

18 FEBRUARY 2025

LEMMENS RENZO, STYNEN JEFFREY, VAN AEL WOUT, VAN BEEMEN  
NALANI, VERAGHTERT DEVIN, VERMANT JARNE



## ANPR MONITORING

RAPPORT REALISATIEFASE

OPDRACHTGEVER: CIPAL SCHAUBROECK – MARC CRUTZEN  
PROJECT 4.0 – THOMAS MORE

# Inhoudsopgave

Inhoudsopgave .....	1
Inleiding .....	3
Opdrachtbeschrijving .....	4
Aanpak.....	5
Werkwijze.....	5
Infrastructuur .....	6
Code Repository Platform .....	6
Private Cloud .....	6
AWS-diagram.....	7
DataFlow-diagram .....	8
Voorbeeld communicatieflow .....	8
Pipelines.....	10
SAST .....	10
Hardware (IoT) .....	13
Communicatie met applicatie .....	13
Componentenlijst.....	14
Modemreset.....	14
Zekeringenstatus .....	15
Stroomkwaliteit.....	15
Data.....	17
Data Flow.....	17
Prioriteiten .....	18
Applicatie.....	21
Frontend .....	21
Login .....	21
Auth0 .....	21
Dashboard .....	22
Meldingen .....	22
Kasten.....	23
Communicatie Backend .....	23

Components .....	23
Paginas .....	24
Ruimte voor Aanpassingen .....	24
Backend .....	25
Auth0 beveiliging.....	25
Helpers .....	26
Externe Services .....	27
Data Model.....	28
Models .....	29
DTO's (Data Transfer Objects) .....	29
Gebruik van AutoMapper .....	29
Data Access Layer.....	30
Controllers .....	31
Database en Docker-Setup .....	31
Bronnen .....	32

# Inleiding

Deze opdracht is uitgevoerd in het kader van het vak Project 4.0, onderdeel van de opleidingsunit IT Factory aan Thomas More.

Contactgegevens:

Thomas More  
Campus Geel  
Kleinhoefstraat 4, 2440 Geel  
[www.thomasmore.be](http://www.thomasmore.be)

Cipal Schaubroeck  
Hoofdkantoor: Cipalstraat 3, 2440 Geel  
[www.cipalschaubroeck.be](http://www.cipalschaubroeck.be)

Wij zijn een team van zes studenten van Thomas More Geel IT Factory en we bevinden ons in het laatste jaar van onze bacheloropleiding. In het vak Project 4.0 bundelen we onze kennis, vaardigheden en ervaring om een complex en uitdagend project tot een succesvol einde te brengen. Dit vak daagt ons uit om multidisciplinair samen te werken en onze expertise te combineren met een sterke focus op innovatie en samenwerking.

Ons team bestaat uit drie verschillende afstudeerrichtingen: Application Development, Cloud & Cybersecurity en Artificial Intelligence. Samen werken we aan een project dat we uitvoeren in opdracht van Cipal Schaubroeck, gevestigd in Geel. Zij hebben ons de kans gegeven om een belangrijke bijdrage te leveren aan de optimalisatie van hun ANPR-systemen (Automatic Number Plate Recognition).

Lemmens Renzo – Cloud & Cybersecurity  
Stynen Jeffrey - Ai  
Van Ael Wout – Application development  
Van Beemen Nalani – Cloud & Cybersecurity  
Veraghtert Devin - Application development  
Vermant Jarne – Application development

## Opdrachtbeschrijving

De opdracht richt zich op het verbeteren en uitbreiden van de functionaliteiten van ANPR-camera's, met bijzondere aandacht voor de monitoring en het beheer ervan. Momenteel worden via het bestaande monitoringsysteem (Zabbix) een grote hoeveelheid meldingen gegenereerd, maar ontbreekt het aan een effectieve classificatie en prioritering van deze meldingen. Dit zorgt ervoor dat belangrijke of kritieke problemen niet altijd de benodigde aandacht krijgen en dat beheerders worden overspoeld met meldingen die niet direct actie vereisen.

