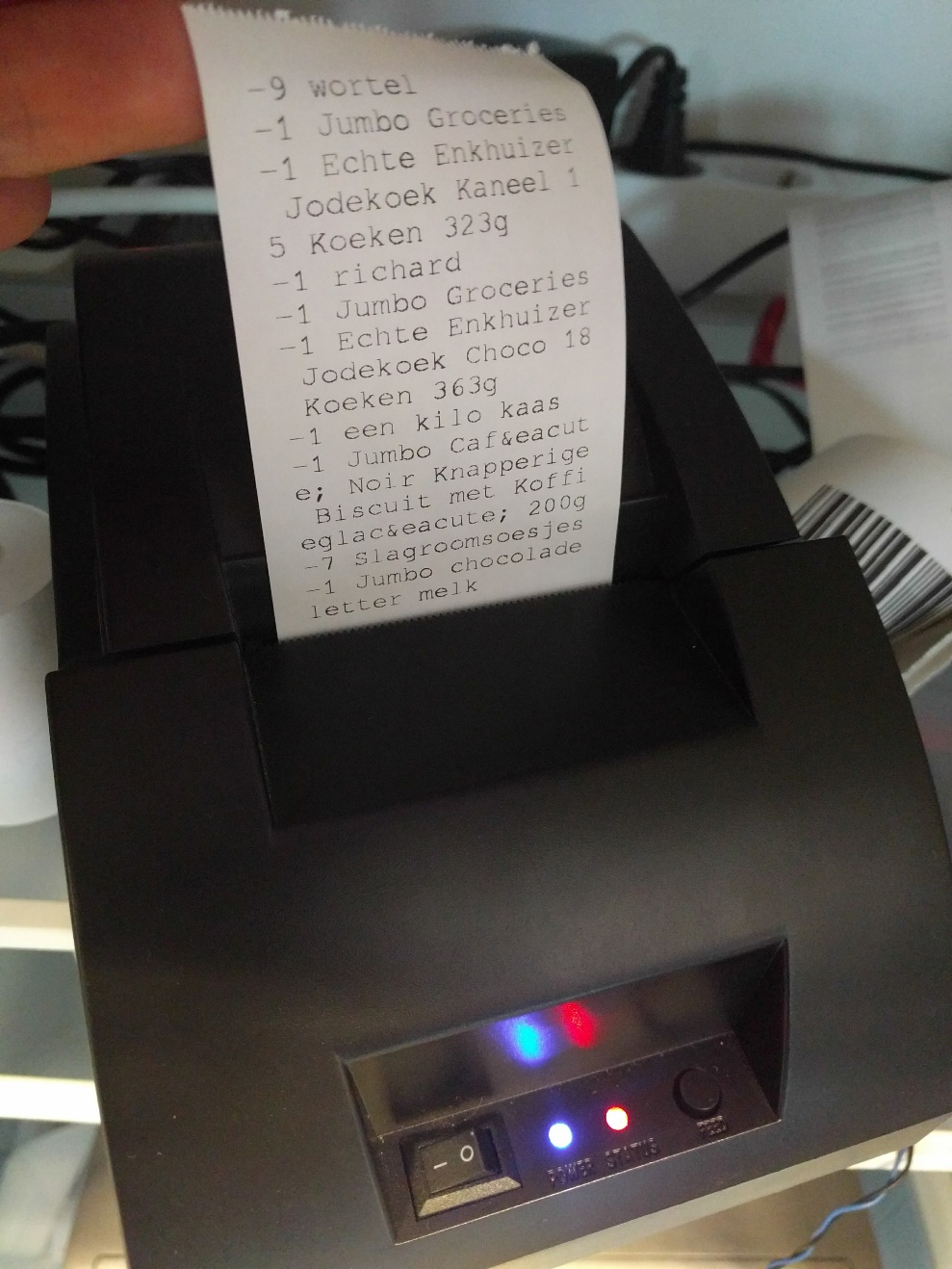
Op de logspagina is precies te zien wat er op welk tijdstip is gebeurd, toegevoegd, aangepast etc. We hebben hier geprobeerd om niet steeds dezelfde teksten te gebruiken, maar voor de meest gebruikte gebeurtenissen hebben we verschillende teksten gemaakt.

Deze lijst wordt direct van de server opgehaald en alleen de gebeurtenissen van het bekijken van de inhoud (het homescherm) worden eruit gefilterd, omdat dit er zoveel zijn en geen nuttige informatie geeft aan de gebruiker. Een functionaliteit die we nog willen implementeren is dat de gebruiker alles kan filteren en alleen bepaalde gebeurtenissen krijgt te zien.

