# Project verslag

Groep 9 - Smart brievenbus

Annick Vink - s1125953
Martin Krikke - s1131049

# Introductie & doelstelling

Hoewel veel post vandaag de dag digitaal gedeeld wordt, krijgen mensen nog steeds dagelijks brieven en pakketjes binnen via een brievenbus. Voor mensen met een brievenbus die los zit van het huis, is het legen van de brievenbus een taak die vaak wordt uitgesteld. Vaak is de post niet zo interessant, of zit er gewoon niks in. Maar tijdsgebonden brieven, zoals belangrijke rekeningen die betaald moeten worden, kunnen hier ook zomaar tussen zitten. Het is belangrijk dat deze tijdig gelezen worden. Het zou ideaal zijn als de eigenaar van de brievenbus op afstand kan bijhouden wat er in de brievenbus ligt, om zo te bepalen of het legen van de brievenbus het waard is.

Ons idee voor het project is een Smart Brievenbus, een ‘slimme’ brievenbus die met behulp van sensoren en een online omgeving de gebruiker op de hoogte kan houden van binnengekomen post. De gebruiker zou dan een notificatie krijgen als er post binnen is gekomen, en in de app kunnen ze extra informatie over de binnengekomen post zien; is het een brief of een pakketje? Hoe zwaar is het? Om welke tijd is het afgeleverd? Daarbij maakt de brievenbus een foto van de zojuist binnengekomen post, zodat ze gelijk ook kunnen zien wat er precies binnen is gekomen.

Daarnaast willen we ook het probleem van het zoeken naar de juiste sleutel voor de brievenbus elimineren, door een vingerafdrukscanner te implementeren bij de brievenbus. Het slot gaat dan van de brievenbus af als de vingerafdruk herkent wordt.

Wij willen de sensoren en microcontroller(s) in de brievenbus van stroom voorzien door middel van een zonnepaneel en een powerbank. Op die manier zal er altijd genoeg stroom zijn en hoeft de gebruiker niet zelf de brievenbus van stroom te voorzien met bijvoorbeeld batterijen of een oplader.

# Hoofdvraag en deelvragen

Tijdens dit verslag zullen we de volgende hoofdvraag beantwoorden:

**Hoe kunnen wij een slimme brievenbus ontwerpen die de gebruiker op de hoogte houdt van binnengekomen post?**

Om deze hoofdvraag te beantwoorden, hebben we belangrijke onderdelen van deze vraag onderverdeeld in 5 deelvragen. Deze luiden als volgt:

- **Welke data hebben we nodig en welke sensoren horen daarbij?**
  Data is eigenlijk de kern van ons project. Zonder de data die we uitlezen van sensoren, is het onmogelijk om ons idee van een slimme brievenbus te realiseren. Daarom gaan we onderzoek doen wat voor data we kunnen gebruiken om ons idee uit te werken.
- **Wat voor andere hardware hebben we nodig?** Zonder onderzoek gedaan te hebben, weten we nu al dat het niet alleen sensoren zijn die we nodig hebben. We hebben het bijvoorbeeld al gehad over de powerbank en zonnepaneel die de brievenbus van stroom moet voorzien. Ook naar deze onderdelen gaan we onderzoek doen, om het beste uit te kiezen voor onze use-case.
- **Hoe maken we de brievenbus veilig?** Tijdens onze project briefing, kwam het onderwerp van veiligheid naar boven. Hoe zorgen we ervoor dat alleen de gebruikers die daar recht op hebben, toegang krijgen tot de informatie van de brievenbus? Een onderzoek naar encryptie, en eerdere kennis uit de module “ISECU” komt hierbij van pas. Daarnaast gaan we bij deze vraag ook in op de fysieke beveiliging van de brievenbus.
- **Hoe moet de brievenbus in elkaar zitten?** Het doel is natuurlijk om uiteindelijk een brievenbus te maken met slimme functionaliteiten. Normaal heeft een brievenbus een vrij eenvoudig ontwerp, maar door onze toevoeging van allerlei hardware en sensoren, is het belangrijk om goed over ons ontwerp na te denken. Denk bijvoorbeeld aan cable management, of de plaatsing voor het zonnepaneel.
- **Hoe tonen we de data op een nuttige en overzichtelijke manier aan de gebruiker?** Zelfs als de brievenbus zelf zo goed functioneert, is het waardeloos als de gebruiker er niks mee kan. Om deze reden is het belangrijk dat de app die we gaan bouwen, gebruiksvriendelijk is.

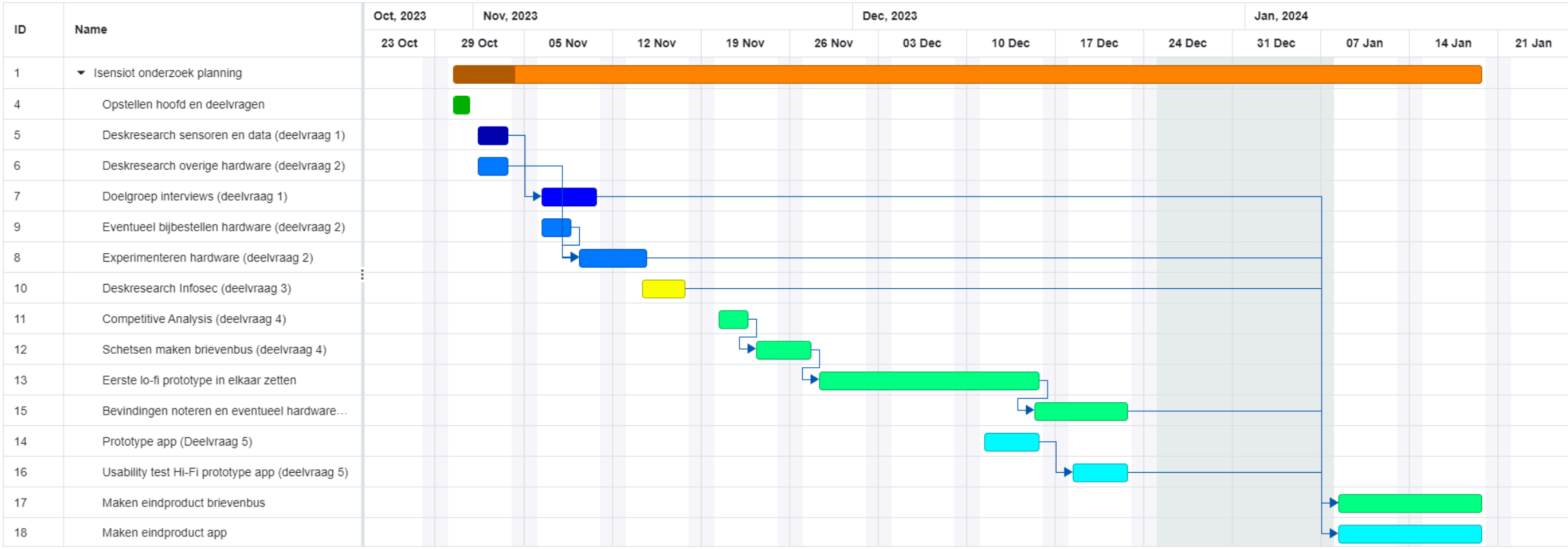
Op basis van de antwoorden op deze deelvragen, kunnen we tot een conclusie komen voor de hoofdvraag. In het volgende hoofdstuk volgen de conclusies op de deelvragen, met daarbij de onderzoeksmethodes die zijn uitgevoerd.

# Onderzoeksopzet en planning

Op basis van de eerder besproken hoofdvraag en deelvragen, gaan we een onderzoek uitvoeren, gebruikmakend van verschillende onderzoeksmethoden. Bij elke deelvraag hoort een aparte onderzoeksmethode, en deze worden hier besproken.

- **Welke data hebben we nodig en welke sensoren horen daarbij?**
  Allereerst is het voor deze deelvraag belangrijk om deskresearch te doen naar de verschillende sensoren die beschikbaar zijn. Natuurlijk weten we van een aantal sensoren af, maar het is belangrijk dat we geen tunnelvisie hebben met de sensoren kennis die we al hebben. Door erachter te komen wat voor andere sensoren er beschikbaar zijn, kunnen we mogelijk op ideeën komen die we eerder nog niet hadden.
  Naast deskresearch willen we ook een aantal mogelijke gebruikers interviewen, om erachter te komen wat voor data zij interessant zouden vinden om te kunnen zien in de app. Ook krijgen we hierdoor een beter beeld van onze doelgroep, door specifiek mensen met een losse brievenbus te interviewen.
- **Wat voor andere hardware hebben we nodig?**
  Om erachter te komen wat voor andere hardware geschikt is voor ons project, gaan we eerst deskresearch doen naar geschikte microcontrollers, powerbank en zonnepaneel. Met deze onderdelen kunnen we vervolgens experimenten uitvoeren als het gaat om stroomverbruik, en hoe lang de brievenbus kan functioneren op een volle powerbank.
- **Hoe maken we het veilig?**
  Hoewel wij als informatica studenten al eerder onderworpen zijn aan het onderwerp van infosec, is het nog steeds belangrijk dat we hier deskresearch naar doen, om de beste oplossingen voor onze use-case toe te kunnen passen.
- **Hoe moet de brievenbus in elkaar zitten?**
  Om ons eindresultaat beter te maken dan die van soortgelijke producten, gaan we beginnen met het maken van een competitive analysis, waarin we gaan kijken wat er goed en slecht is aan al bestaande brievenbussen, en hoe onze brievenbus daarop kan verbeteren. Daarnaast zullen we veel moeten prototypen met het in elkaar zetten van de werkelijke brievenbus. Een Lo-Fi prototype kan ons helpen met het ondervinden van problemen die we al kunnen verhelpen in ons uiteindelijke eindproduct. Hierbij moet je ook bijvoorbeeld denken aan het maken van 3D modellen van de brievenbus.
- **Hoe tonen we de data op een nuttige en overzichtelijke manier aan de gebruiker?**
  Aangezien onze projectgroep bestaat uit studenten met een achtergrond in UX/UI design, zouden we alle kennis in huis moeten hebben voor het realiseren van een app die gebruiksvriendelijk is voor de gebruiker, en die de gebruiker nuttig vindt. Onderzoek om onze ontwerpen te testen, zal gebeuren aan de hand van usability tests.

Aan deze deelvragen zal gewerkt worden aan de hand van de volgende planning:




In de gannt chart hierboven is te zien op welke manier we ons onderzoek willen uitvoeren, om de bevindingen hieruit te kunnen gebruiken voor onze eindproducten. Zo kun je zien dat bijvoorbeeld deskresearch gedaan wordt als voorbereiding op de interviews. Elke deelvraag vult op deze manier aan op het maken van het eindproduct, en in feite op de hoofdvraag van dit project.

# Technische vs functionele eisen en specificaties

De brievenbus heeft zowel technische als functionele eisen en specificaties. Deze staan hier verwerkt, en zullen we aan de hand van de verschillende deelvragen uitwerken.

## Technische eisen/specificaties

Connectiviteit:
- De brievenbus moet draadloze connectiviteit via Wi-Fi ondersteunen, om te kunnen communiceren met een API.
- De API moet zowel met de brievenbus communiceren, als met de app.

Stroomvoorziening:
- Een betrouwbare stroomvoorziening, die ervoor zorgt dat de brievenbus zelfvoorzienend is van stroom.
- Stroomvoorziening moet geen taak worden voor de gebruiker, dus geen batterijen die telkens vervangen moeten worden.
- Efficiënt energiebeheer om ervoor te zorgen dat de brievenbus 24/7 operationeel is.

Sensoren:
- Ingebouwde sensoren die kunnen detecteren of er post is binnengekomen.
- Genoeg sensoren voor accurate detectie van post.

Camera:
- Camera die de gebruiker meer context kan geven over het soort post dat er is binnengekomen.
- Op afstand te bekijken.

Dataopslag:
- Voldoende opslagruimte voor het opslaan van gegevens, zoals afbeeldingen en postdata.

## Functionele eisen/specificaties

Post detectie:
- Brievenbus moet de mogelijkheid hebben om te kunnen detecteren of er poststukken worden ingegooid of weggehaald.
- De detectie moet in te zien zijn in een gebruiksvriendelijke app.

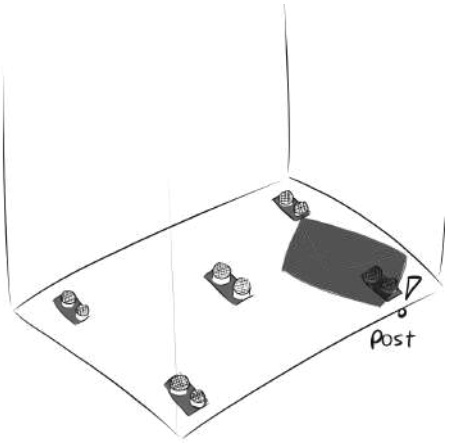
Notificaties:
- Wanneer de post gedetecteerd wordt, moet de gebruiker in real-time een notificatie hierover krijgen.
- De notificaties moeten accuraat zijn, dus alleen als er echt een nieuw poststuk is binnengekomen.

Veiligheid:
- Post moet veilig opgeslagen kunnen worden in de brievenbus.
- Gegevens in de app moeten versleuteld kunnen worden.

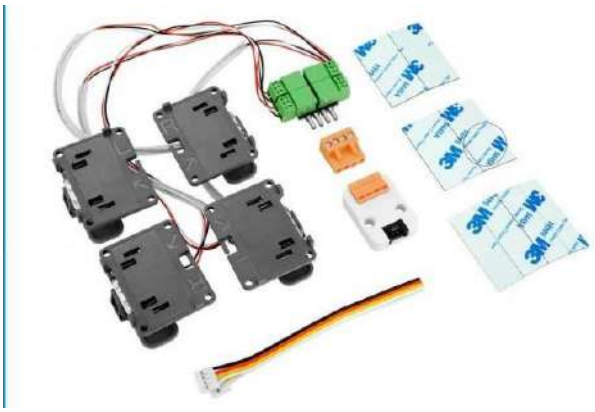
Weersbestendigheid:
- De brievenbus moet buiten kunnen staan, en dus tegen verschillende weersomstandigheden zoals regen en wind kunnen.

# Het onderzoek

## Welke data hebben we nodig en welke sensoren horen daarbij?




Toen we tot het idee waren gekomen voor het maken van een slimme brievenbus, kwam meteen het gebruik van ultrasone sensoren naar boven. Deze sensoren zouden geschikt kunnen zijn voor het detecteren van objecten die zich in de brievenbus bevinden. Wel is dit een vrij simpele benadering, waarbij er niet veel data komt kijken. Je weet bijvoorbeeld niet wat er binnen is gekomen, en het kan ook gebeuren dat een brief net tussen de sensoren valt waardoor deze niet wordt gedetecteerd.
We zijn toen verder op zoek gegaan, en kwamen een weegschaal kit tegen voor gebruik met microcontrollers, met goede documentatie voor gebruik met een Arduino.
Als de brievenbus als een weegschaal functioneert, heb je een stuk meer data dan met ultrasone sensoren. Op basis van het gewicht, kan je dan bijvoorbeeld uitlezen of er een brief of pakket is binnengekomen. Daarnaast heb je ook niet het probleem dat het poststuk naast de sensor kan vallen, als de hele onderkant van de brievenbus als het ware 1 grote sensor is. Wel is een mogelijk probleem dat de weegschaal niet accuraat genoeg zou kunnen zijn om een brief te kunnen detecteren, omdat deze vrij licht zijn. We kwamen tot de conclusie dat een soort combinatie van een nabijheidssensor (zoals een ultrasoon sensor) en een weegschaal, de meest accurate resultaten zou kunnen bieden. De nabijheidssensor detecteert dan of er iets is binnengekomen, en de weegschaal kan dan wat meer details geven over het soort poststuk.



Nu we een algemeen idee hadden van wat voor functionaliteiten onze slimme brievenbus zou hebben, zijn we begonnen met het houden van interviews, met mensen die een brievenbus hebben die los staat van de woning. Tijdens één van de interviews, had iemand het erover dat veel post die binnenkwam, eigenlijk niet echt belangrijk is. Het ging dan voornamelijk om reclamebladen. Hierdoor kregen we het inzicht dat niet alleen de vraag óf er iets is binnengekomen belangrijk is, maar ook wat het is, om te kunnen bepalen of het ophalen van de post het waard is. We zijn toen naar mogelijkheden gaan kijken met betrekking tot het implementeren van een camera in de brievenbus.

Deze hebben we gevonden en functioneert onder andere met een Raspberry PI microcontroller. De resolutie van 640 x 480 px zorgt ervoor dat de camera niet te duur is, maar nog wel relatief goede foto's kan maken. Om te kunnen weten wanneer de camera een foto moet maken, hadden we eerst het idee om een LDR te gebruiken. Deze sensor zou dan detecteren als de klep open wordt gedaan, door een hogere hoeveelheid lichtinval. Probleem met dit idee was dat het detecteren dan niet goed zou werken als er post bezorgd zou worden in de ochtend/avond. We hebben er daarom voor gekozen om een infrarood nabijheidssensor te implementeren in de brievenbus. Deze werkt op basis van infrarood licht, en zou dus niet beïnvloed moeten worden door factoren van buitenaf. Deze detecteert of er een object in de brievenbus is gedaan en leest op die manier af of het maken van een foto nodig is.

Tijdens de testfase kwamen we erachter dat de nabijheidssensor niet altijd de post oppikt die in de brievenbus wordt gedaan. We hebben dus bedacht om een Hall Effect sensor te gebruiken als backup. In de klep van de brievenbus kunnen we dan een magneetje doen, waardoor de Hall Effect sensor naast de gleuf kan detecteren of de klep open is geweest of niet.

Verder is uit de interviews naar boven gekomen dat het idee van notificaties krijgen voor het ontvangen van poststukken wel gewaardeerd zou worden. Uit één interview kwam naar boven dat PostNL al zo'n soort systeem heeft. Gebruikers zullen er dus al enigszins bekend mee zijn, alleen gaan wij het proberen beter te maken, voor al je post in één app. De afgenomen interviews zijn als bijlage toegevoegd aan dit document.

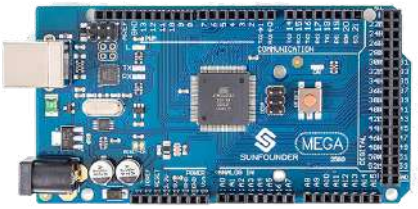
Later in het project hebben we ons bedacht dat het eigenlijk ook erg nuttig zou zijn om te meten hoe vol de brievenbus zit, zo kan de gebruiker zien of het fysiek noodzakelijk is om hem te legen. Dit is te doen door aan het plafond van de brievenbus een ultrasonische sensor naar beneden te richten.

Samenvattend hebben we de volgende sensoren nodig voor de bijbehorende data:

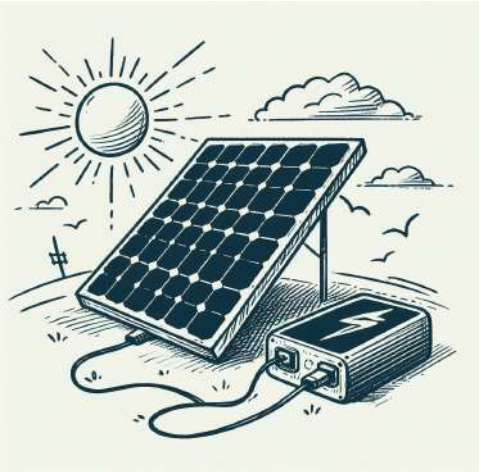
| **Soort data** | **Sensor** |
|---|---|
| Is er iets binnengekomen? | IR nabijheidssensor, Hall Effect sensor, weegschaal |
| Wat is het gewicht van het poststuk? Is het een pakket of een brief? | Weegschaal |
| Is het poststuk belangrijk? Wat zit er allemaal in de brievenbus? | Camera |
| Hoe vol zit de brievenbus? | Ultrasonische sensor |

## Wat voor andere hardware hebben we nodig?

Nu we weten welke sensoren we minimaal nodig hebben om ons doel te kunnen bereiken, zijn we op zoek gegaan naar overige hardware die we nodig hebben. Één van de belangrijkste onderdelen is de microcontroller, waarop alle sensoren worden aangesloten, en eigenlijk fungeert als het hart van onze slimme brievenbus. Omdat wij als studenten al veel gewerkt hebben met Arduino microcontrollers en dus redelijk wat kennis erover in huis hebben, hebben we besloten om ons idee uit te werken op een van de verschillende Arduino microcontrollers. We hebben gekeken naar de verschillende modellen, en dan met name de Arduino NANO, UNO en MEGA. Met het relatief grote aantal sensoren dat we willen aansluiten, zou een Arduino NANO te weinig aansluitingen hebben. Daarnaast heeft een Arduino NANO slechts male pins, die je eigenlijk wel moet aansluiten op een groot breadboard wil je daar goed gebruik van maken. Om ruimte te besparen, willen we dat vermijden. De Arduino modellen UNO en MEGA zijn geschikt om direct hardware op aan te sluiten, en we hadden nog een Arduino MEGA liggen. Met de grote hoeveelheid pins, en het toch redelijk compacte formaat van deze microcontroller, leek ons dit een goede thuis hub waarop we in ieder geval de sensoren konden testen.




Omdat een brievenbus meestal buitenshuis staat, is stroomvoorziening aan de brievenbus een belangrijk aandachtspunt voor dit project. Idealiter willen we dat de brievenbus zichzelf van stroom kan voorzien, dat wil zeggen dat de gebruiker niet bijvoorbeeld zelf elke zoveel maanden batterijen hoeft te vervangen. Er zijn verschillende manieren om natuurlijke energiebronnen te gebruiken voor het opwekken van stroom. Wind- of waterkracht energie zou te complex zijn om te implementeren, vooral omdat het allemaal moet passen in een relatief klein formaat. Vandaar dat zonne-energie ons de beste optie leek, vooral toen we erachter kwamen dat er relatief kleine zonnepanelen beschikbaar zijn op de markt. Aangezien de zon natuurlijk niet altijd schijnt, is het belangrijk dat de brievenbus altijd een reservoir aan stroom heeft voor dit soort momenten. Voor dit reservoir hebben we besloten een powerbank te gebruiken. Deze zijn relatief goedkoop en makkelijk in gebruik. Het zonnepaneel zit dan aangesloten op de powerbank, en de powerbank zit dan aangesloten op de microcontroller. We hadden eerst het idee om een powerbank te gebruiken met een ingebouwd zonnepaneel, maar kwamen erachter dat deze powerbanks te weinig stroom zouden leveren voor de hardware. Daarnaast is dit zonnepaneel makkelijk aan te sluiten via USB, en is niet te groot voor het formaat van de brievenbus. Om erachter te komen hoe goed ons idee zou functioneren, hebben we twee experimenten opgezet, die te maken hebben met het stroomverbruik van de microcontroller, en de opbrengst van het zonnepaneel.



## Stroomverbruik microcontroller

Dit experiment is bedoeld om erachter te komen hoe lang de microcontroller meegaat in twee scenario's: de worst case en de best case. In het best case scenario is de microcontroller aangesloten op de powerbank, maar hoeft het verder niet zoveel te doen. Er is geen hardware aangesloten, en fungeert als een simulatie waarin de Arduino in een soort power-saving mode staat. In de worst case scenario heb ik een 320*240 Pixels TFT scherm aangesloten op de Arduino MEGA. Dit scherm gebruikt bijna alle pins, en heeft hiermee een hoog stroomverbruik. Zulk hoog stroomgebruik zal niet snel voorkomen bij ons eindproduct, maar het is wel belangrijke data om te hebben. Het is natuurlijk te verwachten dat de powerbank het langste meegaat in het beste scenario. Het gaat dan ook voornamelijk om de verschillen in resultaten die interessant zijn. Hier volgen de hypothese voor de best case en worst case scenario's, voor een powerbank van 20.000 mAh:

**Hypothese best case**: de Arduino MEGA gebruikt standaard tussen de 50mA – 75mA. Uitgaande van het meeste stroomverbruik, zou de Arduino MEGA ongeveer 267 uur functioneren op 20.000 mAh.

**Hypothese worst case:** de maximale stroomafname van de Arduino MEGA is 200mA, wat zou betekenen dat de volgeladen powerbank in het slechtste geval ongeveer 100 uur zou moeten meegaan.