Ons doel is om dit proces te optimaliseren door de meldingen te filteren en te structureren op basis van prioriteit. We willen een duidelijk onderscheid maken tussen kritieke problemen, die onmiddellijke aandacht vereisen, en minder urgente kwesties, zodat de beheerders efficiënt kunnen handelen.

Kortom, deze opdracht combineert verschillende technologieën om de monitoring en analyse capaciteit te verhogen, en de werkdruk van de incident responders te verlagen. Doordat de meldingen worden geclassificeerd op prioriteit. Zo is het direct duidelijk welke meldingen direct actie vereisen.

## Aanpak

Ons project ging midden september van start met het analyseren van de noden en vereisten van onze klant. Vervolgens hebben we dit gegoten in een gedetailleerd projectplan. Op basis hiervan hebben we een strategie ontwikkeld om deze vereisten te vertalen naar een praktisch uitvoerbaar plan. Dit omvatte uitgebreid onderzoek en het maken van belangrijke keuzes met betrekking tot de te gebruiken software, programmeertalen en infrastructuur.

Na de planningsfase zijn we gestart met de realisatiefase, die is opgedeeld in drie sprints van telkens één week. Aan het begin van elke sprint stellen we concrete doelen op, die we tegen het einde van die week willen behalen. Na de derde sprint zijn we tot een uitgewerkt eindproduct gekomen.

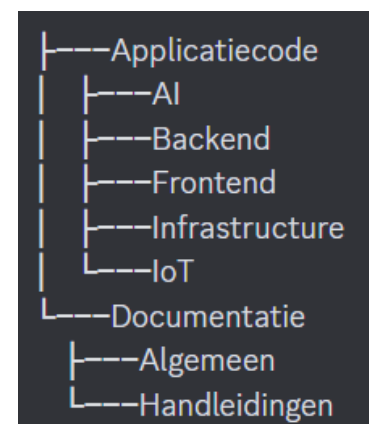
De volgende pagina's geven een overzicht van de technische details en beschrijven hoe wij tot ons eindresultaat zijn gekomen. Hierin belichten we de belangrijkste aspecten op het gebied van infrastructuur, IoT, data-analyse en applicatieontwikkeling.

## Werkwijze

In dit document verwijzen we naar effectieve code. Deze code is gebundeld in één zipbestand, met de naam *SAiD.zip*. Dit zipbestand heeft volgende structuur, achter elke tweede laag directory zitten de files die code bevatten, om het overzicht te bewaren staan deze hier niet mee op.

Doorheen het volledig document hanteren we één verwijzingsmethode. Dit is een voorbeeld van een verwijzing: (*Applicatiecode – AI/specifieke file*), wanneer er naar een specifieke file wordt verwezen kan het als volgt geschreven worden: (*Applicatiecode – AI/EDA.ipynb*).

Sommige directories bevatten meer subdirectories. Wanneer dit het geval is gaat de structuur verder als we begonnen zijn, bijvoorbeeld: (*Applicatiecode – Backend – Models/Alert.cs*).





## Infrastructuur

In dit project is een robuuste infrastructuur essentieel, aangezien de applicatie gehost moet worden. Als cloud- en cybersecuritystudenten richten we ons op een veilige en efficiënte infrastructuur, waarbij automatisering via Terraform en CI/CD-pipelines in GitLab een belangrijke rol speelt.

### Code Repository Platform

Voor het opzetten en beheren van onze infrastructuur maken we gebruik van Infrastructure as Code (IaC), waarbij we GitLab gebruiken als code repository. Onze repositorystructuur is georganiseerd in vijf afzonderlijke repositories, elk gericht op een specifiek onderdeel van het project.