#### Shopping List

Een boodschappenlijstje maken was een van de aangevraagde functionaliteiten. Dat kan op deze pagina. Uit de gebeurtenissentabel(log) wordt alles uitgelezen en alle producten die eruit zijn gehaald komen in deze lijst en het aantal wat er voor staat is het aantal te kopen items om weer op een maximum van eerder te komen. Wat we hiermee bedoelen: Als iemand 3 pakken melk in de koelkast heeft en op reset klikt, dan is de lijst leeg. Als hij er dan 1 uithaalt, dan komt er 1 te staan. Als hij er dan 2 toevoegt, dan zijn er 4 pakken melk in de koelkast, dan wordt dat als het maximum gezien/ het aantal dat erin moet staan.

We hebben een resetknop toegevoegd, zodat een gebruiker na de wekelijkse boodschappen de lijst kan leegmaken. De meeste onderdelen worden natuurlijk weggehaald, maar aanbiedingen of items die je maar een keer hebt gekocht en nooit meer wilt kan je dan zo weghalen, zodat ze nooit meer op je lijstje komen.

****Met de checkboxen kan je aanklikken wat je wel en niet wilt. Standaard staan ze allemaal aan.

### Technische details

In dit deel zullen we ingaan op de technische details van de software van de app. We zullen niet alle stukken code behandelen, omdat sommige delen een herhaling of een kleine aanpassing zijn op andere stukken en vooral omdat het 3600 regels Java zijn plus nog eens 2000 regels XML. We gaan vooral in op de onderdelen voor het contact tussen de app en de server, hoe we data opslaan en hoe we die data uitlezen en ‘slopen’ tot de gewenste data in het gewenste formaat.

#### Overview

De app is eigenlijk om één bestand geschreven, de Smartfridge.java klasse. Dit bestand heeft meer dan 600 regels code en regelt de communicatie met de server, dus de waardes goed versleutelen naar JSON en de URLs bedenken. Smartfride.java heeft een paar models nodig en een klasse RequestPost.java. Models zijn objecten die bepaalde eigenschappen hebben. Deze objecten kunnen we makkelijker mee- en terugsturen naar/van functies in de app dan losse waardes, dus daarom worden ze gebruikt. RequestPost.java stuurt de HTTP-requests en ontvangt het antwoord van de server. Deze klasse weet dus niks van de implementatie van de server of de app, dat wordt geregeld door Smardfridge.java. Daarnaast hebben we een SmartfridgeSave.java geschreven die alle data opslaat in het geheugen van de telefoon.

Als je alle voorgenoemde onderdelen zou kopiëren vanuit onze app en ergens anders in zetten, dan zou het gelijk werken, zonder ook maar één keer naar de documentatie te kijken of uren bezig te zijn om de huidige app te begrijpen.

De rest van de app zijn dus alleen maar onderdelen om het op het scherm te laten zien, zoals adapters of activities. Adapters leggen de link tussen een lijst met objecten en een lijst die je ook daadwerkelijk op het scherm ziet. Activities zijn de klasses die bij een bepaalde pagina horen en regelen dat bijvoorbeeld als je op de print-knop op de Jobspagina klikt dat daadwerkelijk een functie uit de Smartfridge klasse wordt aangeroepen en dat er een berichtje naar de server gaat.

De filetree van de app ziet er zo uit:

├── Smartfridge

│   ├── app

│   │   └── src

│   │   └── main

│   │   ├── AndroidManifest.xml

│   │   ├── java

│   │   │   └── com

│   │   │   └── svshizzle

│   │   │   └── pws

│   │   │   └── smartfridge

│   │   │   ├── activity

│   │   │   │   ├── DefaultFragment.java

│   │   │   │   ├── FragmentDrawer.java

│   │   │   │   ├── HomeFragment.java

│   │   │   │   ├── ItemDialog.java

│   │   │   │   ├── JobFragment.java

│   │   │   │   ├── ListFragment.java

│   │   │   │   ├── LogFragment.java

│   │   │   │   ├── Login.java

│   │   │   │   ├── MainActivity.java

│   │   │   │   └── SettingsFragment.java

│   │   │   ├── adapter

│   │   │   │   ├── HomeListAdapter.java

│   │   │   │   ├── ListListAdapter.java

│   │   │   │   ├── LogListAdapter.java

│   │   │   │   └── NavigationDrawerAdapter.java

│   │   │   ├── api

│   │   │   │   ├── Smartfridge.java

│   │   │   │   └── SmartfridgeSave.java

│   │   │   ├── model

│   │   │   │   ├── Item.java

│   │   │   │   ├── LogItem.java

│   │   │   │   ├── NavDrawerItem.java

│   │   │   │   ├── ShoppingListHolder.java

│   │   │   │   └── ShoppingListItem.java

│   │   │   └── request

│   │   │   ├── RequestClass.java

│   │   │   ├── RequestClassPost.java

│   │   │   └── RequestReturn.java

│   │   └── res

│   │   ├── layout

│   │   │   ├── activity\_home\_fragment.xml

│   │   │   ├── activity\_item\_dialog.xml

│   │   │   ├── activity\_job\_fragment.xml

│   │   │   ├── activity\_list\_fragment.xml

│   │   │   ├── activity\_log\_fragment.xml

│   │   │   ├── activity\_login.xml

│   │   │   ├── activity\_main.xml

│   │   │   ├── content\_login.xml

│   │   │   ├── fragment\_navigation\_drawer.xml

│   │   │   ├── home\_row.xml

│   │   │   ├── log\_row.xml

│   │   │   ├── nav\_drawer\_row.xml

│   │   │   ├── shopping\_row.xml

│   │   │   └── toolbar.xml

│   │   ├── mipmap-xxxhdpi

│   │   │   └── ic\_launcher.png

│   │   ├── values

│   │   │   ├── colors.xml

│   │   │   ├── dimens.xml

│   │   │   ├── strings.xml

│   │   │   └── styles.xml

│   │   ├── values-v21

│   │   │   └── styles.xml

│   │   └── values-w820dp

│   │   └── dimens.xml

│   ├── build.gradle

│   ├── keystore.jks

│   ├── projectFilesBackup

│   └── settings.gradle

└── text.txt

#### Smartfridge API

[Pagina 84-94]

De smartfridge API klasse, Smartfridge.java, is het belangrijkste gedeelte uit de app: deze klasse stuurt de HTTP-requests naar de server, met behulp van RequestPost.java, en leest de uitvoer uit en stuurt dat dan weer terug naar de klasse die de data heeft opgevraagd.

RequestPost.java is een klasse die door elke functie in Smartfridge wordt overschreven, dat betekent dat bepaalde methodes/functies van RequestPost worden herschreven, dus als de functie print() normaal een tekst print, dan kunnen we hem overschrijven dat hij opeens een plaatje print. RequestPost is een asynchrone klasse, dat betekent dat er eerst wat op Thread 1 wordt gedaan, de hoofdthread, waar je alleen de elementen op het scherm in aan kan passen. Die hoofdthread stuurt dan data naar een achtergrondproces, waar er wat gebeurt, in ons geval haalt hij wat op van de server, en die stuurt die data dan weer door naar een proces in de hoofdthread. Dat laatste proces overschrijven we, omdat bij elke pagina die we ophalen er andere data bij hoort en anders geïnterpreteerd moet worden.

Als de data is ontvangen van de server en het hoofdproces weer aan de beurt is moeten we die data weer naar de klasse die de data opvroeg sturen. We hebben gedacht over data via een Broadcast versturen, dat is een berichtje dat uitgezonden wordt overal naartoe, en alleen een stukje code die het wil hebben gebruikt het dan. Dit zou een mogelijke oplossing zijn, maar het zou voor extra complexiteit zorgen, wat het dus minder ‘hackable’ zou maken voor andere mensen die hun eigen app willen maken. We hebben uiteindelijk gekozen voor hetzelfde truckje als we in Smartfridge gebruiken, we overschrijven gewoon een methode/functie.

Voorbeeld:

Dit is een stuk code(vereenvoudigd) uit de Smartfridge klasse. De functies containsError en ContainsDone hebben geen inhoud, ze doen niks als ze aangeroepen worden. Op deze manier is het niet zinvol om de functie contains aan te roepen.

public void contains(String sort){  
  
 RequestClassPost request = new RequestClassPost(activity,jsonObject ){  
 @Override  
 protected void onPostExecute(RequestReturn requestReturn) {  
 super.onPostExecute(requestReturn);  
 if(requestReturn.isError()){  
 containsError(requestReturn.getResponse());  
 }else {

containsDone(process(requestReturn.getResponse()));  
 }  
 }  
 };  
 String url = apiUrl + "contains";  
 request.execute(url);  
}

public void containsDone(ArrayList<Item> e){  
}  
public void containsError(String e) {  
}

Maar als we dan het volgende stuk code (vereenvoudigd uit HomeFragment.java) gebruiken om een object aan te maken, dan is het wel zinvol. Hier overschrijven we de lege functie containsDone en laten we de app iets doen wat we willen als het is gelukt met ophalen. De functie createList is dan gedefinieerd in de aanroepende klasse.

Smartfridge smartfridge = new Smartfridge(getActivity()){  
  
 @Override  
 public void containsDone(ArrayList<Item> items) {  
 createList(items);  
 }  
  
 @Override  
 public void containsError(String e) {  
 errorContains(e);  
 //Shit, vincent did something wrong.  
 }  
};  
  
smartfridge.contains(sort);

Zoals te zien is gebruiken we voor elke functie uit de Smartfridge klasse een (naam)Done en (naam)Error functie. Dit doen we omdat verschillende functies verschillende parameters meekrijgen, in het voorbeeld een ArrayList en een String, terwijl andere functies helemaal niks of alleen maar een boolean meekrijgen. Ook is het voordeel van deze aanpak dat we meerdere aanvragen kunnen doen met één object, door vier of zes (error/done) functies te overschrijven en twee of drie startfuncties aan te roepen in plaats van twee a drie verschillende objecten te maken, wat zorgt voor verwarring en meer RAM gebruikt.