Gelukkig konden we dit experiment in de achtergrond laten draaien terwijl we bezig waren met andere dingen, want flink wat uur later zijn dit de resultaten:

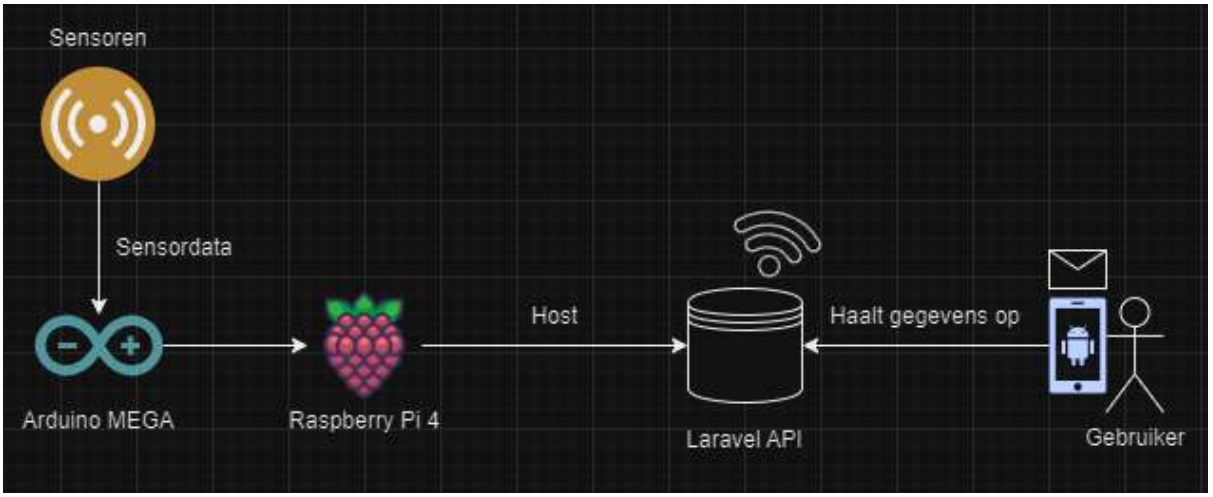
| **Scenario** | **Resultaat** |
|---|---|
| Best case | +- 268 uur |
| Worst case | +- 37,5 uur |

Zoals verwacht is de powerbank veel langer mee gegaan in de best case scenario. Wat opvalt, is dat in het worst case scenario, de powerbank veel minder lang heeft gefunctioneerd dan we hadden verwacht. Het is ons eindproduct dan ook erg noodzakelijk om effectief gebruik te maken van de stroom die beschikbaar is, want het kan het verschil maken tussen een bedrijfstijd van anderhalve dag, of anderhalve week.

## Stroomopwekking zonnepaneel

Om ervoor te zorgen dat de slimme brievenbus 24/7 kan functioneren, is het belangrijk dat er op zonnige dagen stroom wordt opgeslagen, voor gebruik op minder zonnige dagen. Aan de hand van een zonnepaneel en dezelfde powerbank, hebben we getest hoe goed het zonnepaneel onze powerbank kan opladen. Aangezien het herfst was, was het even afwachten op een dag waarop de zon goed scheen, maar uiteindelijk was ie daar. Toen de zon eenmaal scheen op het zonnepaneel, begon de powerbank op te laden. De powerbank heeft vier lampjes die aangeven hoe vol deze is. Aangezien de powerbank een maximale capaciteit van 20.000 mAh heeft, kunnen we ervan uitgaan dat één lampje ongeveer 5000 mAh betekent. De powerbank was helemaal leeg van de test hiervoor, dus er knipperde 1 lampje, om aan te geven dat de powerbank werd opgeladen. Na zo'n vier uur in de zon, begon het tweede lampje met knipperen. We kunnen met deze gegevens concluderen dat het dus ongeveer 4*4 = 16 uur zou duren om de hele powerbank vol te laden, en dus een stroom reservoir te hebben van 20.000 mAh. Op dit reservoir kan de Arduino in het beste geval ongeveer 268 uur mee. We kunnen hieruit dus concluderen dat het zonnepaneel in principe genoeg stroom opwekt om 24/7 te kunnen functioneren, mits het stroom reservoir wordt aangevuld door minstens 1 relatief zonnige dag in een tijdspanne van 11 dagen. Voor ons huidige project functioneert dit goed genoeg, maar het is natuurlijk mogelijk om een powerbank te gebruiken met een nog grotere capaciteit. Als we bijvoorbeeld een powerbank van 100.000 mAh zouden gebruiken, heb je een stroom reservoir wat meer dan een maand mee zou kunnen gaan, en dit kan nuttig zijn voor wintermaanden waarin de zon minder schijnt.

Nu we data hebben over stroomverbruik en opwekking, kunnen we beginnen met het verder uitwerken van ons idee. Het is uiteindelijk de bedoeling dat de sensoren een signaal geven aan de microcontroller, die op haar beurt een API kan updaten, zodat de app hiervan gegevens kan ophalen, om zo aan de gebruiker een notificatie te kunnen geven dat er post binnen is gekomen. De Arduino MEGA heeft geen internet mogelijkheden ingebouwd. Er bestaan Wi-Fi modules die het mogelijk maken dat de Arduino via Wi-Fi kan communiceren, maar eerdere ervaring met zo'n soort module was niet erg goed. De Wi-Fi module kon niet functioneren op 5 of 3.3 volt, en documentatie ervan was erg schaars. We zijn toen naar andere mogelijkheden gaan kijken, en kwamen uiteindelijk op het idee om ook een Raspberry Pi te verwerken in de brievenbus. De Raspberry Pi heeft ingebouwde Wi-Fi, en hier kan direct een microcontroller op worden aangesloten via USB. Daarnaast is deze relatief klein en past dus goed in de brievenbus, en we hebben al ervaring met dit computertje. De Raspberry Pi is dan verantwoordelijk voor het opvangen van de sensordata uit de Arduino MEGA, om deze dan te verwerken in een API en te hosten. Hieronder volgt een schematische tekening van dit systeem:




We hebben ervoor gekozen om Laravel te gebruiken als framework om de API mee te hosten, simpelweg omdat we hier de meeste ervaring mee hebben, en van tevoren al weten dat Laravel ons voldoende functionaliteit zal bieden om ons idee uit te kunnen werken. Zo heeft het namelijk full stack mogelijkheden, en kunnen we de back-end meteen testen in een laravel front-end. Door middel van een MySQL database is het maken van een API redelijk simpel te doen in Laravel.

# Hoe maken we de brievenbus veilig?

Als we het hebben over de veiligheid van de brievenbus, kunnen we deze opdelen in twee categorieën: fysieke veiligheid en digitale veiligheid. Met fysieke veiligheid gaat het erom dat de brievenbus niet makkelijk opengemaakt kan worden door iemand die daar geen recht op heeft. Een fysiek slot zorgt bijvoorbeeld voor beveiliging zodat post niet makkelijk gestolen kan worden. Met digitale veiligheid, hebben we het over hoe de gegevens die de brievenbus ophaalt, worden verwerkt. Het moet voor gerechtigden gemakkelijk zijn om deze gegevens in te zien, maar moet afgeschermd worden voor mensen die hier geen toestemming tot hebben. In dit hoofdstuk gaan we het eerst hebben over deze digitale beveiliging, en daarna de fysieke beveiliging.

## Digitale beveiliging

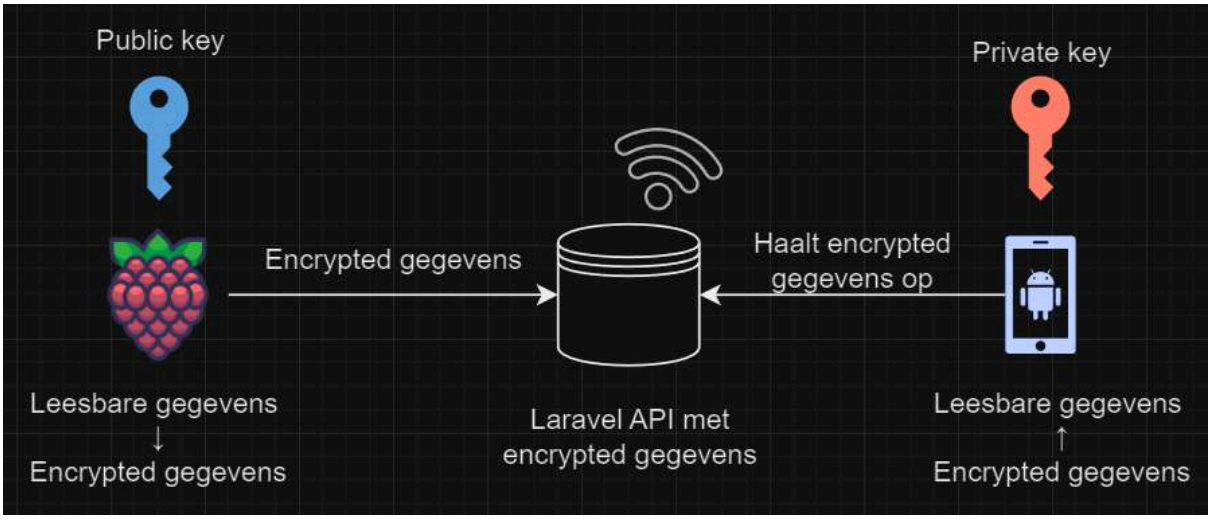
Omdat onze brievenbus veel soorten gegevens ophaalt en bewaard, is het belangrijk dat deze veilig worden opgeslagen. Denk bijvoorbeeld aan adresgegevens die te zien zijn op de foto's die de camera maakt, of boetes die binnenkomen, waarvan je liever niet hebt dat andere mensen die zien.

De API zal lokaal gehost worden op het Wi-Fi netwerk van de gebruiker. Dit zou betekenen dat met standaardconfiguratie iedereen die toegang heeft tot dat desbetreffende Wi-Fi netwerk, gegevens kan ophalen uit die API. Een simpele methode om controle te hebben over wie met je Wi-Fi netwerk kan verbinden, is natuurlijk het hebben van een wifi-wachtwoord. Gelukkig worden Wi-Fi netwerken standaard aangesloten met WPA2 beveiliging, en is een openbaar Wi-Fi netwerk voor thuisgebruik ongehoord. Toch zijn er wel adviezen voor een degelijk sterk (Wi-Fi) wachtwoord:

- Minimaal 10 tot 24 tekens, geen maximum
- Vermijd simpel te raden wachtwoorden, en gebruik ook geen standaard inloggegevens zoals 'admin'
- Variatie tussen tekens zoals: hoofdletters, cijfers en leestekens


Met een wifi- wachtwoord kan er dus in ieder geval onderscheid gemaakt worden tussen mensen die wel en geen verbinding hebben tot het netwerk. Maar ook van de mensen die wel toegang hebben tot het netwerk, wil je onderscheid kunnen maken wie wel en niet bij de gegevens kan komen. Een veelgebruikte manier hiervoor is het vergrendelen van de app met een pincode of wachtwoord. Pas als de gebruiker zich heeft geauthenticeerd met een code, die alleen hij of zij weet, toont de app gevoelige gegevens. Ook kan er authenticatie uitgevoerd worden op basis van iets wat je bent, zoals een vingerafdruk. Op deze manier is in ieder geval de frontend beveiligd, maar zelfs dan zijn we er nog niet.

Als de API gehost wordt op een Wi-Fi netwerk waar je toegang tot hebt is het mogelijk om aan de hand van het IP-adres en de poort waar de API op draait, via een webbrowser toch toegang te krijgen tot database tabellen en gegevens, zonder dat hiervoor de app nodig is. Een oplossing hiervoor is het encrypten van data. Encryptie houdt in dat leesbare tekst door een algoritme wordt gehaald met allerlei wiskundige berekeningen, waardoor het bijna onmogelijk wordt om te achterhalen wat de originele gegevens waren. Bij one-way encryptie is het idee dat je niet meer de originele gegevens kan achterhalen nadat deze encrypt zijn. Daarom hebben wij two-way encryptie nodig, gebruikmakend van een private key en een public key. Het idee is dat door middel van een public key, de Raspberry Pi de gegevens van de API encrypt, waardoor deze niet te lezen is zonder de private key. De private key bevindt zich dan in de app, en wordt gebruikt wanneer een gebruiker zich succesvol geauthenticeerd heeft. Op deze manier is de leesbare data dus ook alleen te zien in de app, door een gebruiker die zich kan authenticeren, omdat de private key alleen te vinden is in de app.




Samenvattend hebben we dus de volgende onderdelen nodig voor een zo veilig mogelijke slimme brievenbus:

- Een beveiligd Wi-Fi netwerk waar de API mee kan verbinden
- Front-end beveiliging in de app in de vorm van authenticatie, bijvoorbeeld via een pincode of biometrie
- Back-end beveiliging in de vorm van encryptie, met behulp van een keypair constructie.

Van de hierboven genoemde punten hebben we er uiteindelijk maar twee kunnen behalen: het veilige Wi-Fi netwerk en de Front-end beveiliging in de app. Normaliter draait de Laravel omgeving en de database op de Pi binnen het eigen netwerk. Dit zorgt ervoor dat de data beschermd is van buitenstaanders door middel van het wifi- wachtwoord. Omdat niet iedereen hier toegang toe heeft, is encryptie van de data niet per se nodig. In onze huidige opstelling (voor het geven van de demo) maken we gebruik van een Virtual Private Server, dit betekent dat iedereen die het IP van onze API heeft de data kan inzien. Hierdoor is de data dus NIET veilig. In dit geval zou encryptie voor het ver- en ontsleutelen van de data gewenst zijn. Helaas hebben wij door de plotselinge groeps verkleining en de herindeling van de taken die daar bij kwamen kijken, geen tijd meer gehad om encryptie op te pakken, en uit te werken in ons project.

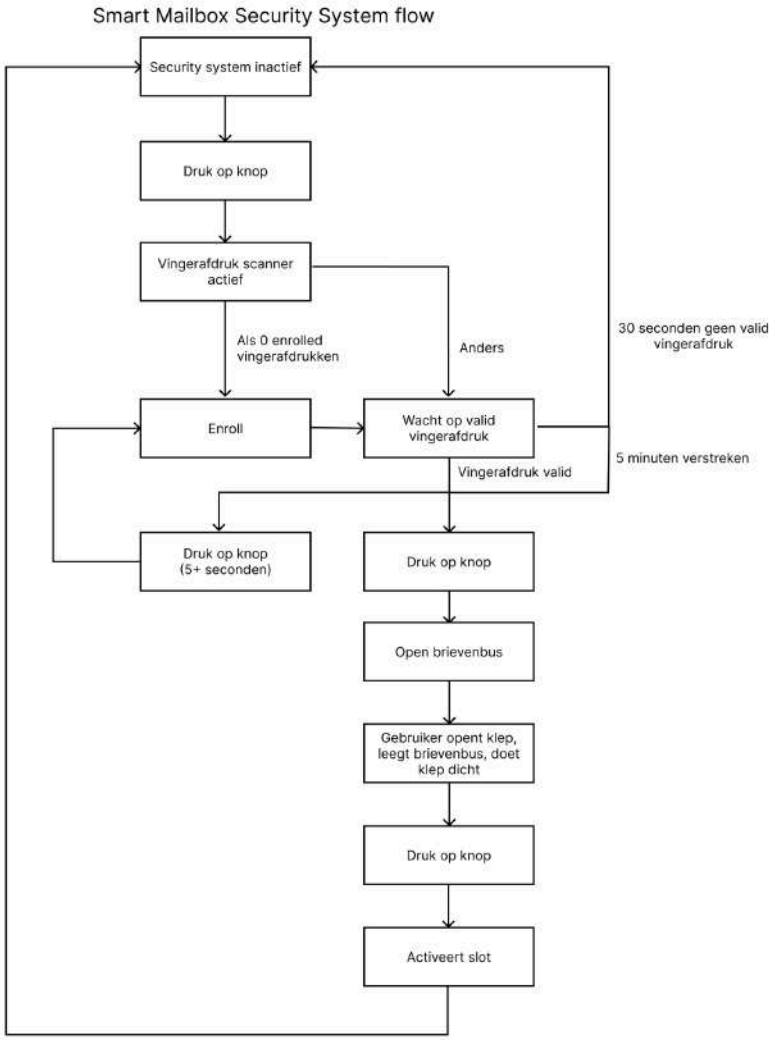
## Fysieke beveiliging

Omdat we van plan zijn een hele brievenbus na te bouwen, is het ook belangrijk om na te denken over de fysieke beveiliging van de brievenbus. Brievenbussen worden natuurlijk beveiligd met een slot, die geopend kan worden met een sleutel die daarbij hoort. We zouden ervoor kunnen kiezen om ook zo'n, niet slim, slot te gebruiken voor onze brievenbus. Maar dat past natuurlijk niet onder de naam van een 'slimme' brievenbus, dus ook het probleem van fysieke beveiliging willen we met technologie gaan oplossen.
Met zo'n slot waar je een sleutel voor nodig hebt, zit je met het probleem dat je de sleutel kan kwijtraken of vergeten. Om dit irritante probleem voor de gebruiker op te lossen, willen we een slot maken op basis van iets wat je niet snel zou kwijtraken, namelijk je vingerafdruk. We hadden eerst het idee om gezichtsherkenning te gebruiken met de ingebouwde camera, maar dit zou veel stroom kosten en je zit met privacy problemen. Daarnaast zou dit niet zo goed werken in het donker. Vingerafdruk is veel sneller en gemakkelijker in gebruik.

Eerst hadden we het idee dat je vanuit de app de brievenbus zou kunnen openen, maar toen bedachten we dat we het sleutelprobleem niet echt zouden oplossen als je alsnog een apart onderdeel nodig hebt om de brievenbus te openen. Je zou je telefoon kunnen vergeten terwijl je naar je brievenbus loopt, of na een lange dag is deze leeg, en moet je eerst helemaal naar binnen om deze op te halen, om vervolgens weer naar buiten te moeten zodat je je post kan ophalen.

Dit kon beter, en we zijn op zoek gegaan naar een vingerafdrukscanner die we zouden kunnen verwerken in de brievenbus zelf. Deze hebben we gevonden, en werkt met een eigen library die goed gedocumenteerd is. Helaas werkte deze scanner niet op een Arduino MEGA 2560 door een andere SPI bus, waardoor de TX/RX pins niet functioneren met deze hardware. We hebben daarom besloten een extra Arduino UNO te verwerken in de brievenbus om de beveiliging te regelen.
Ook hadden we een mechanisme nodig die de deur van de brievenbus kan vergrendelen en ontgrendelen. Hiervoor hebben we gekozen voor een servo motor. Dit is een makkelijk aan te sturen motortje, dat tot een maximale hoek van 180 graden kan draaien, en gebruikt kan worden om het openen van de deur van de brievenbus tegen te houden.

Omdat het vingerafdruksysteem los staat van de app, hebben we een manier nodig om een vingerafdruk te laten registreren, met slechts de hardware van de brievenbus zelf. Om deze reden hebben we besloten om een extra knop toe te voegen aan de brievenbus, om de gebruiker meer controle te kunnen geven. Op de volgende pagina volgt een flowchart met hoe het beveiligingssysteem van de brievenbus in zijn werk gaat en werkt als volgt:




Om stroom te besparen staat het security systeem standaard uit, en wordt er dus niet gecontroleerd op een vingerafdruk. Wanneer een gebruiker op de knop drukt, wordt de vingerafdrukscanner actief en wordt er gekeken of er al eerder een vingerafdruk is opgeslagen. Als dit niet zo is, gaat deze in een 'enroll'-modus, waarin een vingerafdruk geregistreerd kan worden. Als er al eerder een vingerafdruk is ingesteld, wordt er eerst gecontroleerd of deze overeenkomt met de vingerafdruk die op de sensor wordt gelegd. Als dat zo is, kan de brievenbus geopend worden met een druk op de knop. Ook kan er dan gekozen worden om een nieuwe vingerafdruk toe te voegen, door de knop 5 seconde ingedrukt te houden. Dit kan alleen als er al geauthenticeerd is, om te voorkomen dat iedereen zomaar een eigen vingerafdruk kan toevoegen. Om stroom te besparen, wordt het security systeem weer inactief wanneer de brievenbus weer is gesloten, of als er na een halve minuut geen vingerafdruk wordt herkend. Dit geldt ook voor als het systeem nog actief is na 5 minuten nadat de gebruiker toegang heeft gekregen. Dit zorgt voor extra veiligheid voor als de gebruiker vergeet het slot opnieuw te activeren.

Een RGB LED wordt gebruikt voor user feedback, waar een groen licht betekent dat de vingerafdruk succesvol is herkend, blauw dat het op een vinger wacht, oranje om het opnieuw te proberen, en rood dat het herkennen is gefaald.
Hiernaast zorgen we ervoor dat de brievenbus van sterk hout gemaakt wordt, wat niet gemakkelijk kapot gemaakt kan worden.

Door gebruik te maken van slimme technologieën, kunnen we ook het fysieke deel van de brievenbus veilig maken, op een manier die ook nog fijner is voor de gebruiker.

## Hoe moet de brievenbus in elkaar zitten?

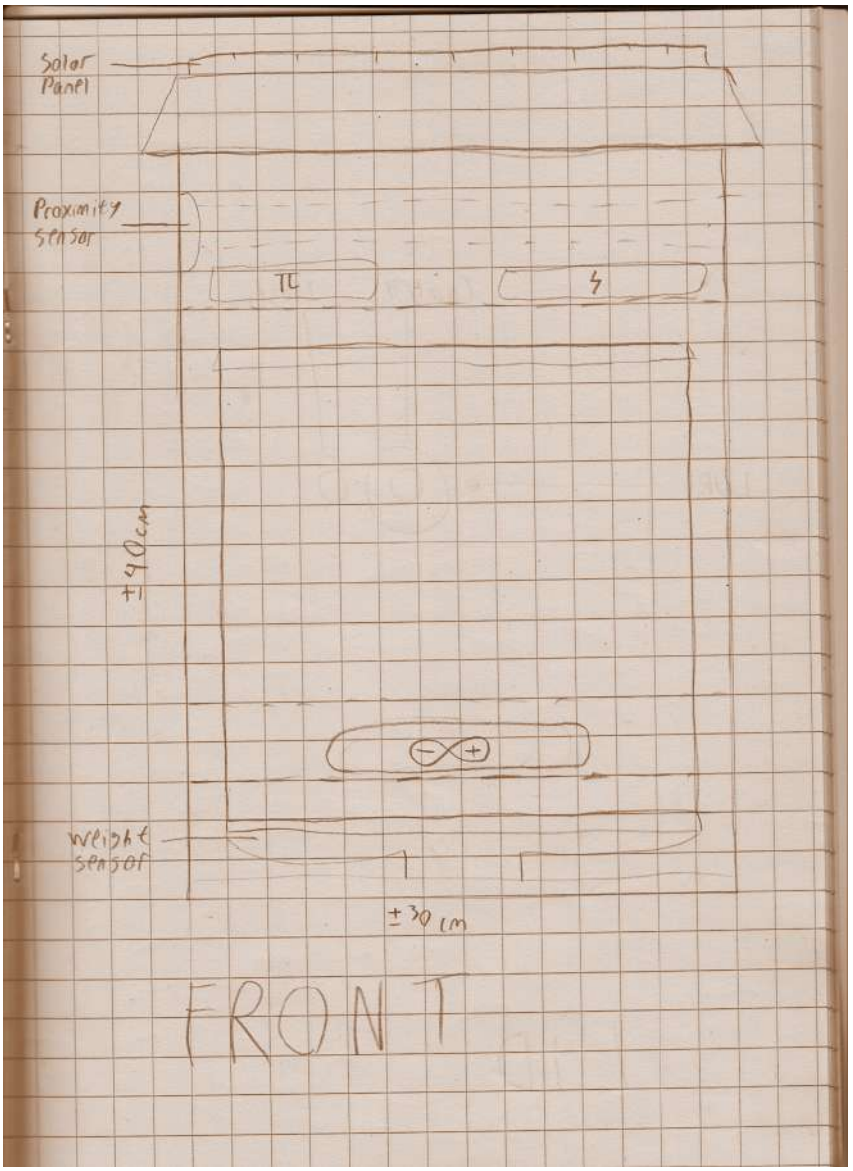
Nu we allerlei informatie hebben verzameld over verschillende sensoren, andere hardware en beveiliging, kunnen we beginnen aan het in elkaar zetten van de eerste versie van onze brievenbus. Voordat we hieraan beginnen, gaan we eerst een competitive analysis doen, om inzicht te krijgen van onze marktpositie met onze slimme brievenbus, tegenover een reguliere brievenbus. Hiermee worden onze sterke en zwakke punten duidelijk, en kunnen hierop anticiperen. De analyse zal gedaan worden aan de hand van 7 belangrijke elementen, en is hieronder te vinden:

| | **Slimme brievenbus** | **Reguliere brievenbus** |
|---|---|---|
| *Kenmerken en Functionaliteiten* | Notificatie systeem, slimme beveiliging, meer gemak voor de gebruiker | Niet veel bijzonders van wat men gewend is. |
| *Prijsstelling* | Moet apart worden aangeschaft, relatief duurder dan reguliere brievenbus. De hardware onderdelen + behuizing kosten zo'n €120. Afhankelijk van de winstmarge, kan de slimme brievenbus 2, 3 of 4x duurder uitvallen dan een reguliere brievenbus | Wordt vaak meegeleverd met huis, prijzen variëren sterk van €35 - €350. |
| *Gebruiksvriendelijkheid* | Verbeterde gebruiksvriendelijkheid dan reguliere brievenbus, door meer gemak functies en geen sleutel om kwijt te raken. | Mindere gebruiksvriendelijkheid door het handmatig controleren van post, en het gebruik van een sleutel voor beveiliging. |
| *Compatibiliteit* | Om van alle functionaliteiten van de brievenbus gebruik te maken, heb je een Android apparaat nodig. IOS is niet compatibel, of mensen zonder smart-apparaat. Daarnaast werkt de brievenbus het beste bij een rijtjes- of vrijstaand huis. | Iedereen met een huis kan gebruik maken van een reguliere brievenbus. |
| *Veiligheid* | Kwetsbaar voor hackers van buitenaf, maar wel slimme en handigere fysieke beveiliging. | Slechts een slot, veilig genoeg mits je de sleutel goed bewaart. Niet kwetsbaar voor digitale beveiligingslekken. |
| *Klantenservice en Ondersteuning* | Wegens de grote hoeveelheid aan technologie in de slimme brievenbus, kan er weleens iets kapot gaan of niet goed werken. Voor een goede klantervaring, is een goede klantenservice en ondersteuning een belangrijk aspect. | Stuk minder aanwezig en nodig omdat een reguliere brievenbus van nature zeer onderhoudsvriendelijk is. Je zult niet snel storingen ervaren. |
| *Toekomstige Ontwikkelingen* | In een gedigitaliseerde wereld is de slimme brievenbus toekomstbestendig, al zal het even kunnen duren voordat de brievenbus op grote schaal wordt gebruikt, vanwege de hogere prijs en het wennen eraan. | Ook in het digitale tijdperk heeft de reguliere brievenbus nog een plek. Voor mensen die de slimme functies niet echt nodig hebben, of de prijs niet willen betalen, is de reguliere brievenbus nog een prima optie. |