`rl-terraform-pipeline`   
`2a959a65` · Deploy new version o

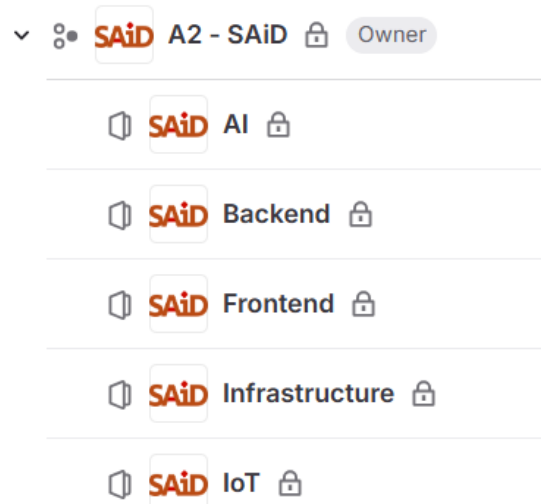
`nvb-pipelineaws`   
`826db438` · Merge branch 'main' i

Bij het beheer van onze repositories hanteren we een branching strategy die een combinatie vormt van feature branching en personal branching. Zoals geïllustreerd in de afbeelding hierlangs, beginnen de branches met onze initialen, gevolgd door een beschrijvende naam van het onderwerp waaraan we werken. Deze aanpak zorgt voor een