#### SmartfridgeSave.java

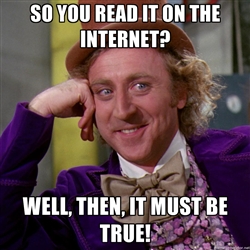
[Pagina 94-97]

Het opslaan van de userId en de apiURL moet goed gebeuren, zodat er in de hele app hetzelfde wordt gebruikt en dat het ook blijft bestaan tussen sessies. We maken in de app gebruik van de sharedPreferences, een functie van Android om teksten en getallen op te slaan. Dit is niet helemaal veilig, want de teksten worden in gewone tekst ergens opgeslagen, waar alleen hackers op een toestel met root bij kunnen. We vinden dit niet erg genoeg om beveiliging in te bouwen, want wat heb je aan wat informatie over een koelkast in vergelijking tot al je andere accounts en wachtwoorden?

Eerst hadden we in elke pagina een stuk code om de data op te halen, maar vanwege een bug in Android kregen we dan niet de informatie uit een andere activiteit, de loginactiviteit (alle pagina’s behalve login zitten in één activiteit). Hierdoor konden we wel inloggen, maar werden we gelijk teruggeleid naar de loginpagina, omdat die pagina’s de data niet konden uitlezen en de loginpagina opstartte. Omdat we toch alles al gingen ombouwen naar de Smartfridge.java ‘API’ hebben we besloten om toen ook alle dataopslag door 1 klasse te laten doen: SmartfridgeSave.java. Deze wordt normaal alleen opgeroepen door Smartfridge.java, omdat alleen die de inloggegevens nodig heeft. Toen we bijna klaar waren hebben we de functionaliteit toegevoegd dat de data wordt gecached, zodat je zelfs zonder internetverbinding ziet wat er in je koelkast zit (van de laatste keer dat je in de app keek). Hierdoor wordt de SmartfridgeSave klasse wel aangeroepen in de normale klassen, maar voor het hackgemak konden we dit beter via de Smartfridge klasse doen.

We hebben SmartfridgeSave een abstracte klasse gemaakt, dat betekent dat je er geen object van kan maken en dat de functies direct aan te roepen zijn, waardoor er minder intern geheugen gebruikt wordt. We hebben het vooral gedaan omdat het via een normale klasse niet werkte, maar met een abstracte klasse wel, iets wat we op stackoverflow.com lazen.





Samen met requestPost ziet het voorbeeld uit het vorige hoofdstuk er zo uit:



Dit plaatje geeft dus aan dat het zeer eenvoudig is om data op te halen met gebruik van de Smartfridge API. Het lijkt daarentegen dat er veel meer code wordt gebruikt/geschreven dan al de onderdelen in één klasse(homefragment). Dit is juist niet zo, want veel code wordt op meerdere plekken gebruikt, zo wordt bijvoorbeeld de functie getLog op de logpagina en de boodschappenlijstpagina gebruikt. De functie getUserId wordt zeker vaak gebruikt, bij elke aanvraag. Daarnaast is deze aanpak ook veel veiliger, want dezelfde code wordt maar op één plek gebruikt. Als er dan aanpassing zijn aan de architectuur moeten er weinig tot geen onderdelen aangepast worden, in plaats van op elke pagina Waardoor de kans kleiner wordt dat er dingen gemist worden. Het kopiëren van onze code naar een andere app is dan ook veiliger, omdat het maar een aantal bestanden zijn en de ‘eigen’ code is weinig in vergelijking tot de rest. Zelfs een andere taal/platform is hiermee mogelijk, want veel high-level talen, zoals C#, C++ en Java zijn erg vergelijkbaar en het ‘vertalen’ is dan meestal ook gemakkelijker dan helemaal opnieuw schrijven.

#### Boodschappenlijstje

Het boodschappenlijstje heeft het moeilijkste algoritme in de zin van ‘los deze puzzel op’. We maken gebruik van de log en rekenen daarmee uit wat er op het boodschappenlijstje moet staan.

We wilden het aantal altijd tot de minimale hoeveelheid sinds de reset aanvullen, dus een toevoeging aan de koelkast is -1 nodig en een delete is +1. Het kan alleen dan voorkomen dat er meer dingen zijn toegevoegd dan dat er zijn uitgehaald, dus moet de teller niet negatief worden.

De pseudocode van het bedenken van het boodschappenlijstje is:

Voor elk item in de log lijst:

Als het van het script itemChange komt:

Als de actie gelijk is aan toevoegen van een product

Aanpassing = -1;

Anders als de aanpassing gelijk is aan een verwijdering:

Aanpassing = +1;

Zoek het item op in een lijst van boodschappen (lokale lijst)

Als het item er nog niet in zit en de aanpassing is +1:

Voeg het item toe aan de lijst

Anders als het item er wel in zit:

Als (het aantal meer is dan 1 en aanpassing is -1) of (aanpassing is +1):

Verander het aantal van het item met aanpassing

Als het van het script resetLog komt:

Maak de lijst leeg.