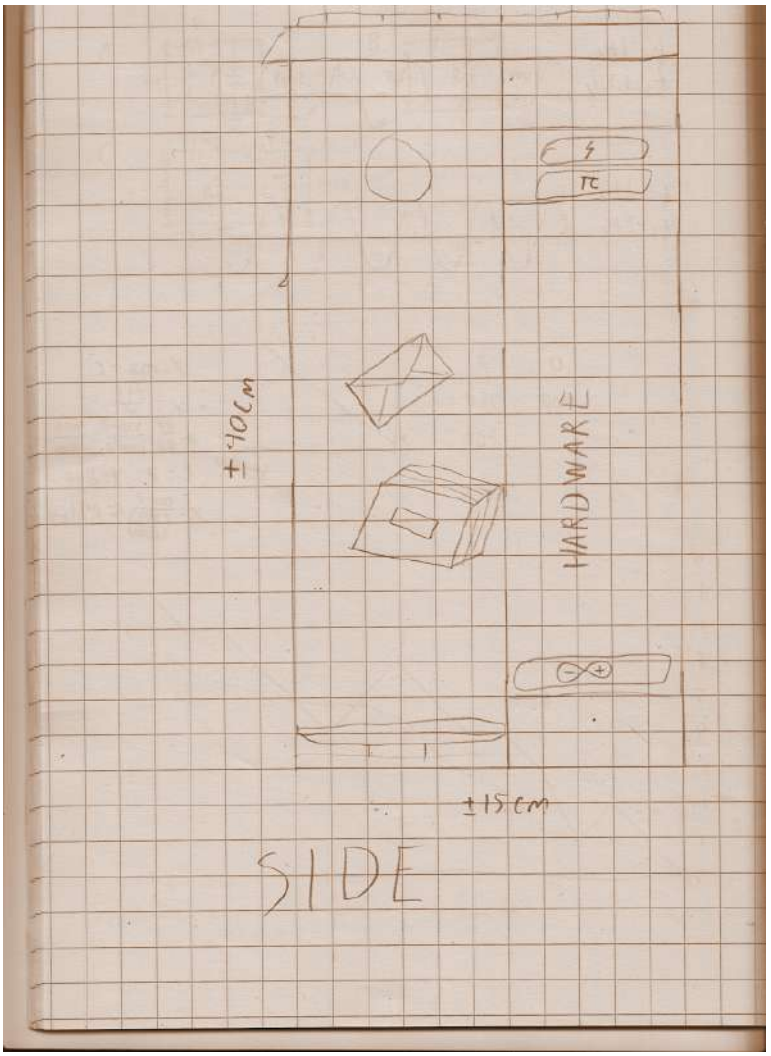
# Het ontwerp

## Het proces

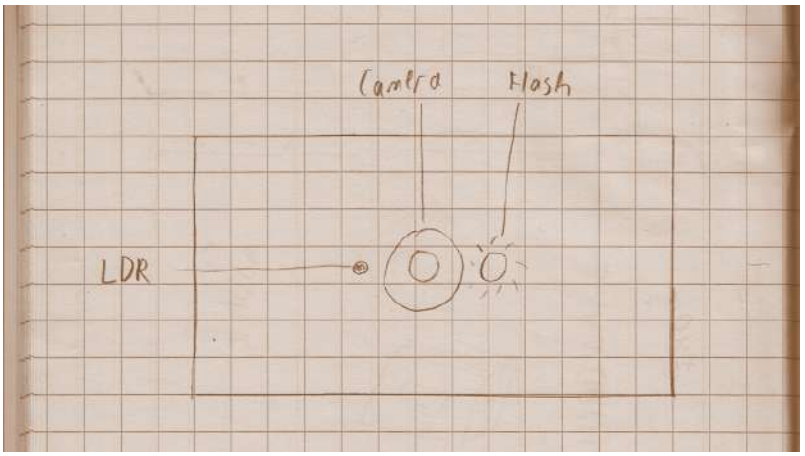
Nu we in kaart hebben gebracht wat onze sterke en zwakke punten zijn, kunnen we gaan beginnen aan het ontwerpen van de brievenbus zelf. Het idee is dat we de brievenbus ontwerpen op basis van een standaard brievenbus die door iedereen herkend wordt, zodat het op het eerste gezicht op een normale brievenbus lijkt. Zo weet iedereen direct hoe deze gebruikt moet worden. Een aantal schetsen van de brievenbus met de hardware erin verwerkt, is hieronder te vinden.




In deze schets zijn de verschillende onderdelen te zien die we eerder besproken hebben, zoals de nabijheidssensor, de weegschaal of de Arduino. Ook is er ruimte voor het zonnepaneel die de brievenbus van stroom zal voorzien. In deze eerste schets is nog te zien dat we van plan waren om aan de bovenkant een klep te gebruiken die open en dicht kan, maar in het definitieve ontwerp hebben we toch gekozen voor een gleuf. De gleuf zorgt er namelijk voor dat poststukken beter te detecteren zullen zijn, door een langzamere val. Daarnaast zorgt het ervoor dat hardware, zoals het zonnepaneel, makkelijker op de bovenkant van de brievenbus kan worden gemonteerd.

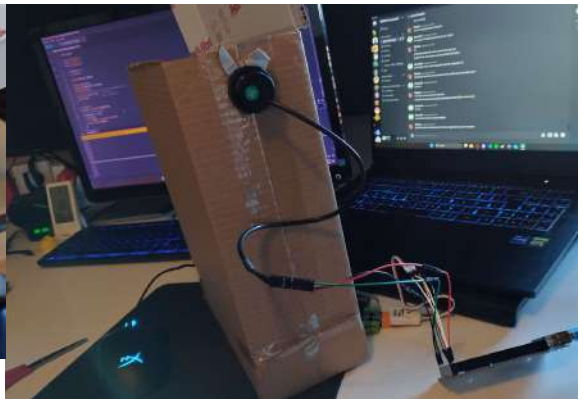



In de zijaanzicht schets is te zien dat we een aparte hardware gedeelte in de brievenbus willen verwerken. Hier kunnen dan de microcontrollers en kabels vandaan komen, voor optimale cablemanagement.




In de schets van de onderkant van de deksel is te zien hoe we de camera willen monteren. Hier is ook een flits nodig zodat de donkere brievenbus verlicht wordt voor de foto. In deze schets was ook nog een LDR te zien, maar deze hebben we uiteindelijk niet nodig gehad, omdat de nabijheidssensor deze functionaliteit kon overnemen.

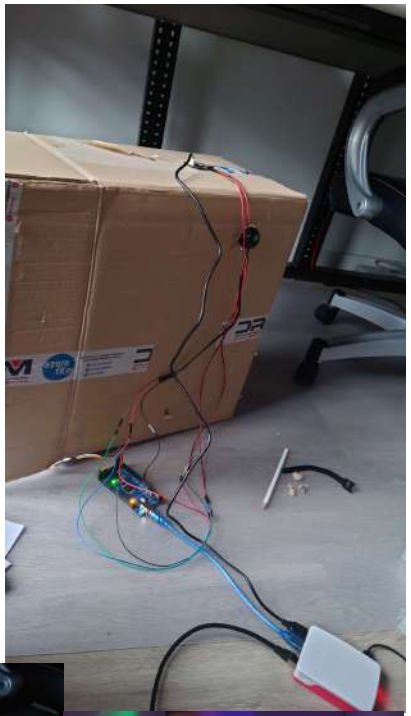
Met deze schets hadden we een goed, globaal idee van hoe de brievenbus eruit moest zien. Ons allereerste prototype was van karton. Dit was, zacht uitgedrukt, nog een erg ruwe representatie van de brievenbus. Dit prototype is vooral gebruikt om de nabijheidssensor en de weegschaal te testen door simpelweg objecten erin te laten vallen.

*De allereerste iteratie van het prototype*

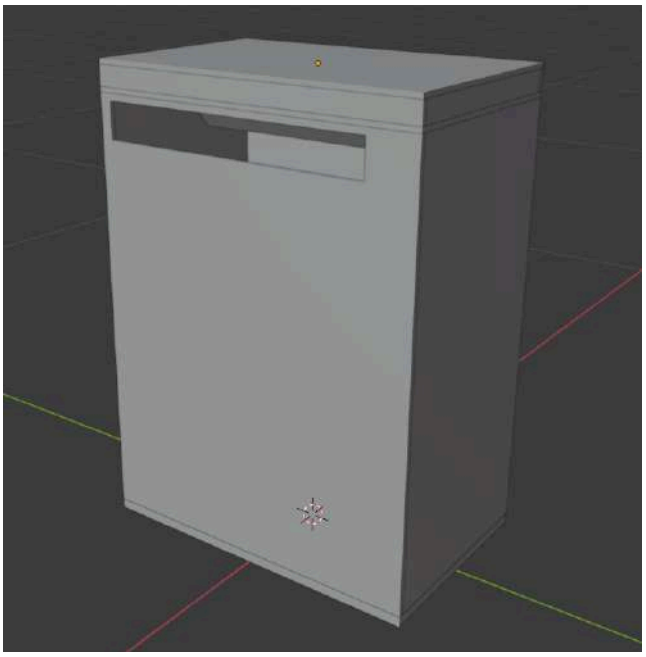
Dit kartonnen prototype werkte prima om snel te testen. Zo kwamen we er al snel achter dat de nabijheidssensor niet snel genoeg is om een vallend object te detecteren, wat ons tot de beslissing bracht om geen bovenklep te doen, en het inbrengen van de post dus alleen via een gleuf te doen.

Het tweede prototype was ook van karton, maar dit keer waren we al enkele weken ouder en wijzer. Deze tweede iteratie is in de afbeeldingen rechts te zien. Op dit punt was het idee van de bovenklep al geschrapt.



Toen we het prototype in elkaar hadden gezet, zijn we begonnen met testen. Het testen ging aardig goed, zowel met een brief als een pakket. De nabijheidssensor detecteerde goed dat er een poststuk in de brievenbus werd gegooid, waardoor de camera een foto maakte.
Wel kwamen we erachter dat de RGB LED module die we als flits gebruikten niet erg goed werkte. De LED kon niet goed één witte kleur van licht geven, waardoor het op een soort disco in de brievenbus leek en de foto's niet erg mooi waren. Om deze reden hebben we een andere flits besteld.
Ook hadden we tot dusver nog niet echt nagedacht over de kleur van de brievenbus, maar na dit prototype weten we dat de binnenkant in ieder geval wit moet zijn, zodat het licht van de flits zoveel mogelijk weerkaatst kan worden voor het beste foto resultaat.


Toen we eenmaal twee kartonnen prototypes hadden getest, hebben we ons idee gedigitaliseerd aan de hand van een 3D model, gemaakt in blender.

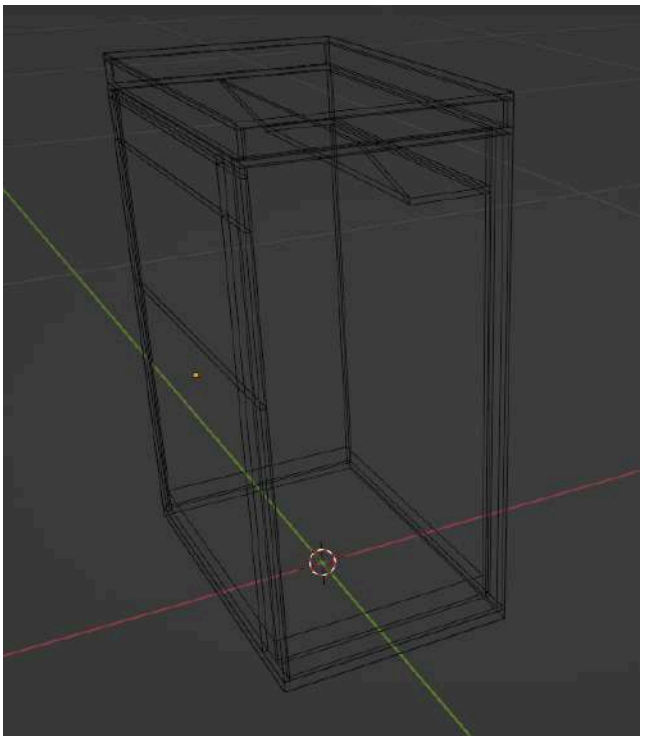


Tijdens het modelleren, kregen we de ingeving om het compartiment voor de hardware een dubbelrol te geven en de muur aan de gleuf kant schuin te maken. Naast het behuizen van de hardware, zou het er zo ook voor kunnen zorgen dat binnenkomende post automatisch naar beneden wordt gedrukt. Dit zorgt dan voor betere detectie door de nabijheidssensor, omdat deze dan in de schuine plank met een hoek naar beneden op het poststuk kan richten, dat dan ook langzamer en verticaal de brievenbus in valt.

Dit 3D model is handig om te hebben als referentiepunt. Het is op een manier ontworpen zodat het goed met houten planken na te maken. Dit is omdat we iemand kennen (Martins vader) die goed is in houtbewerking en heeft aangeboden om ons te helpen dit model te verwerkelijken.

Dit ontwerp is nog enigszins ruw. Het uitzagen van de brievenbus voor onder andere de gleuf, sensoren, de klep en overige hardware is iets dat altijd nog na de constructie van dit basismodel kan.

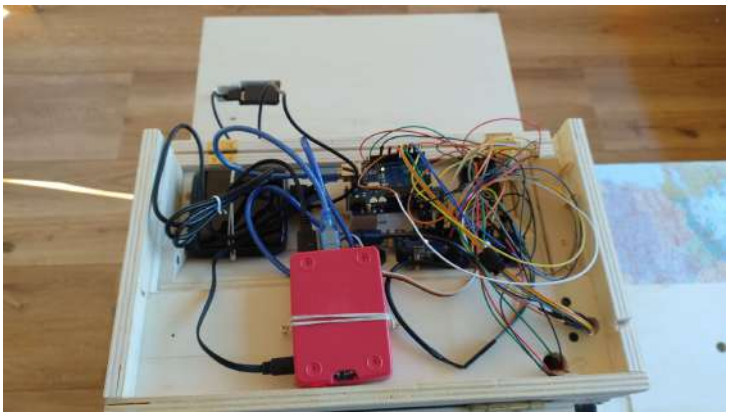
Zoals op de afbeelding rechts te zien, is er boven in de brievenbus een compartiment voor de hardware. Daarboven bevindt zich ook nog een extra “verdieping” om zo het zonnepaneel mooi op het dak te kunnen bevestigen. Ook is er aan de zijkant een compartiment voor bedrading die externe hardware zoals de weegschaal en de vingerafdruksensor moet verbinden met de microcontrollers bovenin. Uiteraard is dit met een plank gescheiden zodat het niet in aanraking komt met de inhoud van de brievenbus.



## Het eindresultaat

Na een paar dagen constructie van dit ontwerp, en het achteraf finetunen zodat de hardware en sensoren er goed in gemonteerd konden worden, hadden we eindelijk een definitieve brievenbus! Als groep zijn we erg dankbaar met de hulp die we hiervoor gekregen hebben en we zijn super blij met het eindresultaat. Hier zullen we beschrijven, maar vooral laten zien hoe de brievenbus in elkaar zit.

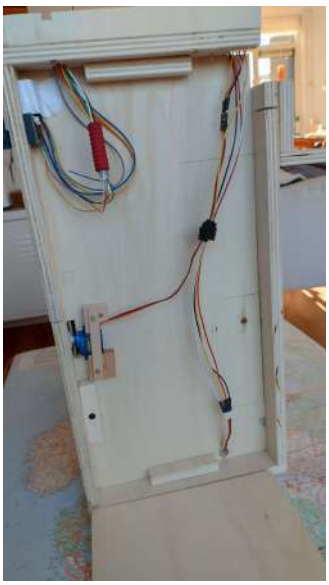
Op de eerste afbeelding zie je hoe de brievenbus in zijn normale staat eruitziet. De vingerafdruksensor bevindt zich rechts van de klep, de bedrading ervan komt dan handig in het zij-compartiment uit. Ook is dit een logische plek voor de bediening ervan. Gebruikers zullen geen moeite hebben om hun vinger hier te plaatsen. Boven de scanner zit de RGB led, op de foto is dit moeilijk te zien. Ook deze is hier geplaatst, ten eerste zodat de bedrading weer handig uitkomt, en ten tweede omdat het zo voor de gebruiker duidelijk is dat dit licht met de vingerafdrukscanner te maken heeft. De knop voor het activeren van het security systeem bevindt zich mooi in het midden van de brievenbus en dicht in de buurt van de klep voor het openen van de brievenbus. Het idee is dat de plaatsing van de knop zo subtiel aangeeft dat het er is voor het openen van de klep.



Hier rechts is de hardware compartiment te zien. De bovenste klep waar het zonnepaneel op bevestigd is gaat makkelijk open zodat we makkelijk bij de hardware kunnen. Het ziet eruit als een rommeltje, maar al met al is dit nog redelijk geoptimaliseerd gezien de hoeveelheid bedrading die hier nodig is.

De kabel van het zonnepaneel gaat naar de powerbank helemaal links. Die ook weer een kabel heeft die naar de Raspberry Pi (onderaan de afbeelding en ondersteboven) gaat om het stroom te geven. De Arduino Mega is enigszins verscholen aan de rechterkant te zien en staat op de schuine plank gemonteerd. De Arduino Uno staat op een verhoging, wat handig is voor de ruimte marge van de sensoren en actuatoren die hieronder zitten: de camera, flits, en ultrasonic sensor. Ook de proximity sensor zit in de schuine plank gemonteerd. Rechtsboven kan je als je goed kijkt een zwart breadboard zien, dat gebruikt wordt voor extra 5V en ground pins. Er zijn meerdere plekken waar bedrading doorheen komt. Rechtsboven voor de servo en de weegschaal, en dan rechtsonder voor de RGB led en de vingerafdruksensor. Linksonder daarvan is nog een gat te zien, dit leidt naar de binnenkant van de brievenbus en is voor de drukknop en Hall Effect sensor. Uiteraard is deze bedrading zorgvuldig aan de muur van de brievenbus geplakt, zodat er geen kans is dat het in contact komt met post, of misschien zelfs als post wordt gedetecteerd door de sensoren.

Vanuit de binnenkant van de brievenbus zien de sensoren in het hardware compartiment er zo uit:



Hier links is goed de binnenkant van het zij-compartiment te zien. De plank die dit normaal bedekt is erg makkelijk en snel los te maken voor eventuele aanpassingen. De bedrading van de RGB led, vingerafdruksensor, servo, en weegschaal is hier te zien.

De servo is op een manier bevestigd zodat het 90 graden heen en terug kan draaien, en zo de klep vast kan zetten of los kan laten. Dit is op de rechter afbeelding beter te zien.

Als laatste is hier rechts te zien hoe de brievenbus eruit ziet met beide kleppen open. Helemaal rechtsonder zie je een kleine constructie waar de servo tussenkomt om de klep dicht te houden. Met extreem goede ogen is wellicht ook te zien dat er op de achtermuur een plastic laag zit met ijsstokjes onderin. Dit zorgt ervoor dat post netjes naar beneden “glijdt” en door de ijsstokjes wordt voorkomen dat de post tussen de muur en de weegschaal vast komt te zitten.

Links is een ingezoomde afbeelding van het gat waar de Hall Effect sensor in zit, samen met het magneetje in de klep. Als de klep dicht is, wordt de sensor dus geactiveerd.

## Hardware en sensoren

In dit hoofdstuk van de deelvraag “Hoe moet de brievenbus in elkaar zitten?” bespreken we de verschillende stukken hardware die we hebben aangeschaft en hoe deze met elkaar verbonden zijn. Ook vertellen we kort waarom we deze hardware gekozen hebben en hoe we van plan zijn deze te gebruiken.

### Specificaties en uitleg

**Weegschaal**

M5 Scale Kit - https://docs.m5stack.com/en/app/scales_kit


- Total weighing range of 200kgs
- HX711:
    - High precision 24bit ADC
    - Programmable gain amplification 32, 64 and 128
    - 10SPS output data rate
- Half-bridge resistive strain gauge.
    - Output sensitivity: 1.0±0.1mV/V
    - Non-linearity: 0.3% F.S
    - Integrated Accuracy: 0.3%F.S
    - Zero output: ±0.3mv/V
    - Difference between upper and lower impedance of each strain gauge: 0.8Ω
    - Output (in) impedance: 1000±5Ω

De weegschaal is het belangrijkste onderdeel voor ons project wat sensoren betreft. Deze zal onder in de brievenbus zitten om het gewicht van de inhoud te meten. Zo willen we per nieuwe bezorging achterhalen hoe zwaar het was, en of we te maken hebben met een brief of met een pakketje.

De M5 Weegschaal Kit leek ons een goede kandidaat. Met een bereik van 200 kilo wisten we zeker dat het alles zou kunnen opvangen wat er in de brievenbus gegooid wordt. Er is een library voor waardoor we relatief makkelijk de weegschaal kunnen configureren in onze Arduino code. Ook leek het ons erg handig dat de weegschaal bestaat uit vier units, het idee is dus om hier een plaat op te zetten. Hier kunnen we zelf kiezen hoe groot deze plaat is, wat ideaal is omdat we voor de weegschaal dan geen rekening met afmetingen hoeven te houden bij het maken van de brievenbus.

**Proximity Sensor**
DFRobot Gravity Digitale IR Nabijheidssensor - 0-200cm -
https://wiki.dfrobot.com/Digital_IR_Proximity_Sensor_0_200cm_SKU_SEN0381

- Power Supply: 5V
- Detection Distance: 0~200cm(adjustable)
- Output: switch quantity(active-high)
- Detection Angle: 30-40° coning angle
- Detection Mode: active
- Operating Temperature: -10~60°C
- Storage Temperature: -20°~70°C
- Waterproof Performance: IP67

Dit leek ons een prima kandidaat voor wat wij met de sensor willen bereiken. 200 cm bereik is meer dan genoeg. Daarbij kan je zelf de detection distance aanpassen met een druk op de knop van de achterkant van de sensor. Dit houdt de bediening ervan simpel. Om het werkend te krijgen hoeven we dus alleen maar de sensor in de brievenbus te monteren en op die knop te drukken. Als de sensor dan een object binnen die afstand detecteert geeft hij als output 1, en anders 0.

**Hall Effect Sensor**
Hall Effect Switch Module -
https://www.tinytronics.nl/shop/en/sensors/magnetic-field/hall-effect-switch-module

- Supply voltage: 5V DC
- Signal voltage: 5V
- Chip: A3144, 44E or SS49E*

Dit leek ons een goede Hall Effect sensor. Omdat de module al een Hall Effect switch op het bord heeft gesoldeerd, hoeven we niet veel zelf aan te passen. Een Hall Effect sensor detecteert een magnetisch veld, en met deze sensor betekent dit dat hij 1 als output geeft met een magneet in de buurt, en anders 0.

Wij willen deze sensor in samenwerking met de proximity sensor laten werken. Het idee is dat we een magneet in de klep van de brievenbus monteren, zodat de sensor een verschil kan detecteren tussen een open of dichte klep. Een dichte klep geeft dan natuurlijk 1 als output. Op het moment dat de proximity sensor een poststuk niet detecteert, ook al is die er wel degelijk, dan kan de Hall Effect sensor alsnog detecteren dat de klep open is geweest, en als er dan gewicht in de brievenbus bij is gekomen, kan onze brievenbus dit alsnog bevestigen en doorgeven.

**Ultrasonische Sensor**
Ultrasonische Sensor - HY-SRF05 -
https://www.tinytronics.nl/shop/nl/sensoren/afstand/ultrasonische-sensor-hy-srf05

- Detectiebereik: 2-450cm
- Spanning: 5V
- Stroom: <2mA
- Resolutie: 3mm
- Sensor hoek: <15o
- Afmetingen: 45x20mm

Deze sensor leek ons goed voor ons project door het hoge detectiebereik, we weten hierdoor zeker dat het door de hele brievenbus kan scannen. Deze sensor gebruikt sonar om de afstand tot een object te bepalen, dit zendt hij uit, en als dit terugkaatst, vangt het het weer op. Door de tijdsperiode tussen deze tijd te meten kan je vervolgens in de code berekenen wat de gemeten afstand was.