overzichtelijke structuur en bevordert een efficiënte samenwerking binnen het team.

### Private Cloud

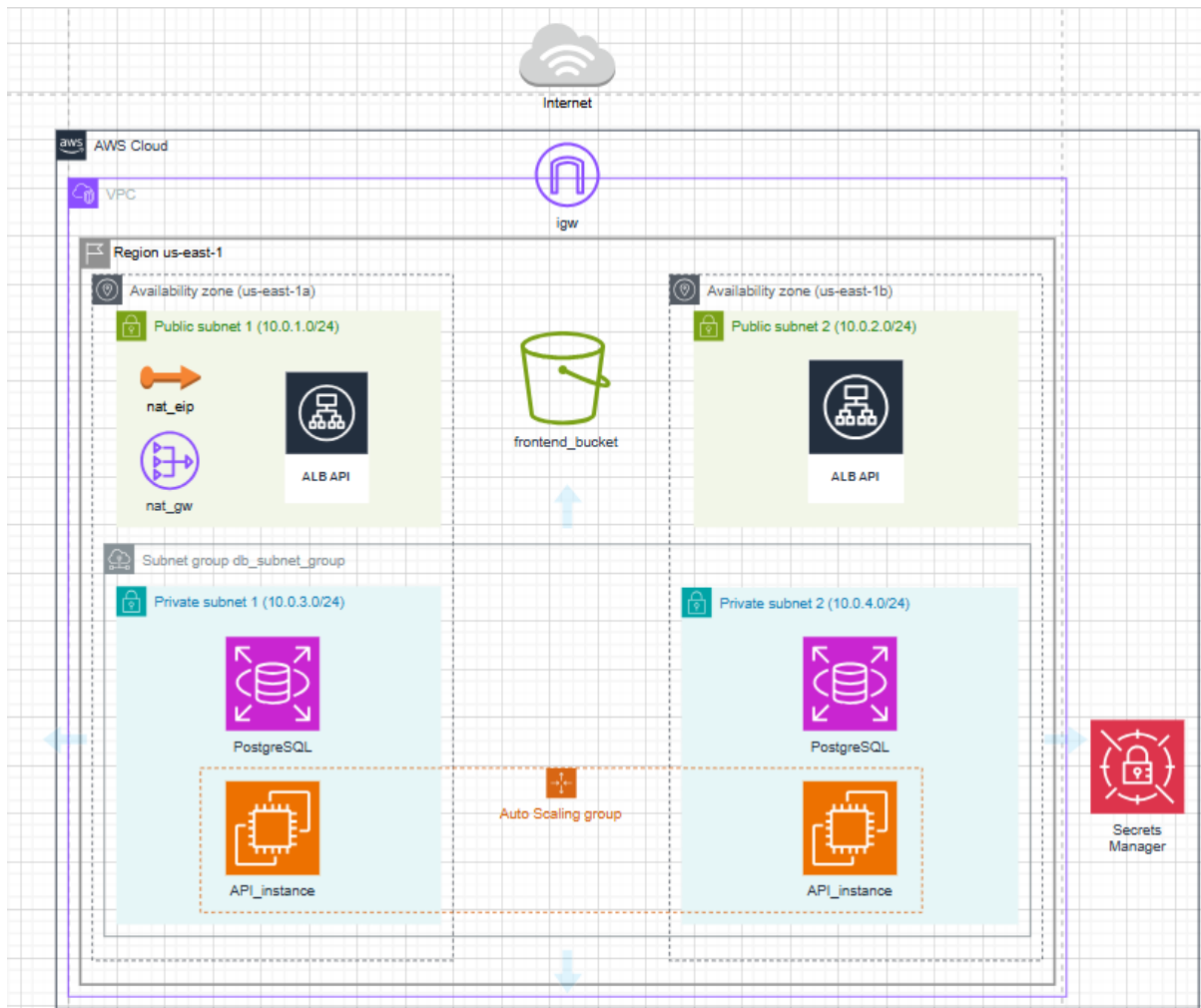
Voor onze cloudomgeving hebben we gekozen voor Amazon Web Services (AWS) vanwege de gebruiksvriendelijkheid en de beschikbaarheid van gratis krediet voor studenten, wat ons toelaat om kostenefficiënt te werken. AWS biedt bovendien een breed scala aan diensten die aansluiten bij de eisen van ons project, zoals schaalbaarheid, betrouwbaarheid en flexibiliteit.