Na het maken van de lijst wordt het omgezet in een andere lijst die dan op het scherm getoond wordt.

Voor het uitprinten wordt de lijst uitgelezen en wordt elk item omgezet tot één String (het getal wordt toegevoegd aan de naam) en wordt de lijst van teksten met JSON naar de server gestuurd.

#### Issues app

De app maken was niet zo makkelijk als gedacht en als de rest van het project. Er waren meerdere dingen waar we van dachten dat het moest werken, maar dat niet deed. We hebben een klein lijstje met de opmerkelijkste foutjes door ons.

##### Willekeurige crashes

Regelmatig en vooral op school crashte de app. En omdat we dan op school waren was het moeilijk te kijken wat er nou misging, want we hadden geen geschikte computer om Android Studio redelijk op te draaien. Na heel wat zoeken bleek dat je elementen, zoals een textView, niet kan aanpassen als ze niet op het scherm zijn. Dit kan gebeuren als je snel van activiteit of fragment wisselt door middel van het menu. Op de een of andere manier draait de code dan nog wel en als het een serveraanvraag is op heel erg langzaam internet, dan wordt die code gedraaid als het element al weg is en dan crasht de app. Wat we hiertegen hebben gedaan is door voor elke aanpassing van listView, textView etc. steeds checken of het wel een echt object is door het object te vergelijken met null. null is een speciale waarde en geeft aan dat het niet bestaat of (nog) geen waarde heeft. Als het gelijk is aan null, dan kappen we gelijk de hele functie af, want het kost anders toch alleen maar processorkracht.

##### Opslaan

Het opslaan van data, zoals userId en APIURL, ging niet helemaal vlekkeloos als het door verschillende activiteiten opgeslagen of aangevraagd werd. Hiervoor hebben we het hele internet afgespeurd en uiteindelijk kwamen we op Stackoverflow.com een antwoord van iemand die hetzelfde probleem had en dat een abstracte klasse het opgelost had. Hierdoor hebben we SmartfridgeSave gemaakt en het werkte.

##### Dubbelen

Op een keer waren we aan het testen met de koelkast en de app. Via SSH hadden we verbinding en hadden we onze code opgestart. Alles werkte totdat we producten gingen toevoegen en verwijderen: in de app werd het met 2 veranderd in plaats van met 1. Na heel wat zoeken, o.a. de server bekeken en de app, gingen we kijken hoeveel python-processen aan het draaien waren: 2. De beide processen ontvingen de barcode van de scanner en stuurden afzonderlijk een bericht naar de server.

## Documentatie

[Pagina 149-153]

We proberen zoveel mogelijk documentatie bij de app, de python code en de server-code te maken, maar we maken ook een documentatie-website voor de webserver, hoe je hem bestuurt en hoe je data opvraagt. Door middel van de documentatie is het tijdens het samenwerken gemakkelijker, omdat we dan altijd op één lijn zitten. We weten dan beide exact hoe bepaalde invoer gelezen of ingevoerd moet worden (hoofdletter gevoeligheid en dergelijken). Voor het gemak hebben we de documentatie gezet op <https://arendjan.github.io/slate/>, zo kunnen we er altijd en overal bij(mits we internet hebben).

## Andere software

Om onze software te maken hadden we software nodig om het te maken. Hieronder hebben we een lijst gemaakt met alle programma’s die we hebben gebruikt om dit project tot een goed eind te brengen.

|  |  |
| --- | --- |
| Wat | Waarvoor |
| Atom | Python + PHP code te schrijven |
| GitHub | Versiebeheer |
| Slate | Repository op GitHub waar we de documentatiewebsite mee hebben gemaakt |
| Putty | SSHverbinding met koelkast/laptop te maken |
| Filezilla | Bestandsoverdracht van/naar server/Raspberry Pi |
| Android Studio | App maken |
| Stackoverflow | Al onze programmeervragen beantwoord! |
| Python | Onze code draaien |
| PHPMyAdmin | Om in onze database te kijken als het misging |
| PostMan | Om onze server code te testen |

# Licenties

We gebruiken een aantal plugins + onderdelen in onze code waar een licentie bij hoort. We hebben hier een lijstje van gemaakt met de bijbehorende licentie.

|  |  |  |
| --- | --- | --- |
| Wat | Waarvoor | Licentie |
| pyBarcode | Barcode plaatje genereren. | MIT |
| Python-crontab | Crontab genereren | LGPLv3+ |
| Slate | Documentatie | Apache 2.0 |

# Versiebeheer

[Pagina 123-135]

We hebben heel veel code geschreven en omdat we vaak tegelijkertijd aan (verschillende) onderdelen werkten moesten we iets van versiebeheer gebruiken, zodat we onze code ook op meerdere computers konden bekijken en aanpassen. We hebben voor Git, specifieker GitHub gekozen, omdat Arend-Jan hier al een studentenaccount had, waardoor we een priverepository konden aanmaken, zodat niet iedereen het kon bekijken/meewerken. Het voordeel van GitHub is dat het heel erg goed op Windows draait, waar we het meeste op gewerkt hebben en ze hebben Atom gemaakt, een code-editor die gelijk aanpassingen en aangepaste bestanden laat zien.