Dit is een sensor die we halverwege het project hebben aangeschaft. Uit het niets kregen we het idee dat het ook nuttig zou zijn om de ‘volheid’ van de brievenbus te meten. Dit willen we doen door de sensor in het plafond van de brievenbus te monteren en naar beneden te richten. Als de post zich dan ophoopt kunnen we achterhalen wat de gemeten afstand naar de bodem is, en vergelijken met de maximale afstand. Zo kunnen we een percentage van volheid achterhalen en die aan de gebruiker doorgeven via de app.

**Camera**
0.3 MegaPixels USB Camera for Raspberry Pi / NVIDIA Jetson Nano / UNIHIKER - https://www.dfrobot.com/product-2089.html

- Ingang: USB port
- Resolutie: 30W, 640 x 480px
- Afmetingen: 30 x 25 x 21.4mm

Deze camera zullen we gebruiken voor het maken van de foto’s van de binnenkant van de brievenbus. Het werkt goed met de Raspberry Pi en kan eenvoudig met USB erop aangesloten worden. De resolutie lijkt misschien niet zo hoog, maar is goed voor onze applicatie, waar de foto’s toch niet zo groot weergegeven zullen worden, maar toch genoeg detail in te zien is. Daarbij geeft de beschrijving aan dat de camera ideaal is voor face en image recognition, iets waar we potentieel nog iets mee willen doen in dit project.

**Flits**

M5Stack UNIT FlashLight - https://docs.m5stack.com/en/unit/UNIT%20FlashLight


- PWM Brightness Adjustment Method
- Over-Temperature Protection
- Over-Voltage and Short-Circuit Protection
- Color temperature: 5000-5700K

Dit is de flitsmodule die we willen gebruiken voor het belichten van de brievenbus inhoud, zodat er een mooie foto van gemaakt kan worden. Deze flits doet wat we willen dat het doet en meer, en was niet duur.

**Vingerafdruksensor**

GROW R307S Fingerprint Module - https://www.otronic.nl/nl/vingerafdruk-fingerprint-sensor.html

- Type: Optisch
- Interface: USB1.1/UART (TTL logisch niveau)
- Resolutie: 500 DPI
- Werkstroom: ≤75 mA
- Spanning: DC 4.2-6.0V (of 3.3V)
- Vingerafdrukcapaciteit: 1000
- Afmeting: 52 *20*22 (mm)
- Beeldvastleggingsoppervlak: 15*11 (mm)
- Achtergrondverlichting: Blauw
- Sensorlevensduur: 100 miljoen keer
- Statische indicaties: 15KV
- Tekenbestand Grootte: 256 bytes
- Sjabloongrootte: 512 bytes
- Beveiligingsniveau: 5 (1,2,3,4,5(hoogste))
- Scansnelheid: < 0,3 seconde
- Verificatiesnelheid: <0,2 seconde
- Overeenkomende methode: 1: N
- FRR (valse afwijzingsratio): ≤0,1%
- FAR (Valse Acceptatie Ratio): ≤0.0001%
- Bedrijfsomgeving Temperatuur: -20°C ---50°C
- RS232 communicatie baudrate: 9.600BPS~115.200BPS (Standaard op 6 = 57.600)

Dit is misschien het meest interessante stuk hardware dat we in ons project gebruiken. We hebben het gekocht van de Otronic website, maar er was hier verrassend weinig (correcte) informatie te vinden. Bij verder onderzoek zijn we erachter gekomen dat deze sensor door GROW gemaakt is, een bedrijf uit China. Er zijn verrassend veel modellen gemaakt, en dit was het R307S model. Van dit specifieke model hebben we later de handleiding als PDF bestand gevonden.

Deze sensor maakt voor het coderen gebruik van de Adafruit-Fingerprint-Sensor-Library, en werkt via Serial communicatie op de Arduino. Omdat we nog nooit een vingerafdruksensor hebben gebruikt wisten we bij de aankoop niet precies waar we naar moesten kijken. We zagen dat deze niet al te duur was, 1000 vingerafdrukken kan opslaan, en 100 miljoen keer een vingerafdruk kan scannen. Dit leek ons dus een prima optie voor ons project. Het idee is dat deze sensor de inhoud van de brievenbus beveiligt, zodat alleen geautoriseerde personen de brievenbus kunnen openen en legen.

**Drukknop**
Blauwe Drukknop 12mm - Reset - PBS-33B -
https://www.tinytronics.nl/shop/nl/schakelaars/manuele-schakelaars/drukknoppen-en-schakelaars/blauwe-drukknop-12mm-reset-pbs-33b

Deze drukknop hebben we aangeschaft voor het activeren van het security systeem. We vonden het belangrijk dat de knop er mooi uitzag en de aandacht trekt van de gebruiker, daarbij is het een fijne knop om in te drukken. In theorie zou hier namelijk vaak op gedrukt moeten worden. Het is een erg simpele drukknop, wat we ook belangrijk vonden. Er hoeven maar twee pins gebruikt te worden. Wel moesten we de twee uiteinden van de knop aan draden solderen voordat we er gebruik van konden maken.

**Servo**
SG90 Mini Servo -
https://www.tinytronics.nl/shop/nl/mechanica-en-actuatoren/motoren/servomotoren/sg90-mini-servo

Dit is een goedkoop Servomotortje dat een van onze groepsleden nog thuis had liggen. Het kan 180 graden draaien en met de bijgeleverde onderdelen die we erop kunnen zetten, is dit een prima optie voor het slotmechanisme van de brievenbus. Op 4.8 volt kan het zo'n 1.3 kilo dragen. Aangezien wij 5 volt zullen gebruiken gaan we ervan uit dat dit voor ons geldt. We willen de servo gebruiken om de plank die als klep van de brievenbus functioneert vast te houden. Dit zou dus goed moeten lukken, de plank draagt zichzelf namelijk voor het grootste deel, en de servo hoeft er alleen voor te zorgen dat het niet naar beneden valt of open gerukt kan worden.

**RGB LED**
RGB LED Module 3.3-5V -
https://www.tinytronics.nl/shop/en/lighting/rings-and-modules/rgb-led-module-3.3-5v

Dit onderdeel spreekt redelijk voor zich. Door verschillende RGB waardes door (PWM) pins te sturen kan je makkelijk bepaalde kleuren licht door de led laten schijnen. Wij willen dit gebruiken om feedback aan de gebruiker door te geven tijdens het gebruiken van de vingerafdruksensor.

**Zonnepaneel**
Zonnepaneel met DC-DC converter en USB - 5V 1A - https://www.tinytronics.nl/shop/nl/power/zonne-energie/zonnepanelen/zonnepaneel-met-dc-dc-converter-en-usb-5v-1a

- Uitgangspanning: ~5V (varieert door de stroom en afhankelijk van sterkte van zon)
- Stroom(max): ~1A
- Afmetingen: 275 x 170 x 2mm (USB aansluiting met DC-DC converter niet meegerekend)
- Is spatwaterdicht op het USB gedeelte na
- Type zonnecellen: monokristallijn

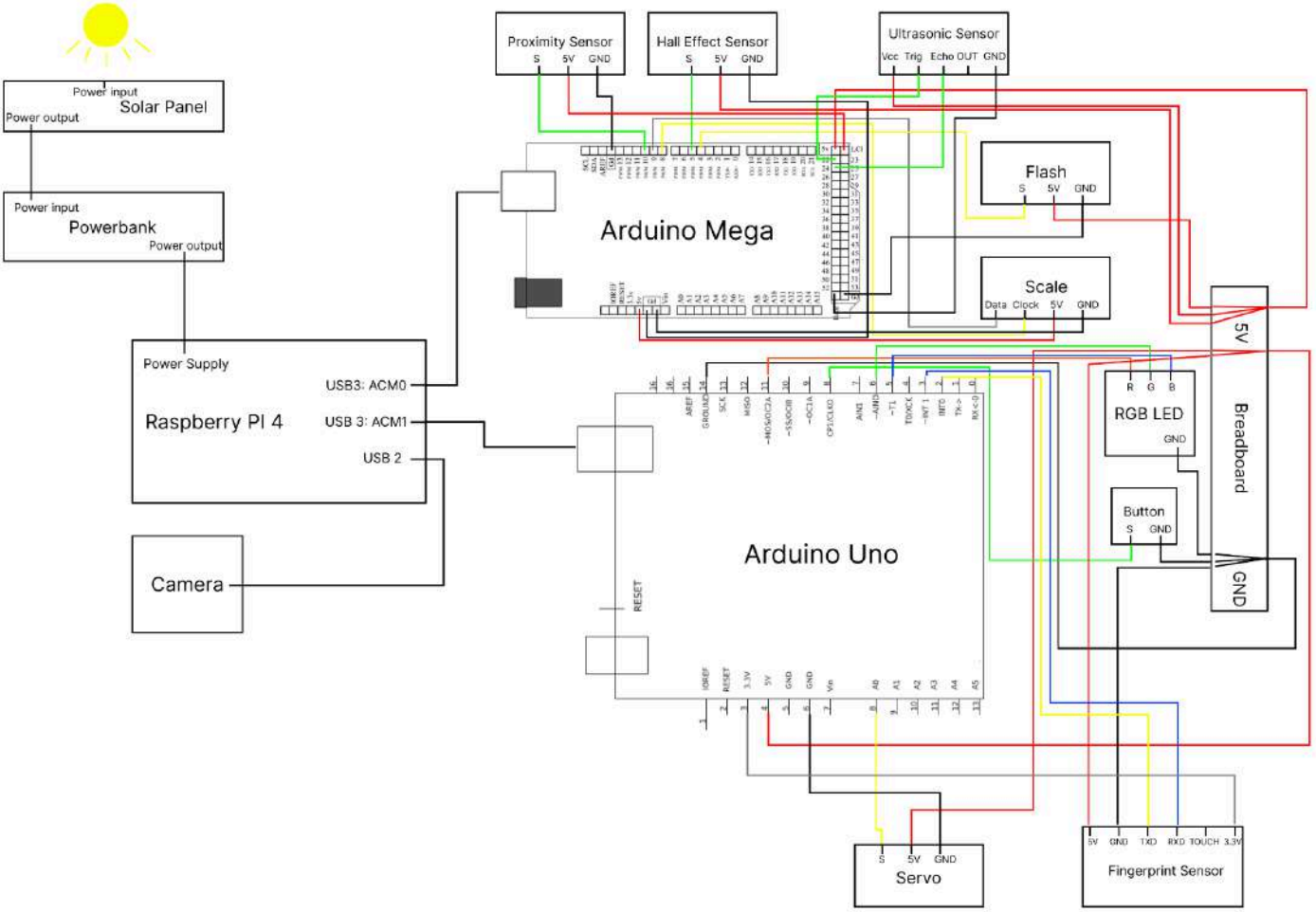
Om ervoor te zorgen dat de hardware binnen de brievenbus altijd voldoende stroom heeft, hebben we dit zonnepaneel aangeschaft. Het gebruik ervan is erg handig. Er zit een rood lampje op die gaat schijnen wanneer het zonne energie opvangt. Ook is het erg simpel met een USB kabel aan de powerbank aan te sluiten. Het formaat komt goed uit om op het dak van de brievenbus te monteren, wat met de vier gaten in de hoeken ook makkelijk vast te zetten is. Het zonnepaneel zal natuurlijk altijd naar boven gericht zijn, en is dus spatwaterdicht.

## De sensor architectuur

De hoofdbasis voor de sensoren is de Raspberry Pi. Hier hebben we twee microcontrollers aan zitten; de Arduino Mega en de Arduino Uno.

De Arduino Uno verzorgt het (fysieke) security systeem van de brievenbus. Het werk dat dit systeem doet is slechts het verzorgen van de veiligheid zodat alleen mensen met een geregistreerde vingerafdruk de brievenbus kunnen openen, dichten, en eventueel andere mensen toegang geven door hun vingerafdruk te laten registreren. Dit deel van de architectuur doet dus niks met de data en is volledig onafhankelijk van de Arduino Mega en zelfs van de Raspberry Pi, de Uno heeft slechts stroom nodig om te functioneren.

De Arduino Mega verzorgt het inhoudelijk functionele deel van de brievenbus. Dit is dus waar alle sensoren aan vast zitten die belangrijke data meten en die doorgeven aan de Raspberry Pi met behulp van een Python script. Hieronder een schematische weergave van onze hardware architectuur:



## De technische werking

De data die de sensoren uitleest moet natuurlijk een bepaald proces doorgaan om zo op een mooie en nuttige wijze in de app terug te komen. Hier zullen we bespreken wat er precies in de brievenbus (en daarbuiten) gebeurt om dit te regelen. Ook is het nuttig om hier te vermelden dat we de belangrijkste bestanden wat de code betreft in de bijlage hebben staan.

### Microcontrollers

**Arduino Uno**
De Arduino Uno verzorgt bij ons het security systeem. Dit systeem is verantwoordelijk voor het beveiligen van de inhoud van de brievenbus, zodat niet zomaar iedereen de post van de gebruiker kan ophalen. Dit deel van de brievenbus is volledig onafhankelijk van de functionele data pipeline, en heeft dus geen invloed op de rest van de architectuur. Eerder in het verslag hebben we al de flow van dit systeem uitgelegd.

De werking van de code ziet er ingewikkeld uit, maar is best goed te begrijpen. Ook moet gezegd worden dat een groot deel van de fingerprint scanner functies zijn overgenomen uit de documentatie van de bijbehorende library en zodanig aangepast dat het functioneert zoals wij voor ogen hadden.

Het eerste deel van de loop in de code zorgt voor het succesvol uitlezen van een druk op de knop. Het is hier belangrijk om onderscheid te maken tussen een korte druk en een lange druk (5+ seconden). Dit bepaalt namelijk de reactie van het systeem; een lange druk op de knop terwijl de gebruiker op dat moment al toegang heeft leidt tot het kunnen registreren (enrollen) van een nieuwe vingerafdruk. Een korte druk op de knop kan meerdere functies hebben, voornamelijk het activeren van de vingerafdruk sensor, en bij een succesvolle scan het openen en dichten van het slot.

In het geval dat, in theoretische zin, de gebruiker net de brievenbus heeft aangeschaft en er dus geen vingerafdrukken geregistreerd zijn in de sensor, dan gaat de scanner automatisch in enroll mode bij de eerste druk op de knop. Hierna is enrollen alleen mogelijk door eerst een geldige vingerafdruk te scannen en dan de knop lang ingedrukt te houden. De voorbeeldcode van de library was zo ingesteld dat de gebruiker zelf het ID moest invullen van zijn vingerafdruk, dit hebben we dus ook aangepast zodat dit niet nodig is, en het ID van een nieuwe vingerafdruk altijd de hoeveelheid geregistreerde vingerafdrukken + 1 zal zijn.

De code voor het verifiëren of enrollen van een vingerafdruk hebben we voor het grootste gedeelte hetzelfde gelaten. Wel hebben we hier aan toegevoegd dat de gebruiker gekleurd licht te zien krijgt. In de code zit een ‘showFeedback()’ functie die op basis van de status van het proces de RGB led een bepaalde kleur laat schijnen. De statussen en bijbehorende feedback zijn:

- SUCCESS
    - Proces is gelukt of gaat goed -> led knippert groen
    - Bij het enrollen gebeurt dit twee keer, omdat de nieuwe vingerafdruk twee keer gescand moet worden
- TRY_AGAIN
    - Proces is niet gelukt maar kan gelijk opnieuw geprobeerd worden -> led knippert oranje
    - In dit geval kan de gebruiker gelijk weer zijn vinger op de scanner doen
- FAILED
    - Proces is fout gegaan -> led knippert rood
    - Het volledige proces wordt geannuleerd, er moet dus opnieuw op de knop gedrukt worden om het opnieuw te proberen
- WAITING
    - Scanner wacht op vingerafdruk -> led knippert blauw
    - Dit geeft aan dat de scanner gereed is om een vingerafdruk te scannen

Een probleem dat we tegen zijn gekomen is dat de led van de vingerafdruk sensor zelf (de backlight) niet consistent is. Soms is het security systeem niet actief, maar dan blijft de backlight alsnog aan staan. Bij het scannen van een vingerafdruk is deze normaal aan het knipperen. We hebben gekeken of we dit konden verbeteren maar met weinig succes. We weten dus niet zeker of we er wat aan kunnen doen of niet.

Een ander ding dat opviel was dat, tijdens het scannen van een vingerafdruk, de servo een beetje aan het ‘trillen’ was. Dit is echter geen probleem met hoe we de servo gemonteerd hebben. Ook in dit geval weten we niet of we hier daadwerkelijk wat aan kunnen doen.

**Arduino Mega**
De Arduino Mega zorgt voor het aflezen en versturen van de data naar de Raspberry Pi. Hierbij moet alvast verteld worden dat er op de Raspberry Pi een Python script draait. Deze luistert naar de Serial poort waar de Mega op aangesloten zit, en zo kan het via Serial prints de data van de microcontroller aflezen.

De grootste focus zit hier op het gewicht. Onze weegschaal heeft een eigen library die we in de code hebben moeten gebruiken. Het programma start met het initiëren van de weegschaal en het kalibreren, zo wordt de functie ‘scale.tare()’ uitgevoerd om de weegschaal op 0 te zetten, en de scale op een waarde gezet waardoor de output naar gram omgezet wordt. Ook wordt het op median average mode gezet, dit vonden wij de meest accurate manier voor het bepalen van het gewicht.

De code die in de loop functie staat is gebouwd op drie condities, en het feit dat dit telkens geloopd wordt. Tenzij de proximity sensor iets gedetecteerd heeft, of de hall-effect sensor doorheeft dat de klep (waar een magneet in zit) open is, zal de code altijd in het ‘else’ deel terechtkomen. Dit deel kan gezien worden als de code voor als er niets aan de hand is. Als er al eerder gedetecteerd is dat er iets in de brievenbus zit, zal het hier ook checken of de brievenbus inmiddels geleegd is. Dit doet het door te kijken of het vorige gewicht gedeeld door 10 groter is dan het huidige gewicht, ofwel; als de brievenbus 90% van zijn gewicht of meer is verloren, gaat het ervan uit dat het geleegd is. Ook wordt hiervoor gekeken of het vorige gewicht wel genoeg is (meer dan 3 gram) om hier een accurate conclusie uit te trekken. Het gewicht kan erg instabiel zijn, dus als het minder is dan dat zal het waarschijnlijk erg snel aangeven dat er minder dan 10% gewicht over is.

Mocht er bij dit deel van de code een open klep of een object voor de proximity sensor gedetecteerd zijn, dan komt het terecht in het stukje code daarboven, in de ‘else if’. Hier wordt snel het voorafgaande gewicht opgeslagen, 5 seconden gewacht, en het nieuwe gewicht opgeslagen. Dan kijkt het of er genoeg gewicht bij is gekomen (minimaal 3 gram) om te concluderen of er wel degelijk nieuwe post bij is gekomen. In dat geval zet het ‘mail_received’ op true, en geeft het aan dat de brievenbus niet meer leeg is.

Wanneer mail_received waar is, komt het in het bovenste deel terecht. Hier wordt dan de ultrasonische afstand binnen de brievenbus opgehaald, dat samen met het gewicht geprint wordt op de Serial port.

De functie ‘getUltrasonicDistance()’ zorgt natuurlijk voor het ophalen van de afstand, maar omdat dit soms erg afweek van de realiteit, is deze functie zodanig in elkaar gezet dat het meerdere (nu vijf) metingen doet, de uitschieters weg laat, en dan het gemiddelde teruggeeft. Daarbij zit er nog een failsafe in dat als dit op een manier als een grotere afstand dan de maximale afstand (35 centimeter) uit de berekening komt, dat het simpelweg deze afstand gelijkstelt aan de maximale afstand.

Hierna wordt de flits aangezet, print het “take picture” op de Serial port, en na twee seconden zet het de flits weer uit.

## Raspberry PI

Onze Raspberry PI is als het ware de ophaler en verzender van onze data. Het leest de data van de microcontroller, slaat dat op, en verzendt het vervolgens naar onze database. Hier leggen we uit in welke stappen dat precies gebeurt. De volledige code van deze bestanden zijn ook in de bijlage te vinden.

**arduinoReader.py**
Dit script is het belangrijkste onderdeel voor de communicatie tussen de brievenbus en de database. Het maakt een SSH verbinding met onze VPS waar ons laravel project, database, en API op staat (hierover later meer). Ook luistert het naar de Serial port waar de Arduino Mega op aangesloten is, en zorgt het natuurlijk voor de juiste configuratie om aan de database toe te kunnen voegen.

Het script begint met het aanmaken van een SSH tunnel, hier wordt dus de verbinding gemaakt, en de mysql connector geactiveerd. Zo kunnen we in de rest van het script SQL queries toepassen op de database, ook al staat deze niet op hetzelfde netwerk.

Na het definiëren van de SQL query, paden, url's, en dergelijke, komt het script in een oneindige while loop terecht. Hier leest het voortdurend af wat er op de Serial port geprint wordt, wat dan ook gelijk in de terminal geprint wordt. Bijvoorbeeld als dit 'take picture' is, dan wordt er een bash script takepicture.sh (hierover later meer) uitgevoerd om daadwerkelijk een foto te maken. Daarna wordt het pad van de foto opgehaald en opgeslagen. Dit wordt de data die uiteindelijk in de database zal komen. Om te bevestigen dat de foto gereed is, wordt 'image_ready' op true gezet.

Als het de string "emptied" tegenkomt, betekent dat dat de brievenbus geleegd is. In dit geval wordt er een post request naar de laravel omgeving gestuurd die vervolgens alle post in de database als 'opgehaald' markeert. De code hiervan zal ik later uitleggen. Als er iets geprint wordt dat geen van de eerste twee commando's matcht, dan is er sprake van andere data die binnenkomt; het gewicht en de ultrasonische afstand. Ook hier wordt er voor bevestiging een boolean gebruikt, 'data_ready' wordt op true gezet.

Dit bovenstaande zit allemaal in een 'try' statement. Als dit dus niet lukt, hebben we te maken met iets dat op de Serial port is geprint waar we niks mee hoeven te doen. In dat geval print het script simpelweg "^ this is not data!". Zo kunnen we in de terminal alsnog zien wat de Arduino aan het printen is, en er zeker van zijn dat het script dit verder niet gebruikt.

Als 'image_ready' en 'data_ready' beide true zijn, betekent dit dat het script de data mag gaan verzenden. Eerst haalt het de datum en tijd op, dat in het gewenste format wordt omgezet. Dit wordt dan met de rest van de data (formatted_date, formatted_time, weight, ultrasonic_distance, image) in een variabele opgeslagen. Deze data wordt dan bij de SQL query bijgevoegd en uitgevoerd. De rauwe data staat in de database!

Nadat dit is gebeurd, moet het script ook aan de Laravel omgeving doorgeven dat er een nieuwe entry is toegevoegd in de raw_data tabel. Dit wordt wederom met een post request gedaan. Waarom we dit doen zal logischer worden wanneer we het Laravel deel uitleggen.

**takepicture.sh**
Dit is een kort bash scriptje dat door arduinoReader.py wordt aangeroepen voor het maken, opslaan, en verzenden van foto's. Allereerst wordt het pad van waar de foto moet worden opgeslagen, inclusief de naam van de foto (de datum en tijd), alvast gedefinieerd. Met de 'fswebcam' commando kan vervolgens de foto gemaakt worden. De resolutie zetten we op 640x480, en de banner zetten we uit zodat we een mooie foto hebben.

Daarna worden de nodige variabelen om ssh verbinding te maken met de VPS gedefinieerd, waarmee we vervolgens met de sshpass commando het fotobestand kunnen versturen naar de juiste plek in de Laravel omgeving.

## Laravel en de Virtual Private Server

**Laravel**
Voor het zichtbaar maken van de database, het verwerken van data, en het verzorgen van de API hebben we het Laravel framework gebruikt. Dit is iets waar we door voorgaande schoolprojecten al enigszins ervaring mee hadden. Laravel werkt met een SQL connectie die met environment variables ingesteld kan worden. Het is erg goed in het aanmaken van routes, en daar toepassingen op doen aan de hand van controllers.

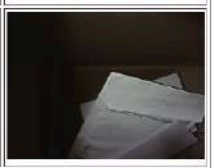
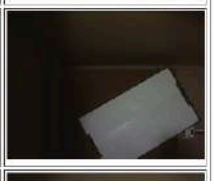
Met 'migrations' hebben we de tabellen 'raw_data' en 'inbox' aangemaakt. Het idee hiervan is dat de rauwe data van de Arduino rechtstreeks in 'raw_data' komt, en dat die data dan automatisch verwerkt wordt en in 'inbox' komt te staan, wat uiteindelijk de data wordt die voornamelijk voor de app wordt gebruikt.

Het Python script dat we hiervoor hebben besproken zorgt vanzelf al voor het toevoegen van entries in de raw_data tabel. De post request die daarna gebeurt is er dus om door te geven dat er een nieuwe entry is om verwerkt te worden. Deze post request heeft een bepaalde URL, waardoor het door middel van routing naar een bepaalde functie in de DataController gestuurd wordt. In dit geval is dat 'triggerEvent()'. Dit slaat de meegeleverde data op in $rawData, wat vervolgens wordt meegegeven als event 'RawDataCreated'. Het bestand 'ProcessRawData.php' is de event listener die hierdoor getriggerd wordt. Hier gebeurt een erg belangrijk deel van de dataverwerking wat we hier verder zullen uitleggen. De inhoud van dit bestand en de DataController zijn in de bijlage te vinden.