Om onze infrastructuur te automatiseren, maken we gebruik van **Terraform**, waarmee we code hebben geschreven voor het aanmaken en configureren van de benodigde resources in AWS. Deze Terraform-configuraties worden geautomatiseerd uitgevoerd via CI/CD-pipelines in GitLab.



## AWS-diagram

Onze (cloud)infrastructuur ziet er als volgt uit:



In de AWS Cloudomgeving zitten er twee ‘availability zones’. Dit zorgt voor redundantie.

In iedere zone werken we met een publiek en privaat subnet, waarbij in het publieke subnet resources worden geplaatst die rechtstreeks blootgesteld zijn aan het internet:

- S3 Bucket → frontend applicatie (Angular)
- Application Load Balancer → backend API (.NET)

In het privaat subnet daarentegen, zitten de kwetsbare resources zoals de:

- Database (PostgreSQL)
- Backend API (.NET)

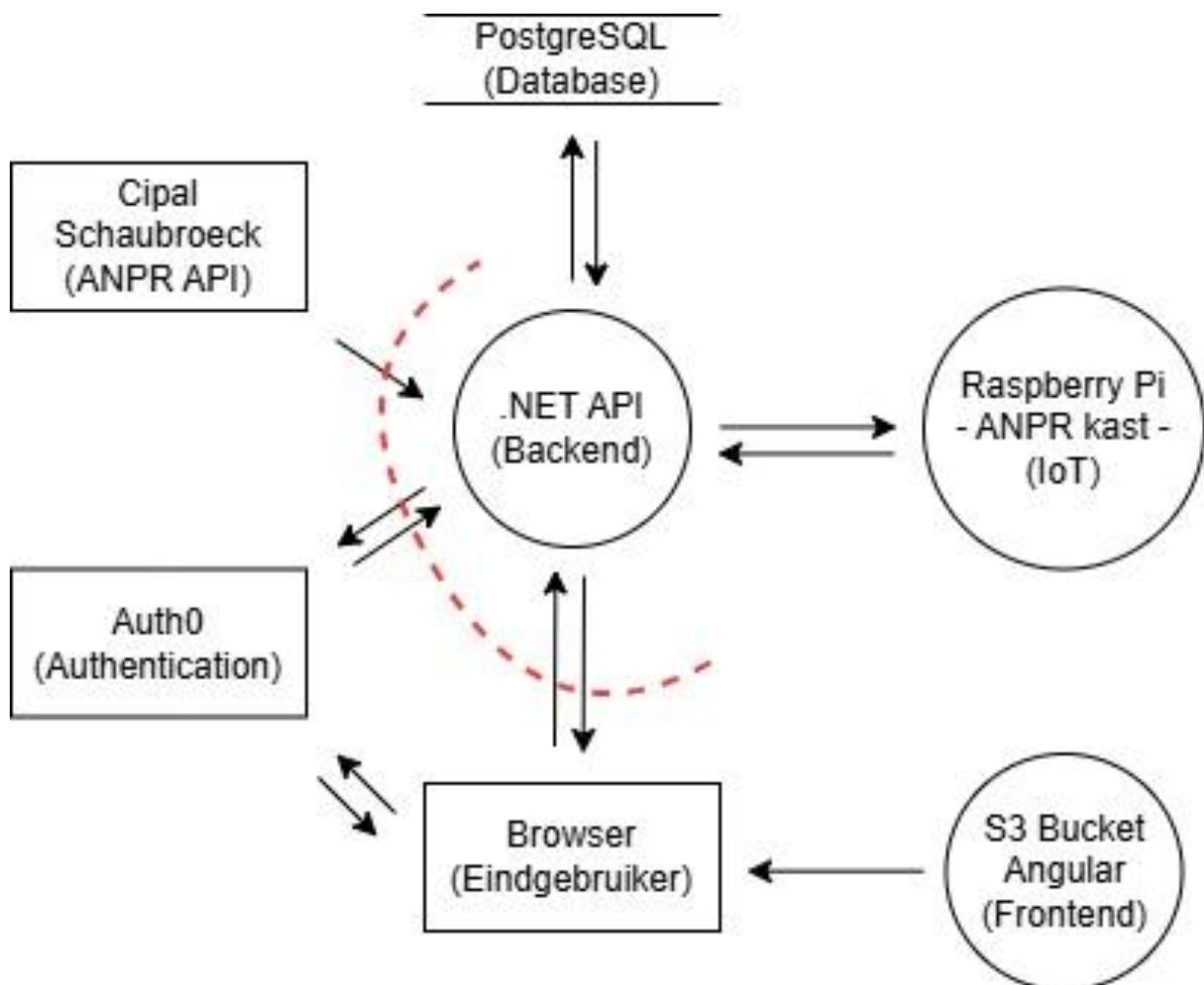
Schaalbaarheid zijn we ook zeker niet vergeten: de API Instances worden automatisch naar wens geschaald door de Autoscaling Group.

AWS Secrets Manager wordt dan weer gebruikt om database credentials op een veilige manier te bewaren en later op te halen voor database authenticatie in de backend.



## DataFlow-diagram

Hieronder is afgebeeld uit welke services ons project bestaat en wat met elkaar kan communiceren:



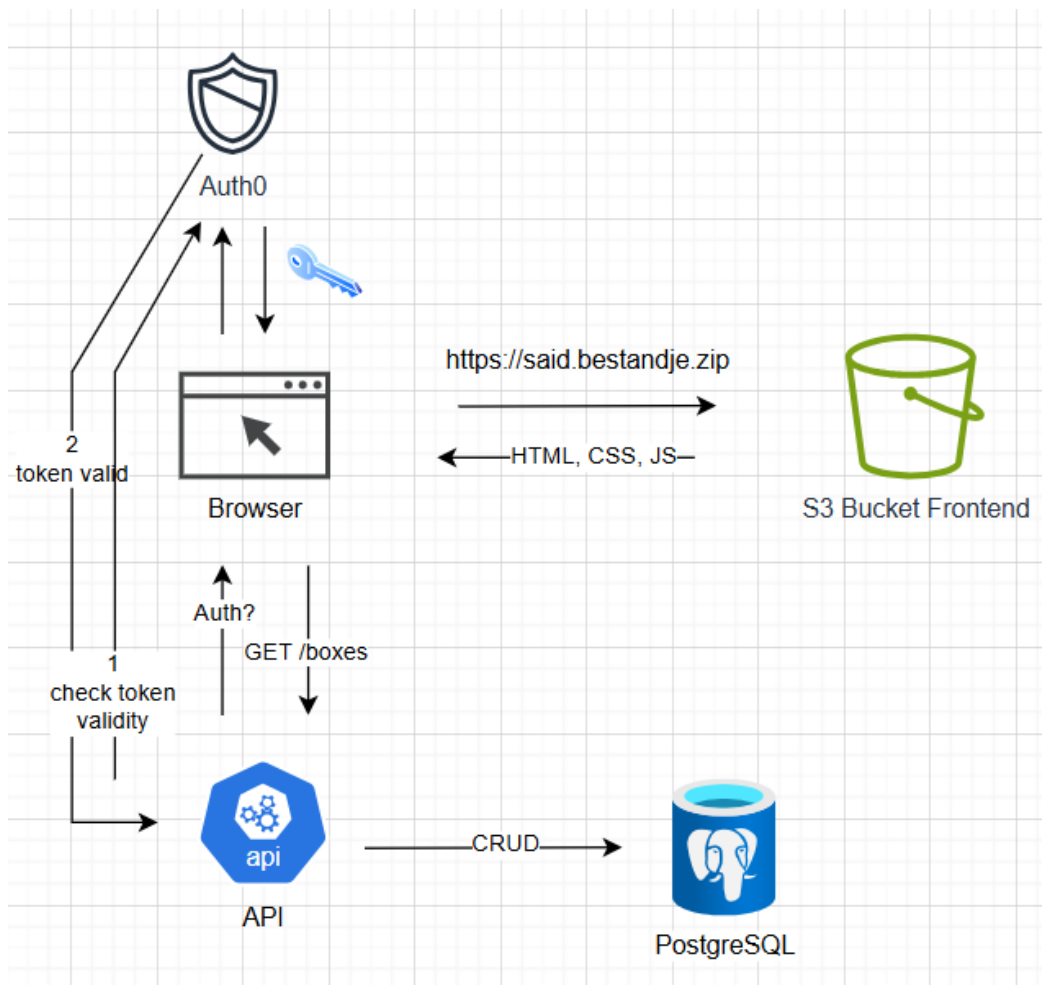
Zo valt op dat er een trust-boundary gelegd wordt tussen onze eigen componenten en externe entiteiten. We maken gebruik van **Auth0** en de **ANPR API** van **Cipal Schaubroeck**, net zoals de gebruikers zelf hun **browser** kiezen waarmee ze gebruik maken van onze applicatie. We hebben deze niet in handen.

Wat we wel in handen hebben, dat zijn:

- **Frontend** (Angular applicatie) → AWS S3 Bucket
- **Backend** (.NET API) → AWS EC2
- **Database** (PostgreSQL) → AWS RDS
- **IoT** (Raspberry Pi) → Hardware ANPR kast

## Voorbeeld communicatieflow

In het onderstaande diagram wordt er afgebeeld hoe de communicatie tussen de verschillende services verloopt, in een situatie waarbij de gebruiker een geldige login opgeeft en rechten heeft om in te loggen op de applicatie:



1. Als de eindgebruiker in zijn browser surft naar onze website <https://said.bestandje.zip>, haalt de browser de HTML, CSS en JS files op waardoor de eindgebruiker de website te zien krijgt.
2. Omdat de applicatie client-side is en er moet ingelogd worden op de website, is er authenticatie nodig.
3. Auth0, onze gekozen oplossing voor authenticatie, zal de eindgebruiker na het inloggen voorzien van een token.
4. Deze token wordt door de browser gebruikt om de data op te halen uit de backend API. Deze ontvangt de token van de browser en controleert bij Auth0 op geldigheid.
5. Bij een geldige token zal de backend API de gevraagde data ophalen zodat de browser dit kan weergeven op de pagina.
6. In de achtergrond communiceert de API met de PostgreSQL database.

## Pipelines

Voor elke repository binnen ons project hebben we een afzonderlijke pipeline ingericht. Dit zorgt ervoor dat wijzigingen of implementaties specifiek per onderdeel kunnen worden getest en uitgerold.