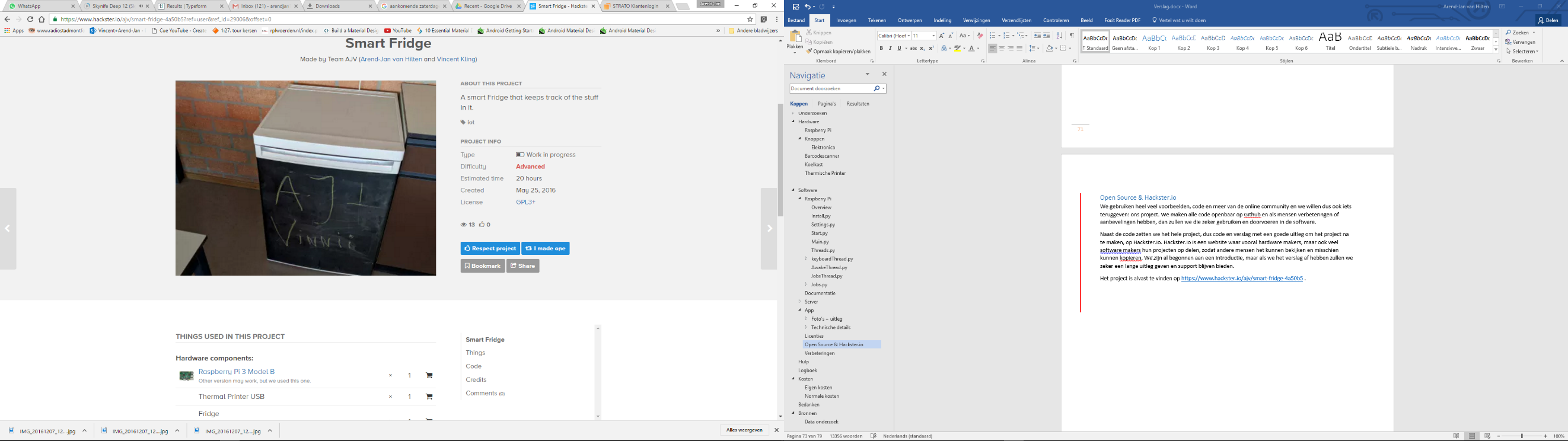
We hebben een aantal keren een conflict gehad met stukken code waar we ongeveer hetzelfde hadden aangepast, waardoor Git dat niet kon oplossen. We hebben vaak de commit weggegooid, de code gedownload, het was vaak server code, en in een nieuwe commit gezet en die wilde dan wel mergen met de master.

We hebben de hele GitHub log in het bijlagenboek gezet oppagina119 tot pagina 131.

# Open Source & Hackster.io

We gebruiken heel veel voorbeelden, code en meer van de online community, wat allemaal open source is zoals Linux, Python, alle plugins en we willen dus ook iets teruggeven: ons project. We maken alle code openbaar op Github en als mensen verbeteringen of aanbevelingen hebben, dan zullen we die zeker gebruiken en doorvoeren in de software. We hebben gekozen voor een GPL (3+) licentie, dit houdt in dat mensen het vrij mogen gebruiken, maar dat ze ons wel credit moeten geven en dat het niet voor commerciële doeleinden gebruikt mag worden. We hebben hiervoor gekozen omdat we het niet voor een bedrijf hebben gemaakt, die er dan dikke winst mee kan maken en omdat wij graag als de originele auteurs gezien willen worden.

Naast de code zetten we het hele project, dus code en verslag met een goede uitleg om het project na te maken, op Hackster.io. Hackster.io is een website waar vooral hardware makers (van makerspace), maar ook veel software makers hun projecten op delen, zodat andere mensen het kunnen bekijken en misschien kunnen kopiëren. We zijn al begonnen aan een introductie, maar als we het verslag af hebben zullen we zeker een lange uitleg geven en support blijven bieden.

Het project is alvast te vinden op <https://www.hackster.io/ajv/smart-fridge-4a50b5> .

# Verbeteringen

Iets om te verbeteren is eigenlijk iets wat iedere programmeur wel kent, het opschonen van code. Ook bij ons is de code niet optimaal mogelijk. In de meeste PHP-scripts zijn bijvoorbeeld de eerste aantal regels hetzelfde. Voor zulke dingen zouden we een los script kunnen maken dat ieder ander PHP-script weer gebruikt.

Een verbetering aan de kant van de Raspberry die we zouden kunnen maken is het checken voor errors, zodat zelfs als het internet het niet doet, het opgeslagen wordt en dat ze verzonden worden als het internet het wel weer doet. Dit is natuurlijk handig voor wanneer het internet eruit ligt door een stroomstoring, werkzaamheden of iets dergelijks.