Voor het verwerken van de data moeten verschillende dingen gebeuren. Met name het bijhouden van het vorige gewicht zodat er per nieuw poststuk achterhaald kan worden hoe zwaar het was. In de code is een if-else statement te zien. Eerst kijkt het of de bijgeleverde data slechts beschrijft dat de brievenbus geleegd is, zo ja, dan zet het alle post die op dat moment in de inbox tabel te vinden is op 'retrieved', wat betekent dat alle post is opgehaald. Het gewicht wordt dan ook weer op 0 gezet.

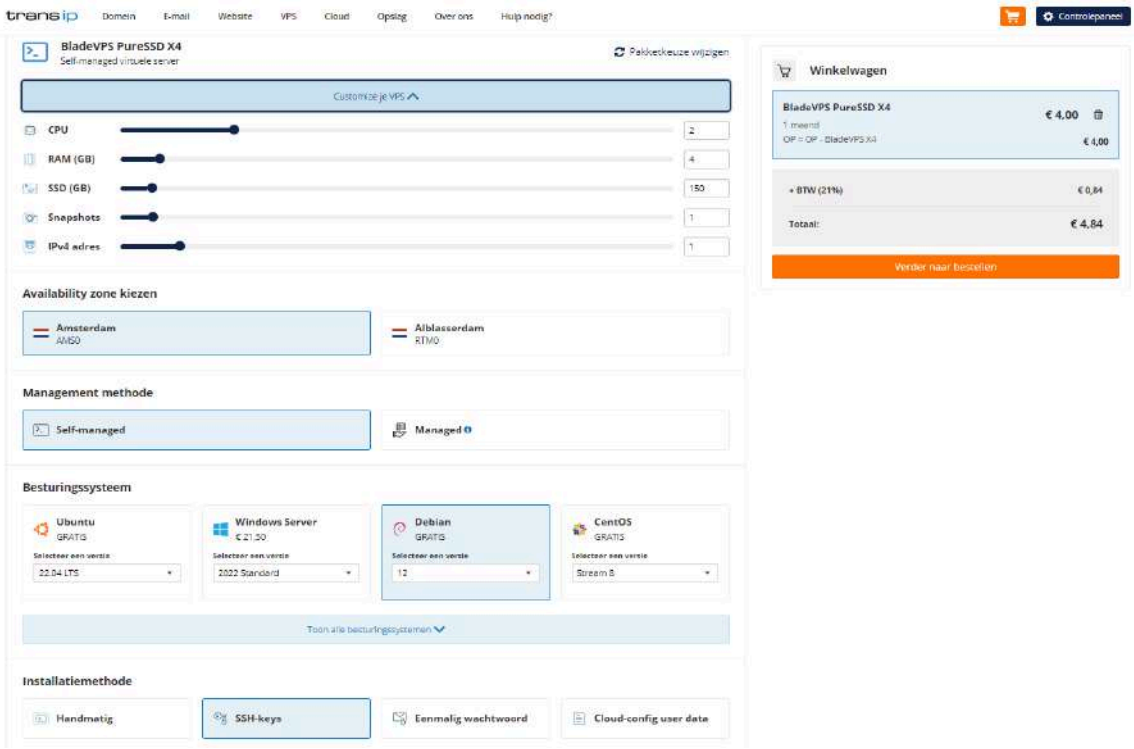
Als het in de else statement komt, betekent dat dat er daadwerkelijk een nieuwe entry in raw_data is gekomen dat verwerkt moet worden. Als eerste wordt het opgehaalde gewicht van de brievenbus (het totale gewicht) afgetrokken van het vorige gewicht dat is opgeslagen, zodat het gewicht van de nieuwe post duidelijk wordt. Dan wordt het percentage van de 'volheid' van de brievenbus berekend door simpelweg de ultrasonische afstand te

vergelijken met de maximale lengte van de post ruimte van de brievenbus. Ook wordt de bezorgdatum en -tijd van strings omgezet naar DateTime objecten.
Daarna is de data gereed om aan de inbox tabel toe te voegen. Hier wordt ook gekeken of het gewicht van het nieuwe poststuk boven de 50 gram is, in welk geval we het als pakketje classificeren, en anders een brief. Uiteraard wordt ‘retrieved’ hier ook op false gezet. Als laatste slaan we het nieuwe totale gewicht op.

| ID | Entry datetime | Delivery Date | Delivery Time | Weight (grams) | Ultrasonic Distance (cm) | Image |
|---|---|---|---|---|---|---|
| 10 | 2024-01-03 16:32:23 | 2024-01-03 | 16:32:23 | 17 | 40 |  |
| 9 | 2024-01-03 16:30:48 | 2024-01-03 | 16:30:48 | 108 | 41 |  |
| 8 | 2024-01-03 16:30:11 | 2024-01-03 | 16:30:11 | 98 | 43 |  |
| 7 | 2024-01-03 16:29:23 | 2024-01-03 | 16:29:23 | 101 | 38 |  |

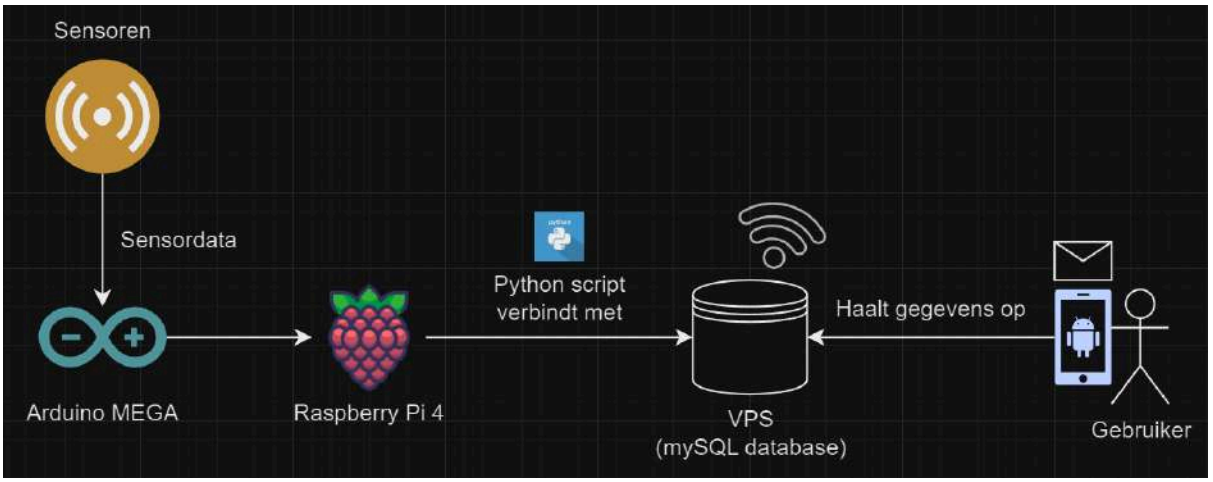
De bovenstaande screenshot is hoe de raw_data tabel eruit zag tijdens de testfase met de data die wordt opgehaald uit de sensoren. Dit is een website (view) die we hebben aangemaakt om de gegevens uit de database op een handige manier te tonen. Dit is waar we vooral bij het testen veel gebruik van hebben gemaakt om snel te kunnen zien of de data goed is opgehaald. Natuurlijk hebben we ook zo’n view voor de inbox tabel gemaakt. Die hebben we dus gebruikt om te zien of de data correct is verwerkt.

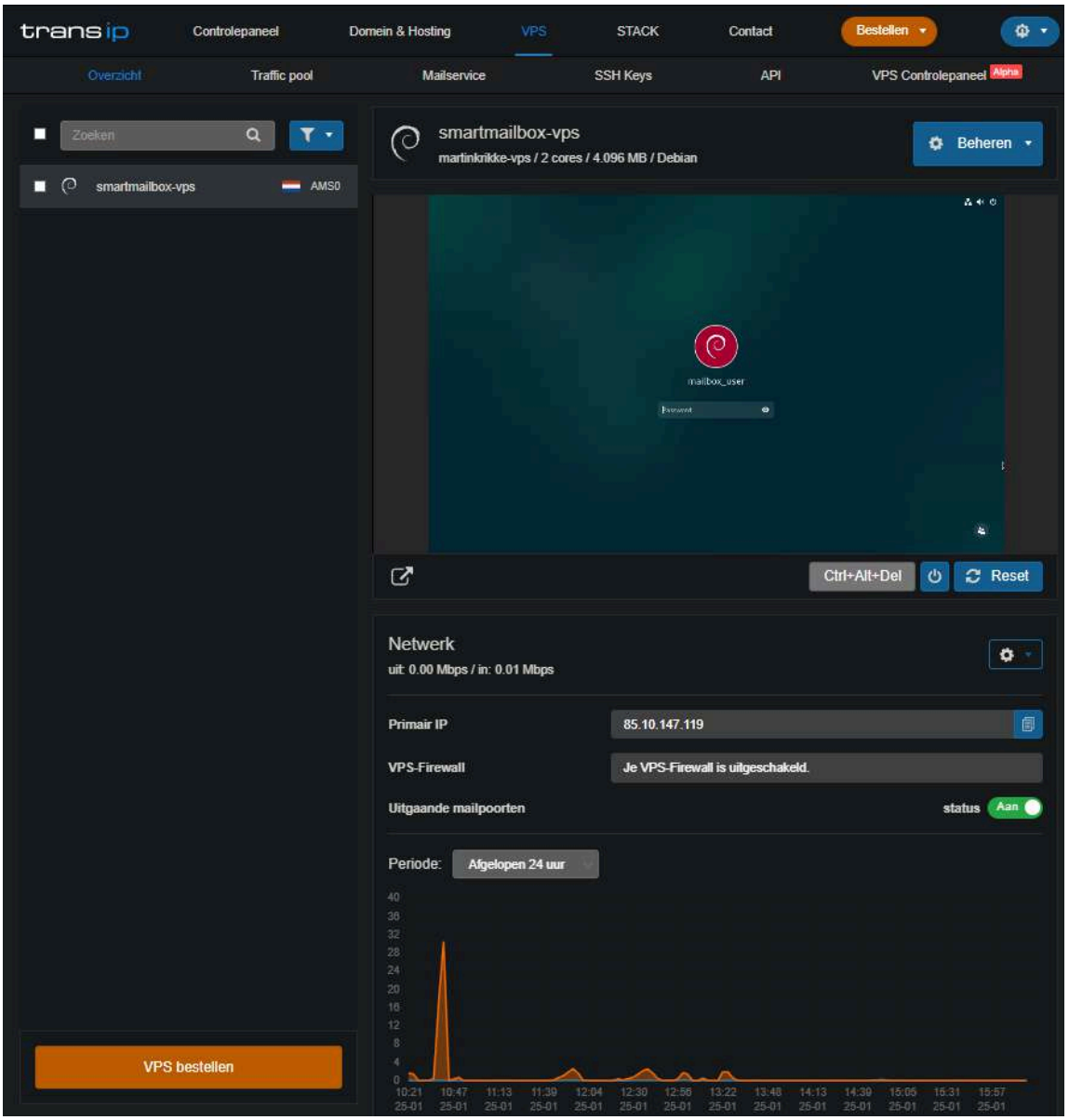
**Aanschaffen van de VPS**
Voor zo goed als de volledige tijdsduur van dit project, hebben we deze Laravel omgeving op de Raspberry Pi gedraaid. Toch werd het tijd om ervoor te zorgen dat deze data niet slechts lokaal gehost wordt, maar ook online op te halen is. Hiervoor gebruiken wij een Virtual Private Server. Met zo’n VPS hebben we toegang tot een virtuele omgeving die lijkt op een eigen server, maar zonder de kosten en verantwoordelijkheden die gepaard gaan met het beheren van zo’n fysieke server. Om deze reden kunnen we onze data snel hosten op zo’n VPS. We hebben gekozen voor een VPS van TransIP, omdat dit een bekend, vertrouwd bedrijf is, en we voor een kleine 4 euro een VPS konden kopen voor 1 maand, genoeg tijd om ons project af te ronden. Een groot voordeel van een VPS is dat we zo ook makkelijk de app kunnen testen met echte gegevens, aangezien we natuurlijk niet altijd op hetzelfde netwerk zitten.




De VPS draait Debian, omdat de Raspberry PI OS gebaseerd is op Debian, en de data tot nu toe altijd daarvandaan is gehost. Het Laravel project met de bijbehorende database wordt nu dus gehost op de VPS in plaats van op de Raspberry PI. Dit zorgt ervoor dat de app ook kan refreshen wanneer deze niet verbonden is met het thuisnetwerk, en bespaart stroom voor de Raspberry PI.
De VPS heeft 2 cores, omdat deze slechts de database zal hosten en we dus niet zoveel rekenkracht nodig hebben. Dit geldt ook voor de 4 GB ram geheugen. Dit is voldoende om Debian goed te laten functioneren en de data te hosten, zonder dat de prijs van de VPS veel hoger wordt. Met de 150 GB SSD voor opslaggeheugen, hebben we alle ruimte om ons project schaalbaar te maken, en komen we niet snel in de problemen met te weinig ruimte voor het opslaan van de afbeeldingen.

Een vernieuwde versie van de schematische tekening ziet er als volgt uit:






Hierboven is een foto te zien van hoe het dashboard van onze VPS eruitziet. Hier kunnen we zien met welk IP adres onze app moet verbinden. Ook kunnen we onder andere bijhouden hoeveel aanvragen er gemaakt worden naar de VPS, de status van de CPU, RAM geheugen, en de opslag.

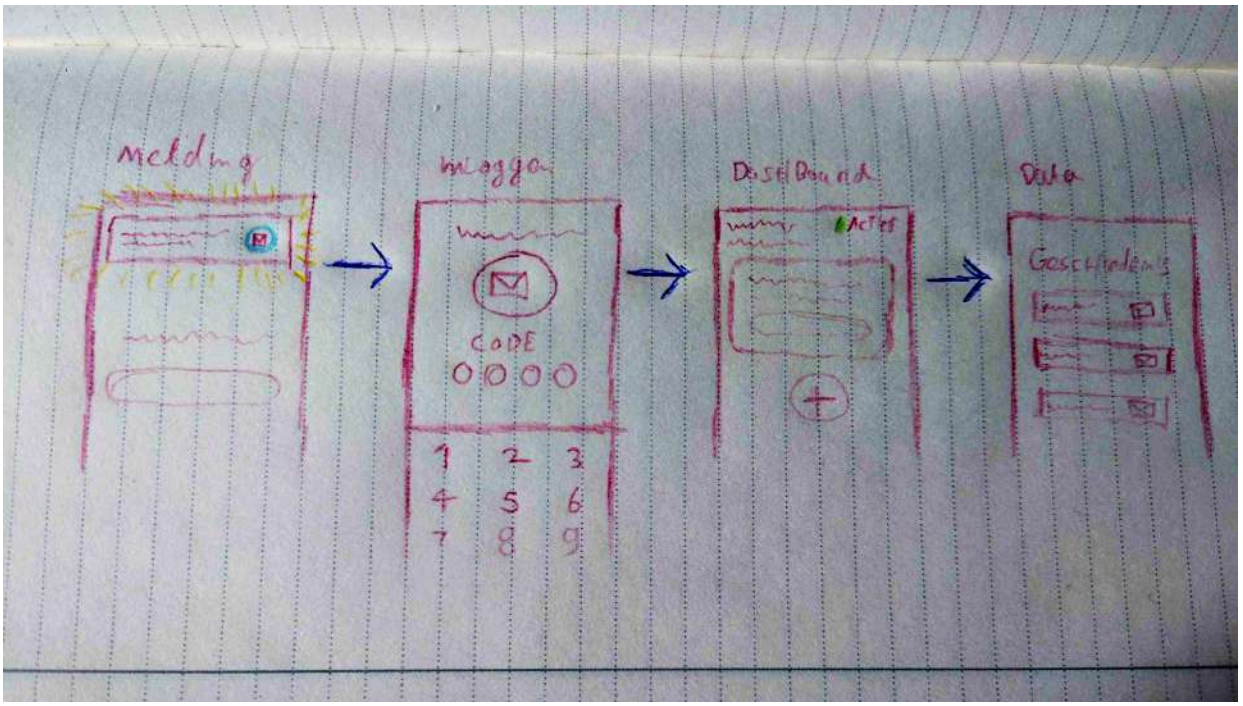
## Hoe tonen we de data op een nuttige en overzichtelijke manier aan de gebruiker?

We zijn nu op het punt gekomen waar we redelijk wat data in een database hebben staan. Het is natuurlijk mogelijk om gewoon deze tabel aan de gebruiker te tonen, maar erg gebruikersvriendelijk is dat natuurlijk niet. Om ervoor te zorgen dat de slimme brievenbus écht toegevoegde waarde heeft voor de gebruiker, moet de data gemakkelijk in te zien zijn, op een manier die weinig moeite kost voor de gebruiker.
Om dit te realiseren, is het belangrijk dat er een (web) app ontwikkeld wordt, die naadloos samenwerkt met de slimme brievenbus.We hebben hiervoor de mogelijkheid te kiezen voor een webapp, of een native app.

- Een webapp is een app die draait op een webbrowser. Het voordeel hiervan, is dat elk apparaat met een (moderne) webbrowser, de app kan uitvoeren, en dat de ervaring op elk apparaat nagenoeg hetzelfde is. Ook hoef je ze niet los te installeren. Het is daarnaast ook mogelijk om via een webapp push notificaties naar de gebruiker te sturen via een gestandaardiseerde javascript API, alleen is hier wel een aparte server voor nodig.
- Een native app wordt specifiek gecompileerd voor specifieke hardware en besturingssystemen. Voordelen hiervan zijn dat je complete toegang tot de hardware hebt, wat voor betere performance kan zorgen. Het sturen van push notificaties gaat dus ook via de OS, en kan dus ook lokaal, zonder server worden geregeld.

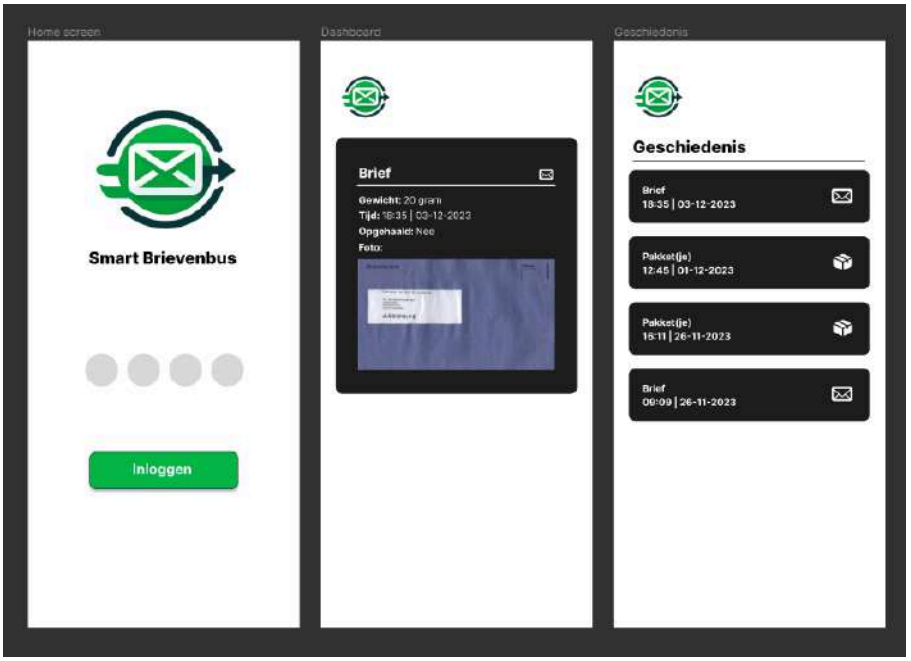
Omdat het versturen van notificaties een belangrijke functionaliteit is in onze app, hebben we ervoor gekozen om een native app te bouwen. Er zijn verschillende soorten frameworks en talen waarin dit kan, en wij hebben gekozen voor het framework Flutter, dat Dart als programmeertaal gebruikt en ondersteund wordt door Google, en Google's Material Design, waarmee we relatief makkelijk een mooie gebruikersinterface kunnen maken. Dit framework hebben we bij een eerdere module gebruikt tijdens de studie, ook met API integratie, dus we hebben hier al enige ervaring mee.
Daarnaast is het aan de hand van de flutter_local_notifications (https://pub.dev/packages/flutter_local_notifications) mogelijk om lokaal notificaties te regelen, wat ons server kosten bespaart.
Het nadeel is wel dat we ervoor hebben gekozen onze app slechts te testen voor het Android besturingssysteem. Hoewel Flutter kan compileren naar iOS, heb je een MacBook met een Apple Developer account nodig om de app te kunnen testen. Dit kost 100 euro per jaar, en dat vinden we het simpelweg niet waard.

Voordat we daadwerkelijk kunnen beginnen aan het programmeren van de app, is het belangrijk om een idee te hebben van hoe we willen dat de app eruit ziet en functioneert. Om deze reden hebben we een simpele schets gemaakt van de user flow.

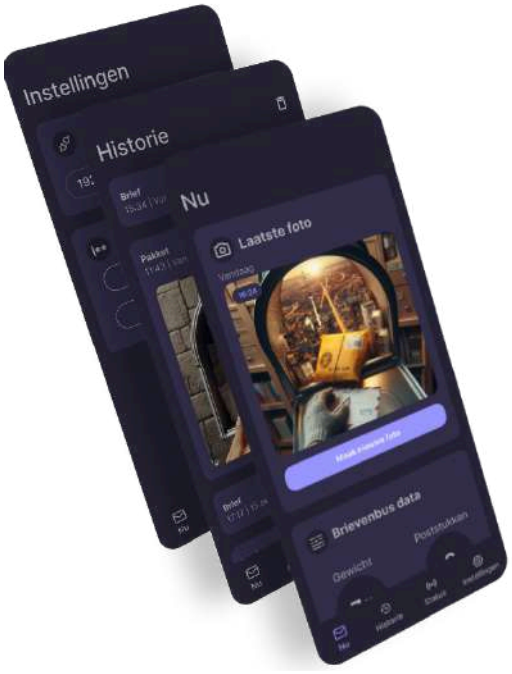



In de userflow is te zien dat de gebruiker een melding ontvangt dat er post is binnengekomen. Dan wordt er om de code gevraagd die de gebruiker heeft ingesteld, om toegang te krijgen tot de data in de app. In het dashboard is dan de laatste foto te zien die de camera gemaakt heeft, met wat informatie over de brievenbus. In de geschiedenis tab is dan te zien wanneer er in het verleden post is binnengekomen, en de foto's die daarbij zijn genomen.

In Figma hebben we op basis van deze userflow, het eerste lo-fi prototype gemaakt. Dit is een versie van het design in een zeer vroeg stadium, waar functionaliteit belangrijker is dan de aankleding van het design. Wegens de simpliciteit van dit prototype, hebben we hier nog geen usability tests op uitgevoerd. Wel geeft het ons een beter beeld op wat voor manier we data in eindontwerp willen presenteren.




Om deze reden hebben we dan ook een klikbaar, Hi-Fi prototype gemaakt.



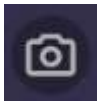

Deze versie van het ontwerp ziet er een stuk meer uit als hoe de uiteindelijke app eruit moet zien. Ons klikbaar prototype is te vinden met de volgende link:

https://bitly.ws/3abyV

Het klikbare prototype laat de flow zien wat de gebruiker te zien krijgt na het inloggen, hoe je de geschiedenis kan bekijken met bijbehorende foto's, en de flow van hoe de cijfercode in de app gewijzigd kan worden.
Om dit design te testen, hebben we met vijf testpersonen usability tests uitgevoerd (zie: bijlage). Tijdens deze usability tests, werden testpersonen gevraagd de volgende taken uit te voeren in de app:

- Bekijk welke foto er is gemaakt op 14 december om 12:31
- Verwijder de historie
- Wijzig de cijfercode van de app
- Kom erachter of de brievenbus verbonden is
- Kom erachter hoeveel gewicht er gedetecteerd wordt in de brievenbus

Aan de hand van de usability tests hebben we de volgende aanpassingen gedaan:

- Een bevestiging bij het verwijderen van de historie
- Betere signifier voor het bekijken van foto's in de historie
  -  Afbeelding is niet zichtbaar
  -  Afbeelding is zichtbaar
- Een CTA knop om handmatig terug te gaan naar de instellingen na het instellen van een nieuwe pincode, in plaats van dat het automatisch gebeurd na een paar seconde
- De tekst "Geen historie gevonden" als er geen historie entries zijn, in plaats van gewoon een leeg scherm.

Dit zijn een paar simpele aanpassingen, omdat de usability tests eigenlijk heel goed gingen. Het kostte gebruikers weinig moeite om de test taken uit te voeren, en hiermee hebben we gevalideerd dat dit ontwerp klaar is om gebruikt te worden voor de daadwerkelijke app.
De ruwe data van de usability tests is te vinden als bijlage voor dit document.

Nadat we tevreden waren over ons Hi-Fi prototype en we de kleine wijzigingen op basis van de usability tests hadden doorgevoerd in ons Figma bestand zijn we langzaamaan begonnen met het maken van de daadwerkelijke app. Voordat we het design van de app konden gaan maken was het belangrijk om de API verbinding tot stand te brengen. Door middel van deze verbinding kunnen wij de door ons opgeslagen data uit de database ophalen en gebruiken in de app. Het verbinding maken met de API in flutter verliep niet vlekkeloos. In eerste instantie konden we de data niet inladen, toen lukte het ophalen van de data alleen in de file waar we verbinding maakten met de API, maar uiteindelijk lukte het ons om ook in andere dart files de data van de API te gebruiken. Nu kunnen we overzicht houden in onze code door onze functionaliteit op het gebied van de API in een eigen file te zetten. Elke pagina van de app maakt verbinding met de api.dart voor het ophalen van de data. Maar wat wordt er dan allemaal geregeld in deze file?

**api.dart**
In de api.dart file regelen we vrijwel alles met betrekking tot de API. We maken verbinding met de API, halen de data op uit de API, slaan deze data op en zorgen ervoor dat de verschillende schermen (pagina's) uit de app updaten wanneer er nieuwe data is.

Om verbinding te maken met onze API gebruiken we één host link (de basis link die nooit verandert) en twee variabelen met een path dat naar de data van één van de tabellen uit de database leidt. Deze twee variabelen zijn: inboxDataurl en rawDataurl. De namen vertellen het al een beetje maar, inboxDataurl verwijst naar de inbox tabel en rawDataurl verwijst naar de raw data tabel.

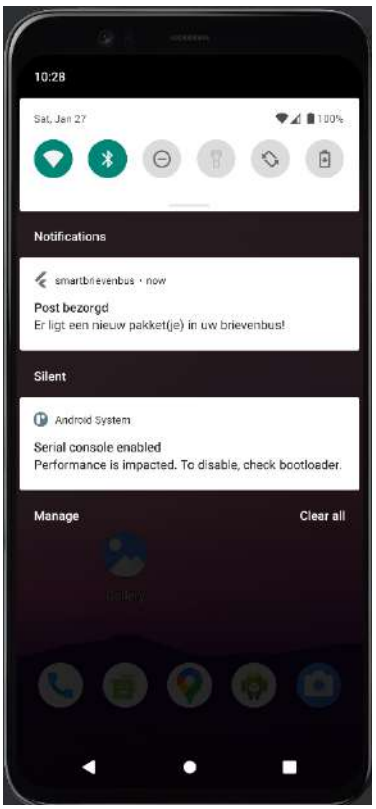
Bij het initialiseren van de file halen we data op vanuit beide tabellen, dit doen we door middel van het aanroepen van de getData(). In de getData functie geef je een url mee, dit is de hostlink + de link van de inbox of rawdata url. De functie kijkt vervolgens of deze link data bevat en zet de uitkomst in een response variabele. Is de response vervolgens 200? dan wordt de response als json.decode(response.body) geretourneerd. Is dit niet het geval? dan kan de data niet opgehaald worden en wordt dit duidelijk gemaakt door middel van een exception. Deze data wordt opgeslagen in de data en de dataRaw variabelen. Hiernaast stellen we ook een timer in die elke minuut de updateData() aanroept.

Nu de data opgehaald is en de timer ingesteld, gaan we kijken naar de werking van de updateData(). Deze functie probeert wanneer deze wordt aangeroepen opnieuw data op te halen van de API en zet de uitkomt in de newData en newDataRaw variabelen. Vervolgens wordt er gekeken of de useFilteredData variabele true is. Dit is de variabele die standaard bij het opstarten van de app op false staat. Deze variabele zorgt er later voor (wanneer deze true is) dat wanneer de geschiedenis tot de huidige dag wordt gewist, de functies deleteHistory() en deleteHistoryRaw() worden aangeroepen, en we niet meer alle data hoeven te gebruiken in onze app. Is deze dus false? dan gaan we naar de else van de functie. Hier wordt compareData() aangeroepen met hierin de data en newData variabele. Als uit deze functie blijkt dat de data verschillend is, updaten we de data en dataRaw variabelen met de nieuwe opgehaalde newData en newDataRaw variabelen. Zo worden dus eigenlijk de data en dataRaw overschreven met de nieuwe gegevens. Vervolgens controleren we door middel van de determineNotificationType() functie of de nieuwe data een package of letter is en geven we deze waarde door aan de notificationType variabele. Hierna geven we de onDataUpdate(), een callback functie, een seintje dat de UI geüpdate moet worden. Deze zorgt ervoor dat dit daadwerkelijk gebeurt. Als de UI geupdate is, roepen

we de notification file aan met de zojuist door ons bepaalde post soort. Deze file zorgt er vervolgens voor dat er een melding naar de gebruiker gestuurd wordt. Mocht de data niet verschillen, dan gebeurt er niks.

Mocht useFilteredData nou true zijn dan komen we terecht in de deleteHistory() en deleteHistoryRaw() functies. Deze functies zullen nu altijd aangeroepen worden totdat de app opnieuw wordt opgestart en de geschiedenis weer vol staat met alle data. Deze functies halen de nieuwste data op uit de API, legen de gefilterde waarde (van de vorige keer door deze functie lopen) en komen dan in een for loop terecht. Hier wordt gekeken of de datum van elk API element gelijk is aan de datum die het ook daadwerkelijk nu is. Hiervoor wordt verwezen naar de checkDeliveryDate(). Deze functie zorgt dat wanneer de datum overeenkomt, de datum wordt vervangen door “Vandaag”. Dan wordt er gekeken welke data uit de API nu “Vandaag” bevat. Alle die dit hebben worden opgeslagen in filteredMail of filteredRaw. Dan kijken we of de filteredMail niet leeg is. Zo ja? dan verlaten we gewoon de functie en doen we dus niks, zo niet? dan kijken we of useFilteredData op true stond. Dit is het geval nadat de code voor de tweede keer wordt doorlopen. Als deze true is dan kijken we of de previousFilteredMail gelijk is aan filteredMail, door middel van de eerder genoemde compareData(). Zijn ze niet hetzelfde? dan bepalen we weer of het een pakket(je) of brief is door middel van determineNotificationType() en geven we deze waarde mee voor de daadwerkelijke notificatie. We zetten de useFilteredData op true, updaten de data met filteredMail of filteredRaw, geven door aan onDataUpdate() dat de UI geupdate moet worden en slaan de useFilterdRawData, filteredMail() en filteredRaw op in SharedPreferences. SharedPreferences zorgt ervoor dat de staat van variabelen wordt opgeslagen. Deze kan zo onthouden worden voor op andere pagina’s of zelfs voor bij het opnieuw opstarten van de app.

*Meldingen*
Voor de meldingen die naar de gebruiker gestuurd worden maken we gebruik van de eerder genoemde flutter_local_notifications. Wanneer een gebruiker zijn app op de achtergrond heeft staan (niet volledig afgesloten) krijgt deze een melding boven in zijn/haar scherm met daarin de tekst: Er ligt een nieuwe brief in uw brievenbus! of Er ligt een nieuw pakket(je) in uw brievenbus! de meldingen zien er zo uit:

Nu de API werkend was zijn we gaan beginnen met de daadwerkelijke pagina's uit de app. Dit zijn het login scherm, het home scherm, het instellingen scherm en het status scherm. Voor het maken van deze pagina's hebben we zo veel mogelijk gekeken naar ons Figma prototype. We hebben geprobeerd deze zo goed mogelijk soortgelijk na te bouwen, omdat we weten dat deze begrepen werd door onze testpersonen. Elk scherm heeft zijn eigen naam die aangeeft waar het scherm voor dient. We zullen ze hieronder scherm voor scherm behandelen.

**login.dart**
Het login scherm van onze applicatie. Dit scherm zorgt voor de beveiliging van onze app. Om bij de gegevens van de brievenbus te komen, dient de gebruiker een cijfercode in te vullen. Standaard is deze 1234. De gebruiker heeft in onze app de mogelijkheid om deze code aan te passen. Dit kan gedaan worden in de instellingenpagina. Bij het verkeerd invullen van de cijfercode krijgt de gebruiker een terugkoppeling door middel van een melding onderin het scherm. Deze melding luidt als volgt: Cijfercode onjuist! probeer opnieuw. Ook wordt door middel van kleur aan de gebruiker getoond welke vakjes van de cijfercode zijn ingevuld/waar de gebruiker nu op staat.

Het login scherm ziet er als volgt uit:



In de code wordt er gekeken of de ingevulde waarde overeenkomt met de opgeslagen waarde in Shared Preferences. Is dit het geval, dan komt de gebruiker automatisch op het home scherm terecht. Als de code fout is, krijg je de melding onderin je scherm.

**home.dart**
Op de home pagina van onze app ziet de gebruiker de laatst bekende informatie vanuit onze brievenbus. De meest recente foto wordt getoond, met daarbij de datum en het tijdstip zodat de eigenaar van de brievenbus precies weet wanneer de post binnen gekomen is. Hieronder staat de brievenbus data. Dit bestaat uit het laatste totaalgewicht dat door de brievenbus is gemeten en het aantal poststukken dat zich in de brievenbus bevindt. Indien de brievenbus leeg is zullen de gewicht en poststukken data dus ook beide het getal 0 bevatten. Op deze manier heeft de eigenaar een inzicht in het aantal poststukken dat er in zijn/haar brievenbus ligt en hoe zwaar deze post weegt. Wanneer post geleverd is op dezelfde datum als de huidige datum, zal er boven de foto de tekst "Vandaag” staan. Voor alle andere dagen staat de complete data string, dus bijvoorbeeld 26-01-2024.

Bij het initialiseren van de pagina halen we de useFilteredData, filteredMail en filteredRaw op, en zeggen we dat als de setOnDataUpdate wordt aangeroepen in de api file we de UI updaten. Het updaten van de pagina gebeurt dus hier door middel van de setState. De setState zorgt er vervolgens voor dat de gehele widget build() en daardoor ook de widget buildDataWidget() opnieuw gebouwd wordt.

In de widget build kijken we naar de connectie met de API. We proberen 30 seconden om verbinding te maken. tijdens het proberen verbinding maken krijgt de gebruiker de tekst: “Zoeken naar brievenbus…” te zien. Is er na 30 seconden geen verbinding of een error? Dan krijgt de gebruiker de tekst: “Geen brievenbus gevonden” te zien. En is er wel een verbinding met de API, dan wordt er gekeken of er sprake is van useFilteredData die op true staat. Als dat zo is dan wordt de data die meegegeven wordt aan de buildDataWidget ingevuld met de filteredMail gegevens.

Binnen de buildDataWidget wordt vervolgens benodigde data opgehaald door middel van ${dataRaw.last['kolomnaam']} en ${data.first['kolomnaam’]}. Voor het aantal pakketjes dat nog niet opgehaald is wordt er een aparte functie gebruikt met de naam countNonRetrieved(). Deze functie loopt alle data langs en kijkt welke hiervan als waarde 0 hebben in de retrieved kolom. Dit aantal wordt opgeteld en als return count uit de functie gestuurd.

De home pagina ziet er als volgt uit:

**history.dart**

Op de geschiedenispagina van onze app kan de gebruiker alle data uit de database zien, of te wel alle ooit binnengekomen post. De post wordt op een kaartje getoond met daarop de soort post: Brief of Pakket(je), de tijd, datum en het gewicht. Ook is er een icoontje van een foto en een pakket(je) of brief te zien. Het icoontje van het pakket(je) of de brief geeft de gebruiker extra verduidelijking of de post een brief of pakket(je) is. Als dit icoontje dezelfde kleur is als de tekst, betekent dit dat het poststuk nog niet is opgehaald. Is deze vervaagd, dan is de post wel opgehaald. Ook de tekst van een niet opgehaald pakketje is dikker gedrukt. Op deze manier proberen wij een soort inbox gevoel na te bootsen zoals een email app ook heeft. Het icoontje van een afbeelding geeft aan dat er een afbeelding beschikbaar is van dat poststuk. De gebruiker kan op dit icoontje klikken en krijgt dan de foto te zien die hoort bij dit poststuk. Zo kan de gebruiker wanneer er post boven op elkaar ligt, alsnog zien wat elk individueel stuk is, en zijn de oude afbeeldingen nog bruikbaar.

Ook heeft de gebruiker in dit scherm de mogelijkheid om de app een soort van op te schonen. Wanneer er op het prullenbak icoontje gedrukt wordt krijgt de gebruiker een popup scherm te zien met de vraag of hij/zij het zeker weet dat alle post tot vandaag verwijderd moet worden uit de app. De gebruiker heeft dan de keuze om op ja of nee te drukken. Bij nee gebeurd er niks, bij ja schoont de app op. Omdat we natuurlijk ook deze functionaliteit moeten laten zien tijdens de presentatie hebben wij voor nu ingesteld dat de app bij het opnieuw opstarten weer alle data bevat. Dit doen we door de variabele van useFilteredData weer op false te zetten wanneer de app geopend wordt. Normaal gesproken doe je dit dus niet waardoor de database alle post kan blijven bevatten en alleen je app wordt opgeschoond. De gebruiker ziet dan alleen de post van vandaag, en kan het elke dag weer opnieuw opschonen. Het opschonen van de app gebeurt door middel van de eerder vernoemde deleteHistory() functie.

De geschiedenis pagina ziet er als volgt uit:



Het geschiedenis scherm heeft dezelfde initialisatie als het home scherm en laat dus ook onder dezelfde voorwaarde “Zoeken naar brievenbus…” en “Geen brievenbus gevonden” zien. Deze schermen zien er als volgt uit:


**status.dart**

Op het status scherm van onze app kan de gebruiker de verbinding met de API zien. Ook kan er wanneer er een verbinding met de API is gezien worden hoe vol de brievenbus is. Dit gebeurt technisch op dezelfde manier als op de andere pagina's. Dit geeft een gebruiker inzicht in wanneer de brievenbus geleegd moet worden en op één plek inzicht in de precieze status van de API. Wanneer wordt getoond hoe vol de brievenbus is, maken we gebruik van kleur. Een lege brievenbus is groen, een brievenbus met post is rood. Een cirkel op basis van kleur, om het brievenbus icoon heen geeft aan hoe vol de brievenbus is. Om het nog verder te verduidelijken geven we ook het percentage aan onder de cirkel met het icoon. Het status scherm kan er als volgt uit zien:




**settings.dart**
In de instellingenpagina van onze app kan de gebruiker de cijfercode wijzigen. Dit kan de gebruiker doen door in het bovenste veld zijn huidige cijfercode in te vullen, in het onderste veld de nieuwe cijfercode in te vullen, en vervolgens op instellen te drukken. Dit kan een aantal scenario's opleveren:

1. De nieuwe cijfercode is hetzelfde als de huidige cijfercode
   In dit geval krijgt de gebruiker de melding: Cijfercode moet anders zijn dan de huidige cijfercode.
2. De ingevulde huidige cijfercode komt niet overeen met de huidige cijfercode
   In dit geval krijgt de gebruiker de melding: Cijfercode onjuist. Probeer opnieuw
3. De huidige cijfercode klopt en er is een cijfercode ingevuld voor de nieuwe cijfercode die niet overeenkomt met de huidige cijfercode.
   In dit geval krijgt de gebruiker de melding: Cijfercode met succes bijgewerkt

Door deze meldingen zorgen we ervoor dat de gebruiker precies weet wat er goed of fout is gegaan, en deze zijn/haar gedrag hierop kan aanpassen.

De nieuwe door de gebruiker ingestelde cijfercode wordt door shared preferences opgeslagen en kan de volgende keer bij het inloggen van de app worden gebruikt.

De scenario's zien er als volgt uit:



# Conclusie

Na zo'n 9 weken aan werk en onderzoek hebben we een veel beter beeld gekregen van het antwoord op de hoofdvraag:

**Hoe kunnen wij een slimme brievenbus ontwerpen die de gebruiker op de hoogte houdt van binnengekomen post?**

Natuurlijk is deze vraag niet zomaar met een korte uitleg te beantwoorden. Het is een samenstelling van alle deelvragen die we uitgebreid in dit verslag hebben behandeld, en dit verslag zelf fungeert dus als één groot antwoord op de hoofdvraag. Om het toch maar even samen te vatten, hebben we hier een kort lijstje van de belangrijkste aspecten:

- Het is belangrijk dat de juiste soorten data worden opgehaald. Voor een brievenbus is het gewicht en de inhoud goed om te weten, zeker als je daarvoor niet in de buurt van de brievenbus hoeft te zijn.
- Aansluitend op het vorige punt, moet deze data accuraat opgehaald zien te worden met de benodigde sensoren. Daarbij helpt het heel erg mee om de sensoren en de ondersteunende hardware goed bereikbaar te maken, zodat er altijd aan gesleuteld kan blijven worden.
- De verwerking van de data moet nuttig zijn, en goed (geautomatiseerd) verlopen. Veel data zal in onverwerkte staat niks zeggen. Dit moet dus omgezet worden naar iets dat elke gebruiker kan begrijpen en interpreteren.
- Post zal af en toe privé-informatie bevatten, en het is dus noodzakelijk om te voorkomen dat iedereen zomaar bij deze informatie kan. Zowel fysieke als digitale veiligheid moet in orde zijn.
- De app die bij de brievenbus hoort, moet gebruiksvriendelijk, nuttig, en overzichtelijk zijn. De gebruiker zou idealiter in één oogopslag moeten zien wat de huidige status van de brievenbus is, en daarop kunnen reageren. Hier hoort bij dat de data waar de app gebruik van maakt altijd beschikbaar moet zijn.

Een concept waar we tijdens deze minor sensortechnologie veel mee te maken hebben gehad, zijn de vier lagen van IoT architectuur. Ook onze Smart Brievenbus kan hier ook goed in verdeeld worden:

- **Devices**
  - De microcontrollers: De Arduino Uno en Arduino Mega, inclusief de aangesloten sensoren. Verantwoordelijk voor het opnemen van data.
- **Communications**
  - De Raspberry Pi. Deze haalt de data van de Device laag op en verstuurt het naar onze online omgeving.
- **Data Processing**
  - Het Laravel project op onze VPS. Deze slaat de data die het van de Communications laag ontvangt op in de database, en verwerkt dit tot bruikbare gegevens.
- **Application**
  - De Flutter app. Dit geeft de nodige informatie op een mooie manier weer.

Over het algemeen zijn we erg blij met het eindproduct. Bijna alles werkt goed. De brievenbus ziet er goed uit en is heel handig in elkaar gezet. Het opvangen, verzenden, en verwerken van de data gebeurt snel en consistent, en het opzetten van de VPS was een goed idee dat in één middag al opgezet was. Iets waarvan we hadden verwacht dat het meer tijd zou innemen. Ook is het ontwerpen en opzetten van de app goed gelukt. Het ziet eruit als een mooie, functionele en professionele app.

Natuurlijk zijn er net als in elk project ook gebreken aan de Smart Brievenbus. Het meeste heeft te maken met de sensoren. Met name het gewicht dat de weegschaal afleest is nog erg consistent, zelfs nadat we dit hebben geprobeerd te optimaliseren. Dit leidt er soms ook toe dat het moeite heeft met detecteren wanneer de brievenbus geleegd is. Ook de proximity sensor detecteert niet altijd een nieuw stuk post.

Het security systeem werkt 90% van de tijd zoals het hoort, maar er zijn momenten dat de backlight van de vingerafdruksensor aan blijft staan, ook als het systeem inactief is, en soms wordt het slot bij een druk op de knop open, maar direct erna dicht gedaan.

Het is voor ons onbekend of dit problemen zijn waar we iets aan hadden kunnen doen. We hebben het wel echt geprobeerd, maar helaas tevergeefs.

Ook vinden we het jammer dat we niet echt tijd hebben gehad om meer met encryptie te doen. Gelukkig hebben we de fysieke veiligheid goed geregeld, en natuurlijk denken we niet dat er hackers zijn die veel behoefte aan het stelen van onze data hebben, maar als dit ooit een echt product zou worden die op de markt uitkomt kan je zo’n veiligheidsaspect natuurlijk niet verwaarlozen.

Kortom: We zijn trots op ons project. Na veel vallen en opstaan is het een erg bevredigend gevoel om naar het resultaat te kijken en te bedenken dat dit twee maanden geleden nog maar een idee was. Wel herkennen we ook dat er dingen zijn die we beter hadden kunnen doen. Dit zal natuurlijk altijd het geval zijn, en vanuit een positief standpunt zijn dit goede lessen om mee te kunnen nemen naar de toekomst.

# Reflectie

De samenwerking binnen het project verliep erg goed. Sinds week 1 hielden we met elkaar contact over de voortgang, vragen en problemen waar we tegenaan liepen. We zorgden ervoor dat we elke week op maandag bij elkaar kwamen om gezamenlijk onze presentatie voor de dinsdagen voor te bereiden, bespraken op dinsdag na de les wie de aankomende week aan welke taak zou gaan werken en hielden we elkaar in de latere weken vrijwel dagelijks op de hoogte van de taken die we uit voerden. Op deze manier had iedereen inzicht in de voortgang en konden problemen vrijwel direct opgelost worden. Ook zorgden we er op deze manier voor dat iedereen per week wat te doen had, en het werk eerlijk verdeeld werd.

Helaas kregen wij na een aantal weken te horen dat Démian stopte met de studie, waardoor de planning die wij in de eerdere weken hadden gemaakt aangepast moest worden. In plaats van met drie man aan de code te werken, moesten we nu zo veel mogelijk van onze ideeën met z’n tweeën gaan waarmaken. Hierdoor hebben wij helaas ook ideeën moeten schrappen zoals bijvoorbeeld de encryptie waarover wij gesproken hebben in ons verslag. Hier was gewoon niet genoeg tijd meer voor. Ondanks het verliezen van ons teamlid zijn wij hard doorgegaan en zijn wij tevreden met het eindresultaat dat we hebben neergezet.

Om ervoor te zorgen dat we elk moment bij elkaars werk konden, hebben wij tijdens ons project gebruikgemaakt van een Google Drive- en GitHub omgeving. De Google Drive omgeving gebruikten wij om gezamenlijk aan de documentatie en presentaties te werken. Door middel van GitHub beschikten wij altijd over de laatste versie van onze Laravel- en Flutter omgeving. Op deze manier kon er makkelijk vanuit huis getest en geholpen worden.

Als afronding van dit verslag willen wij onze dankbaarheid uiten voor Kees, Martins vader, voor de grote hoeveelheid hulp die hij ons heeft gegeven bij de constructie van de brievenbus. Zonder hem hadden we nooit zo’n mooie brievenbus in elkaar kunnen zetten. Ook willen we Démian bedanken voor zijn steun gedurende dit project, zelfs na het stoppen met de opleiding. We respecteren zijn beslissing en wensen hem het beste toe.

# Bronvermelding

Hardware aankopen en bijbehorende informatie

- https://www.otronic.nl/
- https://www.tinytronics.nl/

Sensor en library documentatie

- Weegschaal - https://docs.m5stack.com/en/app/scales_kit
- HX711 Library - https://github.com/bogde/HX711
- GROW R307S fingerprint module handleiding - R307S_fingerprint_module_user_manual-V1.1.2.pdf
- Adafruit Fingerprint Sensor Library - https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library

Framework documentatie

- Laravel 8 - https://laravel.com/docs/8.x
- Flutter - https://docs.flutter.dev
- Arduino - https://www.arduino.cc
- Raspberry Pi - https://www.raspberrypi.com/documentation/

VPS

- Trans IP - https://www.transip.nl

# Bijlage: Usability Tests

In deze bijlage is de ruwe data te vinden van de 5 usability tests die we hebben uitgevoerd van ons Figma prototype. Hieronder volgen de testtaken met de wijze hoe deze uitgevoerd moet worden

Taak 1: **Bekijk welke foto er is gemaakt op 14 december om 12:31**
Navigeer via de navigation bar naar Historie > Scroll naar beneden tot de entry van 14 december om 12:31 > Klik op het camera icoontje
De afbeelding verschijnt en de taak is voltooid.

Taak 2: **Verwijder de historie**
Gebruiker bevindt zich al op de juiste pagina > prullenbak rechtsboven
De historie entries zijn verdwenen en de taak is voltooid

Taak 3: **Wijzig de cijfercode van de app**
Navigeer via de navigation bar naar Instellingen > App cijfercode wijzigen knop > tik 3 keer op het toetsenbord om een code in te stellen > wacht 5 seconden > terug op instellingen scherm
Aangezien het hier gaat om een prototype van de app, is er niet echt een pincode ingesteld, maar hiermee is de taak wel voltooid.

Taak 4: **Kom erachter of de brievenbus verbonden is**
Navigeer via de navigation bar naar Status > bovenin het scherm is te zien of de brievenbus online of offline is
Als de gebruiker kan vertellen of de brievenbus online of offline is, is de taak voltooid.

Taak 5: **Kom erachter hoeveel gewicht er gedetecteerd wordt in de brievenbus**
Navigeer via de navigation bar naar Nu > scroll naar beneden > hier staat hoeveel gewicht er gedetecteerd wordt
Als de gebruiker kan vertellen dat de brievenbus 7 gram detecteert, is de taak voltooid.