Ook zou er in de code iets verwerkt worden dat de Raspberry bij excepties van het markeren niet de job gaat doen, maar overslaat juist en de volgende keer doet, zodat een job niet meermaals gedaan wordt.

Een vaak aangevraagde feature is het vinden van recepten op basis van de inhoud van de koelkast. Dit is makkelijk te doen, door op te vragen wat er in de koelkast zit en dan de browser op te starten en naar <http://www.smulweb.nl/recepten/boter+melk+wortel+kaas> te sturen, waar dan de producten achter elkaar erin gezet moeten worden.

We zouden ook extra jobs kunnen toevoegen. Zodat de Raspberry muziekjes, natuurlijk van DjVinnii, kan afspelen of een foto kan maken.

In de app willen we nog een functionaliteit implementeren zodat de gebruiker de gehele log kan filteren.

Naast het toevoegen van features willen we de lay-out van de app testen. Dit willen we doen door verschillende lay-outs te maken en die te laten zien aan mensen en vragen welke ze beter of overzichtelijker vinden.

# Conclusie

Onze hoofdvraag is “Hoe komt een slimme koelkast tot stand?” en onze deelvragen zijn:

- Welke stappen zijn essentieel voor het maken van een slim apparaat?

- Welk formaat is het beste om data te sturen?

- Wat vinden mensen van het idee slimme koelkast?

Een slimme koelkast komt vooral tot stand door het uitzoeken van de doelgroep en het onderzoeken van methodes om data te versturen. Dit hebben we gedaan door een tweetal enquêtes te doen en een bronnenonderzoek. We moesten naast het uitzoeken van de doelgroep, ook die doelgroep vragen wat zij belangrijk vinden.

Om een slimme koelkast te maken heb je meerdere apparaten nodig, omdat het niet mogelijk is om een app te draaien op een Raspberry Pi, maar de server zou wel daarheen verplaatst kunnen worden.

Naast meerdere apparaten zijn er ook veel regels code nodig om de koelkast slim te maken. Zonder de code zou het een koelkast zijn met een willekeurige computer erin zonder functie. Voor het maken van deze functies is er de input van de doelgroep nodig om het zo goed mogelijk bij hun te laten passen. Op het moment dat dit mislukt wordt er teveel gewerkt, code gemaakt, zonder een uitkomst. Hier moet dus heel erg goed op gelet worden om geen geld of tijd te verdoen.

Een goede manier om data te versturen is zeer belangrijk om de snelheid en de mogelijkheden te verzekeren. Als een manier om data te versturen te veel data kost(internet), dan kan het gebeuren dat het langzamer gaat werken naarmate het meer gebruikt wordt of er meer data verstuurd moet worden. De mogelijkheden van een goede manier zijn ook belangrijk, want als een bepaalde manier om data niet ondersteund wordt door een programmeertaal waar je naartoe wilt verhuizen moet je de manier aanpassen of zorgen dat het ondersteund wordt. Dit zorgt voor meer werk, wat alleen maar geld en tijd gaat kosten.

# Discussie

De tweede enquête hebben we minder lang opengesteld dan de eerste enquête. Hierdoor hebben wij minder resultaten, en zal ook de conclusie bij deze enquête ook wat minder nauwkeurig zijn. Als we de resultaten van de tweede enquête vergelijken met de eerste enquête zien we wel dat de gemiddelden dicht van elkaar in de buurt komen.

Ook valt er te betwijfelen wat het verstandigst is, een webserver op een externe host of een webserver op de Raspberry Pi. Beide opties hebben zowel hun voordelen als nadelen. Het belangrijkste om te overwegen is of de gebruiker overal de app wilt kunnen gebruiken of alleen thuis. Het overal kunnen gebruiken van de app kan er dan wel voor zorgen dat er eerder iets gehackt wordt en dan gelijk een stuk groter is dan wanneer er maar 1 lokale database gehackt wordt.

De samenwerking verliep goed en omdat we beiden iets anders hadden om te doen, liepen we elkaar niet in de weg of hoefden we niet op elkaar wachten. Het nadeel hieraan is dat de manier van werken anders is en dat er soms onenigheid was over de manier van programmeren. Naast de programmeerstijl was er ook even een misverstand over bepaalde variabelen in de database: Arend-Jan dacht dat er twee verschillende getallen waren, gesloten en open, en Vincent dacht dat er twee getallen waren, totaal en open, en dat gesloten het verschil was tussen totaal en gesloten. Dit misverstand hebben we gelukkig snel opgelost, maar dit zorgde wel voor een aantal aanpassingen.

# Logboek

[Pagina 154 - 157]

In het logboek is goed te zien wie wanneer wat heeft gedaan. In totaal zijn we … uur met dit PWS bezig geweest. Arend-Jan heeft er 156 uur in gestoken en Vincent 88.5 uur.