In de tabel hieronder is de rauwe data te zien van de 5 gebruiker en testtaken

| | Testtaak 1 (3 kliks) | Testtaak 2 (1 klik) | Testtaak 3 (5 kliks) | Testtaak 4 (1 klik) | Testtaak 5 (2 kliks) |
|---|---|---|---|---|---|
| Tester 1 | 3 kliks, 10 sec | 1 klik, 5 sec | 6 kliks, 15 sec | 1 klik, 4 sec | 2 kliks, 5 sec |
| Tester 2 | 3 kliks, 11 sec | 2 kliks, 8 sec | 5 kliks, 13 sec | 1 klik, 6 sec | 2 kliks, 7 sec |
| Tester 3 | 4 kliks, 15 sec | 1 klik, 4 sec | 7 kliks, 17 sec | 3 kliks, 11 sec | 4 kliks, 8 sec |
| Tester 4 | 3 kliks, 8 sec | 1 klik, 4 sec | 6 kliks, 16 sec | 2 kliks, 7 sec | 2 kliks, 4 sec |
| Tester 5 | 5 kliks, 17 sec | 1 klik, 9 sec | 5 kliks, 20 sec | 1 kliks, 10 sec | 3 kliks, 10 sec |

*Het getal tussen de haakjes bij de testtaken, laat zien in hoeveel kliks de taak volgens mijn planning mogelijk is*

# Bijlage: Interviews

Hieronder zijn de doelgroep interviews te vinden die we hebben gehouden om meer inzicht te krijgen in onze doelgroep en om ideeën op te doen voor ons project.

**Interview met Julian**
*Wat voor soort losse brievenbus heb je? Beschrijf de brievenbus.*
Brievenbus in een flat binnen, gleuf, relatief klein

*Wat zijn voor- en/of nadelen die je hebt ervaren met een losse brievenbus ten opzichte van een deur brievenbus?*
nadeel: je moet je huis uit, ook niet automatisch door hebben of je post hebt
voordeel: bezorgers hoeven niet gelijk bij je woning komen en is miss handig voor ze, maar is in de praktijk nooit echt een probleem mee geweest

*Lijkt het je handig om meer inzicht te hebben over de status van je brievenbus? Dus bijvoorbeeld of er brieven of pakketjes in zitten?*

Ja opzich wel, dan weet je wanneer je langs moet. PostNL heeft al zoiets dat notificatie geeft als je iets krijgt

*Zo ja, wat voor informatie zou je graag willen zien?*
hoeft niet heel veel meer voor mij, binnengekomen post en of het brief/pakketje is is wel voldoende

*Hoe vaak leeg je gemiddeld per week je brievenbus? Hebben bijvoorbeeld weersomstandigheden invloed hierop?*
*Heb je dan vaak dat er niks in ligt?*

ongeveer 1 keer per week, meestal geen post, meestal kijk ik even als ik weer thuis kom ipv dat ik vanuit mijn kamer naar beneden loop om te checken

*Is het belangrijk voor jou om op tijd je brievenbus te legen? Krijg je bijvoorbeeld tijdsafhankelijke post? (rekeningen bijvoorbeeld).*
*Heb je wel eens problemen gehad door het te laat legen van je brievenbus? (bijvoorbeeld belastingbrieven ofzo)*

nooit echt problemen mee gehad omdat ik alles digitaal heb, maar weleens brief van gemeente gekregen waar het was van ik heb 3 weken om te reageren maar er waren al 2 weken voorbij

*Heb je je ooit zorgen gemaakt over de veiligheid van je post in de brievenbus als je niet thuis bent?*

Nee nooit echt, zover ik weet kan niemand erbij, ik heb ook een slot. het zou ook wel opvallen als iemand probeert te breken want de brievenbussen staan erg openbaar. dan kan je beter de bezorger overvallen.

**Interview met Florian**

*Wat voor soort losse brievenbus heb je? Beschrijf de brievenbus.*

In de ingang van mijn flat aan de begane grond zit een muur met verschillende brievenbussen. Aan de andere kant van die muur kan dan de post opgehaald worden. Kost ongeveer een halve minuut om daar te komen via 4 trappen en 2 verdiepingen omlaag. We hebben een lift, maar die gebruik ik niet. Traplopen is gezonder. Door de kleine klep passen er geen brievenbuspakketjes in, misschien hele kleine alleen. De brievenbus is ong. 30 cm breed, 20 cm hoog, 30 cm diep.

*Wat zijn voor- en/of nadelen die je hebt ervaren met een losse brievenbus ten opzichte van een deur brievenbus?*

Voordelen: Nooit een gleuf gehad, maar geen irritante kinderen die dingen door de brievenbus gooien, zoals modder. Anders ligt dat ineens in de hal en moet je dat schoonmaken.

Nadelen: Altijd het huis uit moeten om de post te checken.

*Lijkt het je handig om meer inzicht te hebben over de status van je brievenbus? Dus bijvoorbeeld of er brieven of pakketjes in zitten?*

Denk het wel, want vanuit mijn woonkamer weet ik niet of iets is aangekomen. Met een gleuf aan de deur hoor je het, maar met mijn brievenbus niet.

*Zo ja, wat voor informatie zou je graag willen zien?*

Of er reclame binnen is gekomen, of daadwerkelijk belangrijke post. Een brief van de overheid is bijvoorbeeld wel fijn om tijdig te weten dat deze is aangekomen.

*Hoe vaak leeg je gemiddeld per week je brievenbus? Hebben bijvoorbeeld weersomstandigheden invloed hierop?*

Weersomstandigheden hebben geen invloed, want het is binnen, maar ongeveer 1 keer per week.

> *Heb je dan vaak dat er niks in ligt?*
> Door reclamebladen eigenlijk bijna nooit.

*Is het belangrijk voor jou om op tijd je brievenbus te legen? Krijg je bijvoorbeeld tijdsafhankelijke post? (rekeningen bijvoorbeeld).*

Nee daar heb ik geen last van, maar als ik iets bestel is het wel leuk om snel te weten of ik het heb ontvangen. Maar ik heb geen financiele post waarvan het belangrijk is dat ik deze snel heb.

> *Heb je wel eens problemen gehad door het te laat legen van je brievenbus? (bijvoorbeeld belastingbrieven ofzo)*
> Wegens bovenstaande redenen heb ik daar nooit problemen mee gehad.

*Heb je je ooit zorgen gemaakt over de veiligheid van je post in de brievenbus als je niet thuis bent?*

Nooit zorgen over gemaakt, het is mogelijk om met dunne handen mijn post te pakken. Maar waarom zou je dat doen als er constant mensen langs lopen. Beetje raar als je in die hal staat en je staat met je hand te graaien in een van die brievenbussen. En daarnaast is de post niet echt belangrijk genoeg om te jatten, want pakketten worden aan de deur geleverd. Die passen niet door de brievenbus.

**Interview met Luca**

*Wat voor soort losse brievenbus heb je? Beschrijf de brievenbus.*

Ik heb een brievenbus in een appartementencomplex. Dit is een grijze brievenbus met een slot die je kunt openen door middel van een sleutel. Om bij mijn brievenbus te komen moet ik ongeveer 4 meter lopen, het is maar 1 trap af. Ik woon er eigenlijk gewoon naast.

*Wat zijn voor- en/of nadelen die je hebt ervaren met een losse brievenbus ten opzichte van een deur brievenbus?*

Voordelen: De deurmat ligt niet vol, dus je struikelt ook niet over de post.
Nadelen: Het is verder lopen om je post te halen dan met een gleuf in je deur.

*Lijkt het je handig om meer inzicht te hebben over de status van je brievenbus? Dus bijvoorbeeld of er brieven of pakketjes in zitten?*

Nee, het is voor mij niet echt nodig. Ik gebruik de postnl app om te zien wat voor post ik krijg. Daarnaast krijg ik ook niet zoveel post, dus een status van mijn brievenbus heb ik niet echt nodig.

*Zo ja, wat voor informatie zou je graag willen zien?*

-

*Hoe vaak leeg je gemiddeld per week je brievenbus? Hebben bijvoorbeeld weersomstandigheden invloed hierop?*

*Heb je dan vaak dat er niks in ligt?*

Ik leeg bijna elke dag (5-6x per week) mijn brievenbus. Weersomstandigheden hebben hier geen invloed op omdat mijn brievenbus toch in het appartementencomplex zit. Ik heb heel soms dat er bij het legen van de brievenbus niets in zit.

*Is het belangrijk voor jou om op tijd je brievenbus te legen? Krijg je bijvoorbeeld tijdsafhankelijke post? (rekeningen bijvoorbeeld).*

*Heb je wel eens problemen gehad door het te laat legen van je brievenbus? (bijvoorbeeld belastingbrieven ofzo)*

Als het belangrijke post is wel. Mijn rekeningen etc krijg ik via de mail en daardoor heb ik nooit problemen gehad met het te laat zien van post.

*Heb je je ooit zorgen gemaakt over de veiligheid van je post in de brievenbus als je niet thuis bent?*

Nee, want er zit een slot op dus niemand anders kan erbij.

**Interview met Daniel**

*Wat voor soort losse brievenbus heb je? Beschrijf de brievenbus.*

Mijn brievenbus hangt naast de deur van mijn huis. Via een gleuf kan er post inkomen, en met een sleutel kan je een deurtje openen om de brievenbus te openen. Hij is ongeveer 30cm breed en 40cm hoog. Afhankelijk van waar je in huis bent, heeft invloed op hoe ver je moet lopen naar de brievenbus, maar dit zal niet snel langer dan 1 minuut zijn.

*Wat zijn voor- en/of nadelen die je hebt ervaren met een losse brievenbus ten opzichte van een deur brievenbus?*

Voordelen: als de post vroeg is, wordt je niet wakker gemaakt door het kletterende geluid van een deurbrievenbus. Daarnaast kunnen mensen niet zien wat voor brieven je hebt. Bij een deurbrievenbus zouden mensen de brieven/pakketjes kunnen zien liggen door het raam van de deur.

Nadelen: je moet naar buiten om de post op te halen, en je hebt een sleutel nodig die je niet kwijt moet raken.

*Lijkt het je handig om meer inzicht te hebben over de status van je brievenbus? Dus bijvoorbeeld of er brieven of pakketjes in zitten?*

Ja opzich wel. Ik bestel best vaak van aliexpress, en die levertijden zijn altijd best lang en onvoorspelbaar. Ik heb best vaak gehad dat ik uit het niets een aliexpress pakketje in mijn brievenbus zag liggen. Een melding dat deze is binnengekomen zou dan wel leuk zijn.

*Zo ja, wat voor informatie zou je graag willen zien?*

Ik zou ook graag willen zien wat er dan is binnengekomen, zodat ik weet of het een pakket is of gewoon reclame ofzo.

*Hoe vaak leeg je gemiddeld per week je brievenbus? Hebben bijvoorbeeld weersomstandigheden invloed hierop?*

Ik check elke dag de brievenbus als ik terug kom van werk. Soms hebben weersomstandigheden hier wel invloed op, als het bijvoorbeeld heel hard regent wil ik gewoon zo snel mogelijk naar binnen. Gister was het bijvoorbeeld zo hard aan het hagelen, dat het gewoon pijn deed, dus toen had ik het maar even overgeslagen.

> *Heb je dan vaak dat er niks in ligt?*
>
> Redelijk vaak wel, maar dat komt denk ik ook wel omdat ik de brievenbus best vaak controleer.

*Is het belangrijk voor jou om op tijd je brievenbus te legen? Krijg je bijvoorbeeld tijdsafhankelijke post? (rekeningen bijvoorbeeld).*

Nee eigenlijk niet. Rekeningen enzo krijgen we gewoon digitaal, en boetes heb ik eigenlijk nog nooit gehad.

> *Heb je wel eens problemen gehad door het te laat legen van je brievenbus? (bijvoorbeeld belastingbrieven ofzo)*
>
> Nee eigenlijk niet.

*Heb je je ooit zorgen gemaakt over de veiligheid van je post in de brievenbus als je niet thuis bent?*

Niet echt eigenlijk, onze brievenbus gaat best diep, dus het is praktisch onmogelijk om er iets uit te halen vanuit de gleuf. Daarnaast hangen er best veel deurbel camera's in onze wijk, dus ik zie niet snel voor me dat iemand hier post gaat proberen te stelen.

# Bijlage: Code

## Arduino Uno (Security systeem)

```
#include <Servo.h>
#include <Adafruit_Fingerprint.h>

#define SERVOS_PIN A0
#define BTN_PIN 8
#define RGB_R 11
#define RGB_G 6
#define RGB_B 5

Servo servos;

// Fingerprint Sensor
SoftwareSerial mySerial(2, 3);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);

uint8_t id;

int buttonState = HIGH;

unsigned long buttonPressStartTime = 0;
unsigned long accessStartTime = 0;
unsigned long fingerprint_check_startTime = 0;

const int requiredPressTime = 5000;  // 5 seconds in milliseconds
const int fingerprintCheckTimeoutTime = 30000; // 30 seconds
const long accessTimeoutTime = 60000 * 5; // 5 minutes

const int OPEN = 90; // degrees the servo should turn to in order to open the lock
const int CLOSE = 0; // degrees the servo should turn to in order to close the lock

bool long_button_press = false;
bool short_button_press = false;
bool newButtonPress = false;
```

```
bool access_granted = false;
bool prev_access_granted = false;

bool opening = true;
bool check_fingerprint = false;
bool was_checking = false;

bool opened = false;
bool btn_released = true;
bool just_pressed = false;

enum Feedback { SUCCESS, TRY_AGAIN, FAILED, WAITING };

void setup() {
  servos.attach(SERVOS_PIN);

  pinMode(BTN_PIN, INPUT_PULLUP);

  pinMode(RGB_R, OUTPUT);
  pinMode(RGB_G, OUTPUT);
  pinMode(RGB_B, OUTPUT);

  servos.write(CLOSE);

  Serial.begin(9600);

  finger.begin(57600);
  //finger.emptyDatabase();
  finger.getTemplateCount();
}

void loop() {
  int newButtonState = digitalRead(BTN_PIN);
  // Check if the button state has changed
  if (newButtonState == 0 && buttonState == 1 && !just_pressed) {
    // Button has been pressed, record the start time
    buttonPressStartTime = millis();
    newButtonPress = true;
  }
  just_pressed = false;

  // Check if the button has been pressed for 5 seconds or longer
  if (buttonState == 0 && !check_fingerprint && newButtonPress) {
```

```
  if ((millis() - buttonPressStartTime >= requiredPressTime) && !long_button_press) {
    Serial.println("long button press");
    newButtonPress = false;
    long_button_press = true;
  }
  // if button hasn't been pressed for 5 seconds or longer
  else if (newButtonState == 1) {
    delay(10);
    if (btn_released) {
      newButtonPress = false;
      btn_released = false;

      Serial.println("short button press");
      short_button_press = true;
      if (!access_granted) { // checking for fingerprint to open mailbox
        Serial.println("checking fingerprint");
        check_fingerprint = true;
      }
    }

  } else {
    btn_released = true;
  }
}

if (check_fingerprint) {
  short_button_press = false;
  long_button_press = false;
  delay(50);
  if (finger.templateCount == 0) {
    Serial.println("enrolling by empty templateCount");
    enroll();
  } else {
    setColor(0,0,255); // blue led
    fingerprint_check_startTime = millis();
    while ( getFingerprintIDez() == -1 );   // get fingerprint id easy version
  }

} else if (was_checking) {
  setColor(0,0,0);
  finger.LEDcontrol(FINGERPRINT_LEDOFF);
}
```

```
  if (access_granted) {
    if (!prev_access_granted) {
      check_fingerprint = false;
      Serial.println("Access granted!");
      accessStartTime = millis();
    }

    if (long_button_press) {
      Serial.println("enrolling by long button press");
      long_button_press = false;
      enroll();
    }
    else if (short_button_press) {
      if (!opened) {
        Serial.println("opening servo");
        servos.write(OPEN);
        opened = true;
      } else {
        Serial.println("closing servo");
        servos.write(CLOSE);
        opened = false;
        access_granted = false;
      }
      delay(100);
      short_button_press = false;
      just_pressed = true;

    }
    if (millis() - accessStartTime >= accessTimeoutTime) {
      // revoke access after 5 minutes, make sure the mailbox is closed before this!
      showFeedback(FAILED);
      Serial.println("access revoked: timed out");
      access_granted = false;
    }

  }
  prev_access_granted = access_granted;
  was_checking = check_fingerprint;
  buttonState = newButtonState;
}

void enroll() {
  // get amount of enrolls, set id to amount of enrolled fingerprints + 1
```

```
  id = finger.templateCount + 1;
  while (!  getFingerprintEnroll() );
  setColor(0,0,0);
  finger.getTemplateCount();

}

uint8_t getFingerprintEnroll() {
  // get first image of fingerprint - led is blue. if succesful, led goes green, if try again, led goes orange, if failed, led goes red
  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #"); Serial.println(id);
  showFeedback(WAITING);

  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.println(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      showFeedback(TRY_AGAIN);
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      showFeedback(TRY_AGAIN);
      break;
    default:
      Serial.println("Unknown error");
      showFeedback(TRY_AGAIN);
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
```

```
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features (featurefail)");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features (invalidimage");
    showFeedback(TRY_AGAIN);
    return p;
  default:
    Serial.println("Unknown error");
    showFeedback(TRY_AGAIN);
    return p;
}

showFeedback(SUCCESS);
Serial.println("Remove finger");
setColor(0,0,0);
delay(2000);

p = 0;
while (p != FINGERPRINT_NOFINGER) {
  p = finger.getImage();
}
Serial.print("ID "); Serial.println(id);
p = -1;
Serial.println("Place same finger again");
showFeedback(WAITING);

// get second image of fingerprint - led is blue. if succesful, led goes green, if try again, led goes orange, if failed, led goes red
while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
```

```
      break;
    case FINGERPRINT_NOFINGER:
      Serial.print(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      showFeedback(TRY_AGAIN);
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      showFeedback(TRY_AGAIN);
      break;
    default:
      Serial.println("Unknown error");
      showFeedback(TRY_AGAIN);
      break;
  }
}

// OK success!

p = finger.image2Tz(2);
switch (p) {
  case FINGERPRINT_OK:
    Serial.println("Image converted");
    break;
  case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features (featurefail)");
    showFeedback(TRY_AGAIN);
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features (invalidimage)");
    showFeedback(TRY_AGAIN);
    return p;
  default:
```

```
    Serial.println("Unknown error");
    showFeedback(TRY_AGAIN);
    return p;
}

// OK converted!
Serial.print("Creating model for #");  Serial.println(id);

p = finger.createModel();
if (p == FINGERPRINT_OK) {
  Serial.println("Prints matched!");
} else if (p == FINGERPRINT_PACKETRECIEVEERR) {
  Serial.println("Communication error");
  showFeedback(TRY_AGAIN);
  return p;
} else if (p == FINGERPRINT_ENROLLMISMATCH) {
  Serial.println("Fingerprints did not match");
  showFeedback(FAILED);
  return p;
} else {
  Serial.println("Unknown error");
  showFeedback(TRY_AGAIN);
  return p;
}

Serial.print("ID "); Serial.println(id);
p = finger.storeModel(id);
if (p == FINGERPRINT_OK) {
  Serial.println("Stored!");
} else if (p == FINGERPRINT_PACKETRECIEVEERR) {
  Serial.println("Communication error");
  showFeedback(TRY_AGAIN);
  return p;
} else if (p == FINGERPRINT_BADLOCATION) {
  Serial.println("Could not store in that location");
  showFeedback(TRY_AGAIN);
  return p;
} else if (p == FINGERPRINT_FLASHERR) {
  Serial.println("Error writing to flash");
  showFeedback(TRY_AGAIN);
  return p;
} else {
  Serial.println("Unknown error");
```

```
    showFeedback(TRY_AGAIN);
    return p;
  }

  showFeedback(SUCCESS);
  access_granted = true;
  return true;
}

void showFeedback(Feedback message) {
  switch (message) {
    case SUCCESS:
      setColor(0,255,0); // green
      delay(500);
      setColor(0,0,0);
      delay(100);
      setColor(0,255,0);
      delay(500);
      setColor(0,0,255); // return to blue
      break;
    case TRY_AGAIN:
      setColor(165,100,0);  // orange
      delay(500);
      setColor(0,0,0);
      delay(100);
      setColor(165,100,0);
      delay(500);
      setColor(0,0,255); // return to blue
      break;
    case FAILED:
      setColor(255,0,0); // red
      delay(500);
      setColor(0,0,0);
      delay(100);
      setColor(255,0,0);
      delay(500);
      setColor(0,0,0); // return to inactive
      break;
    case WAITING:
      setColor(0,0,255);
      delay(500);
      setColor(0,0,0);
      delay(100);
```

```
      setColor(0,0,255);
      break;
  }
}

// returns -1 if failed, otherwise returns ID #
int getFingerprintIDez() {
  if (millis() - fingerprint_check_startTime >= fingerprintCheckTimeoutTime) {
    Serial.println("fingerprint scan failed: timed out");
    check_fingerprint = false;
    showFeedback(FAILED);
    return -2;
  }

  uint8_t p = finger.getImage();
  if (p != FINGERPRINT_OK) {
    showFeedback(WAITING);
    return -1;
  }

  p = finger.image2Tz();
  if (p != FINGERPRINT_OK) {
    showFeedback(WAITING);
    return -1;
  }

  p = finger.fingerFastSearch();
  if (p != FINGERPRINT_OK) {
    showFeedback(WAITING);
    return -1;
  }

  // found a match!
  Serial.print("Found ID #"); Serial.print(finger.fingerID);
  Serial.print(" with confidence of "); Serial.println(finger.confidence);
  showFeedback(SUCCESS);
  setColor(0,0,0);
  access_granted = true;
  return finger.fingerID;
}

void setColor(int R, int G, int B) {
  analogWrite(RGB_R, R);
```

```
  analogWrite(RGB_G, G);
  analogWrite(RGB_B, B);
}
```

# Arduino Mega (ophalen en verzenden van de sensordata van de brievenbus)

```
#include "HX711.h"

// Scale
HX711 scale;
#define SCALE_DATA_PIN 9
#define SCALE_CLK_PIN 8

float weight = 0;
float prevWeight = 0;
float minWeight = 3.0;

// Flash
#define FLASH_PIN 4

// Hall Effect Sensor
#define HEsensor 5

// Proximity Sensor
#define PRX_PIN 10

// Ultrasonic Sensor
#define trigPin 22
#define echoPin 24
const int numReadings = 5;
unsigned long readings[numReadings];
long duration;
const int max_distance = 35;

// General
bool mail_received = false;
bool mail_inserted = false;
int mailbox_open = false;
bool isEmpty = true;

void setup() {
  pinMode(FLASH_PIN, OUTPUT);
  pinMode(HEsensor, INPUT);
  pinMode(PRX_PIN, INPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
```

```
  scale.begin(SCALE_DATA_PIN, SCALE_CLK_PIN);
  scale.tare(10);

  // The scale value is the adc value corresponding to 1g
  scale.set_scale(27.61f);  // set scale
  scale.set_medavg_mode();

  Serial.begin(9600);

}

void loop() {

  if (mail_received) {
    // Detected new mail
    Serial.println("mail received");
    long ultrasonicDistance = getUltrasonicDistance();
    Serial.print(weight);
    Serial.print(", ");
    Serial.println(ultrasonicDistance);

    // activating flash and taking picture
    digitalWrite(FLASH_PIN, HIGH);
    Serial.println("take picture");
    delay(2000);
    digitalWrite(FLASH_PIN, LOW);

    mail_received = false;
  }
  else if (mailbox_open or mail_inserted) {
    prevWeight = weight; // get weight before new mail
    mailbox_open = false;
    mail_inserted = false;
    delay(5000);
    weight = scale.get_units(30); // get weight after new mail
    if (prevWeight + minWeight < weight) { // check if there was a significant change in weight in the mailbox
      mail_received = true;
      isEmpty = false;
      prevWeight = weight;
    }
  }
  else {
```

```
    mail_inserted = digitalRead(PRX_PIN);
    mailbox_open = digitalRead(HEsensor);

    if (isEmpty == false) {
      weight = scale.get_units(30);

      // if either: current weight is less than 10% of the previous weight (if that was more than 3 grams)
      // -> detect the mailbox as emptied
      if ((weight < prevWeight / 10) and (prevWeight > minWeight)) {
        Serial.println("emptied");
        isEmpty = true;

      }
    }
    if (weight < -1.0) {
      scale.tare(10);
    }

  }

}

long getUltrasonicDistance() {
  // Perform multiple readings
  for (int i = 0; i < numReadings; ++i) {
    // Trigger the ultrasonic sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Read the signal from the sensor
    pinMode(echoPin, INPUT);
    duration = pulseIn(echoPin, HIGH);

    // Convert the time into a distance
    readings[i] = (duration / 2) / 29.1; // Divide by 29.1 or multiply by 0.0343

    delay(10); // Delay between readings to avoid interference
  }

  // Sort the array to easily discard outliers
```

```
  for (int i = 0; i < numReadings - 1; ++i) {
    for (int j = i + 1; j < numReadings; ++j) {
      if (readings[i] > readings[j]) {
        // Swap values
        unsigned long temp = readings[i];
        readings[i] = readings[j];
        readings[j] = temp;
      }
    }
  }

  // Calculate the average of the middle readings
  long sum = 0;
  for (int i = 1; i < numReadings - 1; ++i) {
    sum += readings[i];
  }
  long avg_distance = sum / (numReadings - 2);

  if (avg_distance > max_distance) {
    avg_distance = max_distance;
  }

  return avg_distance;

}
```