Wat belangrijk is om te weten is dat Arend-Jan vooral bezig is geweest met de python en de app. In de app is echter veel tijd gaan zitten. Vincent is vooral bezig geweest met de PHP en de database.

We hadden van tevoren niet gedacht er zoveel uren aan te zitten en de app is qua uren uit de hand gelopen.

# Kosten

## Eigen kosten

Voor ons PWS hebben we natuurlijk wat kosten gemaakt. Hieronder hebben we een lijstje met producten die we nodig hebben gehad, in combinatie met de bijbehorende kosten.

|  |  |
| --- | --- |
| Product: | Kosten: |
| Koelkast | - |
| Raspberry Pi | - |
| Thermische Printer | - |
| Omnidirectionele Barcode Scanner | - |
| 3 Arcade knoppen | - |
| Relais | - |
| SD-kaart | € 7,95 |
| Oplader | € 12,95 |
| Case | € 8,95 |
| Overvalsluiting | € 4,49 |
| Extra sleutel voor discusslot | € 4,60 |
| Heel veel jodenkoeken | Heel veel |
| Totaal: | € 38,94 |

## Normale kosten

Het is natuurlijk ook interessant om te weten wat een normaal persoon zou moeten betalen als ze het zelfde projectje na willen maken. Aangezien wij het een en ander gesponsord hebben gekregen, zijn onze eigen kosten hier niet geschikt voor. Ook hebben wij een overvalsluiting gebruikt die natuurlijk niet nodig is in een normaal huishouden. Hier is een compleet lijstje met onderdelen die nodig zijn voor dit project:

|  |  |
| --- | --- |
| Product: | Kosten: |
| Koelkast | ~ €150,- |
| Raspberry Pi | € 40,40 |
| Thermische Printer | ~ € 25,- |
| Omnidirectionele Barcode Scanner | ~ €25,- |
| 3 Arcade knoppen | € 26,85 |
| 3 Relais | ~ €25,- |
| SD-kaart | € 7,95 |
| Oplader | € 12,95 |
| Case | € 8,95 |
| Totaal: | € 322,10 |

# Bedanken

Wij willen graag de onderstaande personen bedanken:

Meneer Kok, voor het verlenen van divers gereedschap.

Mark Zimmerman, voor het regelen van de koelkast.

Ariën en Annet Smorenburg, voor de tijdelijke opslag van onze koelkast.

Mevrouw van Doorn, voor goede tips en het ter gebruik stellen van het TOA-lokaal.

Erik van Hilten, voor het sponsoren van de barcode scanner en de thermische printer.

Bob Zevenbergen, voor het lenen van een laptop (die eigenlijk voor de Kalsbeek reünie website is).

Meneer Beukers, voor het proberen van het openstellen van de poorten voor de FTP van Strato.

Meneer Prins, voor het openen van het lokaal.

Koen van Elsen, voor het begeleiden van ons PWS.

Wim Groeneveld, voor het kleine beetje begeleiden in leerjaar 2015-2016.  
Roland Hollaar, voor het sponsoren van de Raspberry Pi en de relais.

Bart Hornman, ook voor het sponsoren van de Raspberry Pi en de relais en het geven van commentaar.

Nick Gans, voor het sponsoren van de vele jodenkoeken.

Richard Eveleens, ook voor het sponsoren van de vele jodenkoeken.

Mensen uit de “Ik ben programmeur” groep op facebook, voor het invullen van de enquête en feedback geven.

Deelnemers van onze enquêtes, voor het invullen van onze enquêtes (duh).

SOS Solutions, voor de 75 euro die we hebben gewonnen.

Marieke van Hilten, voor het doorlezen van het verslag en maken van opmerkingen.

De printer van huize van Hilten, voor het lijden tijdens het printen van ons verslag.

# Bronnen

## Data onderzoek

<http://www.w3schools.com/xml/>

<http://www.w3schools.com/json/default.asp>

<http://stackoverflow.com/questions/51492/what-usable-alternatives-to-xml-syntax-do-you-know>

<http://intellimath.bitbucket.org/axon/>

## Barcode Scanner

<http://www.embedds.com/software-debouncing-of-buttons/>

<http://www.alibaba.com/trade/search?fsb=y&IndexArea=product_en&CatId=&SearchText=rfid+tags>

<http://www.gs1.org/how-calculate-check-digit-manually>

## Install.py

<http://scruss.com/blog/2015/07/12/thermal-printer-driver-for-cups-linux-and-raspberry-pi-zj-58/>

## KeyboardThread.py

<https://www.raspberrypi.org/forums/viewtopic.php?f=45&t=55100>

## Algemeen

[http://stackoverflow.com/questions/.\*/.\*](http://stackoverflow.com/questions/.*/.*)

<http://www.sttproducts.nl/werkwijze/>

1. *https://www.raspberrypi.org/forums/viewtopic.php?f=45&t=55100* [↑](#footnote-ref-2)