# arduinoReader.py (aflezen van de Arduino data en verzenden naar database)

```
import mysql.connector
import serial
from datetime import datetime
import os
import glob
import requests
import paramiko

# SSH Configuration
ssh_config = {
    'hostname': '85.10.147.119',
    'username': 'martin',
    'password': 'ugotmail',
    'port': 22,
}

port = serial.Serial("/dev/ttyACM0", baudrate=9600, timeout=3.0)

# MySQL Database Configuration
mysql_config = {
    'user': 'root',
    'password': 'ugotmail',
    'database': 'mailbox_db',
    'host': '85.10.147.119',
    'port': 3306,
}

# Establish SSH tunnel
with paramiko.SSHClient() as ssh:
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(**ssh_config, allow_agent=False, look_for_keys=False, compress=True)
    ssh_port = ssh.get_transport().request_port_forward('', mysql_config['port'])

    # Update MySQL configuration with the SSH tunnel
    mysql_config['port'] = ssh_port
    mysql_config['host'] = '85.10.147.119'

    # Connecting to mailbox database
    mydb = mysql.connector.connect(**mysql_config)
    mycursor = mydb.cursor()
```

```
sql = "INSERT INTO raw_data (delivery_date, delivery_time, weight, ultrasonic_distance, image) VALUES (%s, %s, %s, %s, %s)"

takePicture = "./takepicture.sh"
pic_just_taken = False

pi_images_path = "/home/pi/smartmailbox-api/public/storage/img/*"
image_local_path = "/storage/img/"

image_ready = False
data_ready = False

laravel_event_trigger_url = 'http://85.10.147.119:8000/api/trigger'
laravel_shared_data_url = 'http://85.10.147.119:8000/api/receive-data'

while True:

    arduino_line = port.readline().decode("utf-8").strip()
    print(arduino_line)

    if arduino_line != "":
        try:
            if arduino_line == 'take picture':
                os.system(takePicture)
                pic_just_taken = True
                images = glob.glob(pi_images_path)
                image_filename = max(images, key=os.path.getctime).split("/")[-1]
                image = image_local_path + str(image_filename)
                image_ready = True
            elif arduino_line == "emptied":
                data_to_share = {"emptied": "True"}
                response = requests.post(laravel_shared_data_url, json=data_to_share)
            else:
                data = list(arduino_line.split(","))
                weight = int(float(data[0]))
                ultrasonic_distance = int(data[1])
                data_ready = True
        except:
            print("^ this is not data!")

    if image_ready and data_ready:
        current_datetime = datetime.now()
        formatted_date = current_datetime.date().strftime('%Y-%m-%d')
```

```
        formatted_time = current_datetime.time().strftime('%H:%M:%S')

        values = (formatted_date, formatted_time, weight, ultrasonic_distance, image)
        print("inserting data..")
        mycursor.execute(sql, values)
        mydb.commit()

        # Trigger the event in Laravel by sending a POST request
        payload = {'data': {'delivery_date': formatted_date, 'delivery_time': formatted_time, 'weight': weight, 'ultrasonic_distance':
ultrasonic_distance, 'image': image}}
        response = requests.post(laravel_event_trigger_url, json=payload)

        image_ready = False
        data_ready = False
```

## takepicture.sh (bash script voor maken, opslaan en verzenden van de fotos)

```
#!/bin/bash

DATE=$(date +"%d-%m-%Y_%T")
LOCAL_IMAGE_PATH="/home/pi/smartmailbox-api/public/storage/img/$DATE.jpg"

# Take picture
fswebcam -r 640x480 --no-banner $LOCAL_IMAGE_PATH

# Send the image to the VPS using scp
REMOTE_USERNAME="martin"
REMOTE_HOST="85.10.147.119"
REMOTE_DIRECTORY="/home/martin/Documents/smartmailbox-api/public/storage/img/"
SSH_PASSWORD="ugotmail"

sshpass -p $SSH_PASSWORD scp $LOCAL_IMAGE_PATH $REMOTE_USERNAME@$REMOTE_HOST:$REMOTE_DIRECTORY
```

# DataController.php (uitvoeren van functies adhv routes in Laravel)

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use DB;
use Carbon\Carbon;
use App\Events\RawDataCreated;

class DataController extends Controller
{
    public function triggerEvent(Request $request) {
        // Extract data from the request
        $rawData = $request->input('data');

        // Dispatch the event
        event(new RawDataCreated($rawData));

        return response()->json(['message' => 'Event triggered successfully']);
    }

    public function getData() {
        // Fetch the data
        $data = DB::table('raw_data')
            ->orderBy('entry_created_at', 'desc')
            ->get();

        return response()->json($data);
    }

    public function getInbox() {
        // Fetch the inbox data
        $inbox = DB::table('inbox')
            ->orderBy('entry_created_at', 'desc')
            ->get();

        return response()->json($inbox);
    }

    public function receiveData(Request $request)
```

```
    {
        $data = $request->json()->all();
        info("data received!");
        event(new RawDataCreated($data));

        return response()->json(['status' => 'success']);
    }

}
```

# ProcessRawData.php (Laravel listener, verwerkt rauwe data tot bruikbare data)

```
<?php

namespace App\Listeners;

use App\Events\RawDataCreated;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use DateTime;
use DB;
use Cache;

class ProcessRawData implements ShouldQueue
{
    use InteractsWithQueue;

    private $max_distance = 35;
    private $prevTotalWeight = 0;

    /**
     * Create the event listener.
     *
     * @return void
     */
    public function __construct()
    {
        // No need for explicit initialization here
    }

    /**
     * Handle the event.
     *
     * @param  RawDataCreated  $event
     * @return void
     */
    public function handle(RawDataCreated $event)
    {
        // Access the raw data from the event
        $rawData = $event->rawData;
        $emptied = "False";
```

```
        if (isset($rawData['emptied'])) {
            $emptied = $rawData['emptied'];
            if ($emptied == "True") {
                // Update 'retrieved' to true for all previous entries
                DB::table('inbox')->update(['retrieved' => true]);
                // Reset prevTotalWeight and remove it from cache
                $this->prevTotalWeight = 0;
                Cache::forget('prevTotalWeight');
            }

        } else {
            // Retrieve the previous total weight from cache
            $prevTotalWeight = Cache::get('prevTotalWeight', 0);

            $totalWeight = $rawData['weight'];
            $weight = $totalWeight - $prevTotalWeight;

            $fulness_percentage = ($this->max_distance - $rawData['ultrasonic_distance']) / $this->max_distance * 100;

            $delivery_date = new DateTime($rawData['delivery_date']);
            $delivery_time = new DateTime($rawData['delivery_time']);

            // Logic to create an entry in the 'inbox' table
            DB::table('inbox')->insert([
                'delivery_date' => $delivery_date->format('d-m-Y'),
                'delivery_time' => $delivery_time->format('H:i'),
                'type' => ($weight > 50) ? "Package" : "Letter",
                'weight' => $weight,
                'fulness' => $fulness_percentage,
                'retrieved' => false,
                'image' => $rawData['image'],
            ]);
            // Update prevTotalWeight for the next event and store it in cache
            $prevTotalWeight = $totalWeight;
            Cache::forever('prevTotalWeight', $prevTotalWeight);
        }
    }
}
```

# api.dart

```
import 'dart:convert'; //Import for encoding and decoding of json data
import 'package:http/http.dart'; //Import for interacting with API
import 'dart:async'; // Import for asynchronous programming support
import 'package:intl/intl.dart'; //Import for date formatting
import 'package:shared_preferences/shared_preferences.dart'; //Import to use for saving variable states
import 'package:smartbrievenbus/util/notifications.dart'; //Import for the notifications

class ApiConnector {
  //Variables for the API connection url
  static final String API_HOST = "http://85.10.147.119:8000"; //If Flutter and Laravel run on the same device use: localhost or 127.0.0.1
for the url else use http://10.0.2.2:8000 for emulator
  static final String inboxDataurl = API_HOST + "/api/inbox";
  static final String rawDataurl = API_HOST + "/api/raw_data";

  //Variable for the 'filterdMail' that is later used by deleteHistory()
  List<Map<dynamic, dynamic>> filteredMail = [];
  List<Map<dynamic, dynamic>> filteredRaw = [];

  //Variable that checks if the 'filteredMail' needs to be used as data
  bool useFilteredData = false;

  //Variable for the data of the API
  late Future data;
  late Future dataRaw;

  //Variable for the Timer that checks if there is new data
  late Timer timer;

  late String notificationType;

  //The declaration of the callback function that gets called when there is new data
  late Function() onDataUpdate;

  Notifications notifications = Notifications();

  //The function that gets called when there is new data (Updates the UI)
  void setOnDataUpdate(Function() callback) {
    onDataUpdate = callback;
  }
```

```
//Initializes the API connection and activates a timer that checks for new data
void initialize() {
  //Get data from the API
  data = getData(inboxDataurl);
  dataRaw = getData(rawDataurl);

  //Timer to check for new data every x amount of time
  timer = Timer.periodic(
      const Duration(seconds: 60), (Timer timer) => updateData()
  );
}

void dispose() {
  //Cancel the timer to avoid memory leaks when the widget is destroyed
  timer.cancel();
}

//Function to reset the saved boolean by restart of the app
void resetValues() {
  useFilteredData = false;
  saveUseFilteredDataToSharedPreferences();
}

//Function to retrieve data from the API
Future getData(String url) async {
  //Establish a connection with the API based on the 'url' variable
  Response response = await get(Uri.parse(url));

  //Function to check the response from the API
  //If there is a successful connection, return the data
  if (response.statusCode == 200) {
    return json.decode(response.body);
  } else {
    //Else, tell there is an error occurred
    print(response.statusCode);
    throw Exception('Failed to fetch data');
  }
}

//Function to check if UI needs to be updated
void updateData() async {
  print("timer triggered"); //Print statement to test if the timer is triggered
```

```
  try {
    // Get the data from the API
    var newData = await getData(inboxDataurl);
    var newRawData = await getData(rawDataurl);

    // Check if using filtered data is true
    if (useFilteredData) {
      deleteHistory();
      deleteHistoryRaw();
      //notifications.showNotification();
    } else {
      // Check if new data is different from the filtered mail
      if (!compareData(await data, newData)) {
        print("Data is anders, dus update UI ");
        data = Future.value(newData);
        dataRaw = Future.value(newRawData);
        notificationType = determineNotificationType(newData);

        onDataUpdate();
        notifications.showNotification(notificationType);
      } else {
        // If the data is the same, do nothing
        print('data is hetzelfde geen reload nodig');
      }
    }
  } catch (error) {
    print(error);
  }
}

//Function to compare the json data from the API (currentData and newData)
bool compareData(dynamic currentData, dynamic newData) {
  //Encode the currentData and newData to json strings
  String jsonData = json.encode(currentData);
  String jsonNewData = json.encode(newData);

  //Compare the json strings and return true if they are equal, otherwise return false
  return jsonData == jsonNewData;
}

//Function to compare the Delivery date with the current date
String checkDeliveryDate(String deliveryDateStr) {
  //Get the current date
```

```
  DateTime currentDate = DateTime.now();
  //Change the Date format and convert it to a DateTime object
  DateTime deliveryDate =  DateFormat('dd-MM-yyyy').parse(deliveryDateStr);

  //If the date of 'currentDate' and 'deliveryData' are the same, return "Vandaag"
  if (deliveryDate.year == currentDate.year && deliveryDate.month == currentDate.month && deliveryDate.day == currentDate.day){
    return "Vandaag";
  } else {
    //Else, return the original date String
    return deliveryDateStr;
  }
}

//Function to change the Mailtype to Dutch
String checkMailType(String mailType) {
  if (mailType == "Package") {
    return "Pakket(je)";
  } else {
    return "Brief";
  }
}

//Function that updates the data based on "vandaag" or newer
Future deleteHistory() async {
  //Get the API data
  List mail = await getData(inboxDataurl);

  List previousFilteredMail = List.from(filteredMail);

  //Clear the filteredMail to save new data in it
  filteredMail.clear();

  //For each item in data, check if the date is "Vandaag" or newer. If so, add to filteredMail.
  for (Map singleMail in mail) {
    String deliveryDateStr = singleMail['delivery_date']?.toString() ?? '';
    String formattedDate = checkDeliveryDate(deliveryDateStr);

    if (formattedDate == "Vandaag") {
      filteredMail.add(singleMail);
    }
  }

  //Function to check if there is data == "Vandaag" otherwise do nothing
```

```
  if (filteredMail.isEmpty) {
    return;
  }

  //If we use FilteredData and there is new data, give a notification
  if (useFilteredData == true){
    if (!compareData(previousFilteredMail, filteredMail)) {
      notificationType = determineNotificationType(filteredMail);
      // If there is a difference, update the UI and show the notification
      notifications.showNotification(notificationType);
    }
  }

  //Change the variable of 'useFilteredData' to true
  useFilteredData = true;

  // Update the 'data' variable with 'filteredMail'
  data = Future.value(filteredMail);
  onDataUpdate();

  // Save the value of useFilteredData to shared preferences
  await saveUseFilteredDataToSharedPreferences();
  await saveFilteredMailToSharedPreferences();

  return filteredMail;
}

//Function that updates the data based on "vandaag" or newer
Future deleteHistoryRaw() async {
  //Get the API data
  List mail = await getData(rawDataurl);

  //Clear the filteredMail to save new data in it
  filteredRaw.clear();

  //For each item in data, check if the date is "Vandaag" or newer. If so, add to filteredMail.
  for (Map singleMail in mail) {
    String deliveryDateStr = singleMail['delivery_date']?.toString() ?? '';
    String formattedDate = checkDeliveryDate(deliveryDateStr);

    if (formattedDate == "Vandaag") {
      filteredRaw.add(singleMail);
    }
```

```
  }

  //Function to check if there is data == "Vandaag" otherwise do nothing
  if (filteredRaw.isEmpty) {
    return;
  }

  // Update the 'data' variable with 'filteredMail'
  dataRaw = Future.value(filteredRaw);
  onDataUpdate();

  // Save the value of useFilteredData to shared preferences
  await saveFilteredRawToSharedPreferences();

  return filteredRaw;
}

//Function to check if a new item in the mail is a package or a letter, and insert it into the notification
String determineNotificationType(dynamic dataType) {
  String mailType = dataType.first['type'];

  if (mailType == "Package") {
    return "nieuw pakket(je)";
  } else {
    return "nieuwe brief";
  }
}

//Functions to save and load the saved variables to use in the other screens
Future saveUseFilteredDataToSharedPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  prefs.setBool('useFilteredData', useFilteredData);
}

Future saveFilteredMailToSharedPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  prefs.setString('filteredMail', json.encode(filteredMail));
}

Future saveFilteredRawToSharedPreferences() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  prefs.setString('filteredRaw', json.encode(filteredRaw));
}
```

```
  Future loadUseFilteredDataFromSharedPreferences() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    useFilteredData = prefs.getBool('useFilteredData') ?? false;
    //print('Loaded useFilteredData: $useFilteredData');
  }

  Future loadFilteredMailFromSharedPreferences() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    String filteredMailString = prefs.getString('filteredMail') ?? '[]';
    filteredMail = (json.decode(filteredMailString) as List<dynamic>)
        .cast<Map<dynamic, dynamic>>();
    //print('Loaded filteredMail: $filteredMail');
  }

  Future loadFilteredRawFromSharedPreferences() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    String filteredRawString = prefs.getString('filteredRaw') ?? '[]';
    filteredRaw = (json.decode(filteredRawString) as List<dynamic>)
        .cast<Map<dynamic, dynamic>>();
    //print('Loaded filteredMail: $filteredRaw');
  }
}
```

# home.dart

```
import 'package:flutter/material.dart'; //Import the Flutter Material package for UI
import 'package:smartbrievenbus/api/api.dart'; //Import for the API connection
import 'appTheme.dart'; //Import for the app colors
import 'navigationBar.dart'; //Import for the navbar

class HomeScreen extends StatefulWidget {
  final ApiConnector apiConnector;

  HomeScreen({required this.apiConnector});

  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<HomeScreen> {
  //Variable that stores if the history clear has been done
  bool loadedHistory = false;

  //Create an instance of ApiConnector to manage API connections and fetch mailbox data
  final ApiConnector apiConnector = ApiConnector();

  //Variable to set the index for the navbar
  int _currentIndex = 0;

  //Variable to handle image loading errors
  Object? imageError;

  Future<void> loadHistoryIfNeeded() async {
    // Load useFilteredData and filteredMail only if the history has not been cleared
    if (!loadedHistory) {
      await apiConnector.loadUseFilteredDataFromSharedPreferences();
      await apiConnector.loadFilteredMailFromSharedPreferences();
      await apiConnector.loadFilteredRawFromSharedPreferences();
      loadedHistory = true;
    }
  }

  //Function to initialize the API connection and update the UI based on new data
  @override
  void initState() {
```

```
  super.initState();
  //Get the variables that has been saved
  apiConnector.loadUseFilteredDataFromSharedPreferences();
  apiConnector.loadFilteredMailFromSharedPreferences();
  apiConnector.loadFilteredRawFromSharedPreferences();

  //Call the 'setOnDataUpdate' function and update the UI if there is new data
  apiConnector.setOnDataUpdate(() {
    setState(() {
      //Because we are not adjusting anything specific in the UI, we can leave set state empty
    });
  });
  //Initialize the API connection
  apiConnector.initialize();
}

//Function to clean up resources and avoid memory leaks when the widget is destroyed
@override
void dispose() {
  apiConnector.dispose();
  super.dispose();
}

//Function to check the amount of mail that is not retrieved
int countNonRetrieved(List <dynamic> dataList) {
  int count = 0;

  for (Map<String, dynamic> data in dataList) {
    if (data['retrieved'] == 0) {
      count++;
    }
  }
  return count;
}

//Function for the build structure
@override
Widget build(BuildContext context) {
  return Scaffold(
    //The AppBar with the text "Nu"
    appBar: AppBar(
      title: Text("Nu",
        style: AppTheme.AppBarTitle,
```

```
  ),
  backgroundColor: AppTheme.primaryColor,
  elevation: 0.0,
),
//Function that handles the API connection and displays a UI based on the connection state
body: FutureBuilder(
  //Timer for connecting with the API before giving an error
  future: apiConnector.data.timeout(Duration(seconds: 30)),
  builder: (context, snapshot) {
    //If connecting to the API return a loading screen
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Container(
        color: AppTheme.primaryColor,
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              CircularProgressIndicator(),
              AppTheme.indicatorTextDistance,
              Text(
                'Zoeken naar brievenbus...',
                style: AppTheme.indicatorText,
              ),
            ],
          ),
        ),
      );
    } else if (snapshot.hasError) {
      //If timer is over or there is an error, return "Geen brievenbus gevonden"
      return Container(
        color: AppTheme.primaryColor,
        child: Center(
          child: Text(
            'Geen brievenbus gevonden',
            style: AppTheme.noConnectionText,
          ),
        ),
      );
    } else {
      //If there is a connection with the API build the widget with the API data
      var data = snapshot.data;
      return FutureBuilder(
        // Get the future for apiConnector.dataRaw
```

```
        future: apiConnector.dataRaw,
        builder: (context, rawSnapshot) {
          if (rawSnapshot.connectionState == ConnectionState.waiting) {
            return Container(
              color: AppTheme.primaryColor,
              child: Center(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    CircularProgressIndicator(),
                    AppTheme.indicatorTextDistance,
                    Text(
                      'Zoeken naar brievenbus...',
                      style: AppTheme.indicatorText,
                    ),
                  ],
                ),
              ),
            );
          } else if (rawSnapshot.hasError) {
            return Container(
              color: AppTheme.primaryColor,
              child: Center(
                child: Text(
                  'Geen brievenbus gevonden',
                  style: AppTheme.noConnectionText,
                ),
              ),
            );
          } else {
            // If raw data is available, build the widget with both data and dataRaw
            var dataRaw = rawSnapshot.data;
            if (apiConnector.useFilteredData) {
              data = apiConnector.filteredMail;
              dataRaw = rawSnapshot.data; // Use raw data here
            }
            return buildDataWidget(data, dataRaw);
          }
        },
      );
    }
  },
),
```

```
    //The navbar of the app
    bottomNavigationBar: NavBar(
      //Get the index of this app tab
      currentIndex: _currentIndex,
      //Function to change the navigation in the app based on the index
      onTap: (index){
        setState(() {
          _currentIndex = index;
        });
      },
      apiConnector: apiConnector,
    ),
  );
}

//Builds the UI with the data
Widget buildDataWidget(data, dataRaw) {
  int nonRetrievedCount = countNonRetrieved(data);
  return Container(
    color: AppTheme.primaryColor,
    alignment: Alignment.center,
    child: ListView(
      scrollDirection: Axis.vertical,
      padding: AppTheme.cardPadding,
      children: [
        //The container of the last picture "Laatste foto"
        Container(
          decoration: AppTheme.cardBoxDecoration,
          child: Padding(
            padding: AppTheme.inCardPadding,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                //Row with icon & text
                Row(
                  mainAxisAlignment: MainAxisAlignment.start,
                  children: [
                    Container(
                      width: AppTheme.iconCircleSize,
                      height: AppTheme.iconCircleSize,
                      decoration: AppTheme.boxDecorationCircle,
                      child: Icon(
                        Icons.camera_alt_outlined,
```

```
        color: AppTheme.text,
      ),
    ),
    AppTheme.iconTextWidth,
    Text(
      'Laatste foto',
      style: AppTheme.cardTitle,
    )
  ],
),
AppTheme.iconTextHeight,
//Row with picture and text
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Visibility(
      visible: (imageError == null),
      child: Padding(
        padding: EdgeInsets.only(left: 5.0),
        child: Text(
          apiConnector.checkDeliveryDate(data.first['delivery_date']),
          style: TextStyle(
            color: AppTheme.text,
          ),
        ),
      ),
    ),
    Container(
      child: Stack(
        alignment: Alignment.topLeft,
        children: [
          Padding(
            padding: EdgeInsets.fromLTRB(5.0, 4.0, 5.0, 5.0),
            child: ClipRRect(
              borderRadius: BorderRadius.circular(12.0),
              //Insert picture into "laatste foto" container. If 404 or other error show empty card
              child: Image.network(
                (ApiConnector.API_HOST + '${data.first['image']}'),
                fit: BoxFit.cover,
                errorBuilder: (BuildContext context, Object error, StackTrace? stackTrace) {
                  //Update the imageError variable
                  imageError = error;
                  return Text(
```

```
                        'Geen foto beschikbaar',
                        style: TextStyle(
                          color: AppTheme.accent,
                        ),
                      );
                    },
                  ),
                ),
              ),
              Visibility(
                visible: (imageError == null),
                child: Container(
                  margin: EdgeInsets.all(12.0),
                  padding: EdgeInsets.fromLTRB(10.0, 3.0, 10.0, 3.0),
                  decoration: AppTheme.cardBoxDecoration,
                  child: Text(
                    "${data.first['delivery_time']}",
                    style: TextStyle(
                      color: AppTheme.accent,
                    ),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    ],
  ),
),

//Distance between the 2 containers
AppTheme.betweenCardHeight,

//Container Mailbox data "Brievenbus data"
Container(
  height: 200,
  decoration: AppTheme.cardBoxDecoration,
  child: Padding(
    padding: AppTheme.inCardPadding,
    child: Column(
```

```
mainAxisAlignment: MainAxisAlignment.start,
children: [
  //Row with icon and text
  Row(
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
      Container(
        width: AppTheme.iconCircleSize,
        height: AppTheme.iconCircleSize,
        decoration: AppTheme.boxDecorationCircle,
        child: Icon(
          Icons.line_style,
          color: AppTheme.text,
        ),
      ),
      AppTheme.iconTextWidth,
      Text(
        'Brievenbus data',
        style: AppTheme.cardTitle,
      )
    ],
  ),
  SizedBox(height: 8),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      //Left text + circle
      Column(
        children: [
          Text(
            "Gewicht",
            style: AppTheme.mailboxDataText,
          ),
          AppTheme.textCircleHeight,
          Container(
            width: AppTheme.textCircleSize,
            height: AppTheme.textCircleSize,
            decoration: AppTheme.boxDecorationCircle,
            child: Center(
              child: Row(
                //Text in circle left
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
```

```
            Text(
              nonRetrievedCount == 0 ? "0" : "${dataRaw.last['weight']}",
              //"${dataRaw.last['weight']}",
              style: AppTheme.inCircleText,
            ),
            Text(
              'g',
              style: TextStyle(
                color: AppTheme.text,
              ),
            )
          ],
        ),
      ),
    ),
  ],
),
SizedBox(width: 50),
//Right Text + circle
Column(
  children: [
    Text(
      "Poststukken",
      style: AppTheme.mailboxDataText,
    ),
    AppTheme.textCircleHeight,
    Container(
      width: AppTheme.textCircleSize,
      height: AppTheme.textCircleSize,
      decoration: AppTheme.boxDecorationCircle,
      child: Center(
        child: Row(
          //Text in circle right
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              "$nonRetrievedCount",
              style: AppTheme.inCircleText,
            ),
          ],
        ),
      ),
    ),
```

```
                        ],
                      ),
                    ],
                  ),
                ],
              ),
            ),
          )
        ],
      ),
    );
  }
}
```