## SAST

De pipelines van alle repositories zijn uitgerust met Static Application Security Testing (SAST). Dit wordt gebruikt om kwetsbaarheden te detecteren in de code. Hiervoor gebruiken we een template van gitlab zelf. Deze test worden altijd doorlopen in de stage “test”.

```
include:
  - template: Jobs/SAST-IaC.gitlab-ci.yml #gitlab SAST template
```

Wanneer deze “test” stage is doorlopen is er altijd een rapport beschikbaar. Deze kan teruggevonden worden onder “Secure” → “Vulnerability report”. Hier is heel veel relevante informatie terug te vinden. De meldingen zijn zeer duidelijk en accuraat, een mooi hulpmiddel met het schrijven van code.

### Vulnerability report

[+ Submit vulnerability](#) [⬆ Export](#)

The Vulnerability Report shows results of successful scans on your project's default branch, manually added vulnerability records, and vulnerabilities found from scanning operational environments. [Learn more](#).

Development vulnerabilities 77 Operational vulnerabilities 0 Container registry vulnerabilities 0

Security reports last updated 20 hours ago #1674854280

<div><div></div>Critical</div> <div>4</div>	<div><div></div>High</div> <div>0</div>	<div><div></div>Medium</div> <div>15</div>	<div><div></div>Low</div> <div>0</div>	<div><div></div>Info</div> <div>58</div>	<div><div></div>Unknown</div> <div>0</div>
---	---	--	--	--	--

Group by: None ▾

Status || Needs triage, Confirmed ✕ Activity || Still detected ✕

<input type="checkbox"/>	Detected	Status	Severity ▾	Description	Identifier	Tool	Activity
<input type="checkbox"/>	2025-02-12	Needs Triage	Critical	DB Instance Storage Not Encrypted <a href="#">aws/main.tf:490</a>	DB Instance Storage Not Encrypted	SAST	
<input type="checkbox"/>	2025-02-12	Needs Triage	Critical	S3 bucket allows public policy <a href="#">aws/main.tf:184</a>	S3 Bucket Allows Public Policy	SAST	
<input type="checkbox"/>	2025-02-12	Needs Triage	Critical	Sensitive Port Is Exposed To Entire Network <a href="#">aws/main.tf:403</a>	Sensitive Port Is Exposed To Entire Network	SAST	
<input type="checkbox"/>	2025-02-12	Needs Triage	Critical	Unrestricted Security Group Ingress <a href="#">aws/main.tf:408</a>	Unrestricted Security Group Ingress	SAST	

## Pipeline Backend

Voor de backend hebben we twee pipelines voorzien. De eerste pipeline is `.gitlab-ci.yml` (*Applicatiecode – Backend/.gitlab-ci.yml*). Deze pipeline bevat de SAST template en bevat de verschillende stages. Hier wordt ook verwezen naar de tweede pipeline

De pipeline `main.gitlab-ci.yml` is verantwoordelijk voor het bouwen van de Docker-image. Vervolgens wordt deze image gepubliceerd in de Container Registry van de Gitlab Repository Backend (*Applicatiecode – Backend/main.gitlab-ci.yml*).

### *Pipeline Frontend*

Ook de Frontend repo bevat twee verschillende pipelines. De `.gitlab-ci.yml` pipeline bevat de SAST template en de stages die worden doorlopen (*Applicatiecode – Frontend/.gitlab-ci.yml*). Hier wordt dan ook verwezen naar de tweede pipeline, `main.gitlab-ci.yml`.

Deze pipeline automatiseert het proces van het bouwen, verpakken en publiceren van de frontend bestanden naar de Gitlab Package Registry. De bestanden worden verzonden in een gecomprimeerd zipbestand (*Applicatiecode – Frontend/main.gitlab-ci.yml*).

De eerste fase, `build_frontend`, zorgt voor het installeren van de benodigde dependencies met behulp van npm en bouwt de frontend-applicatie. Vervolgens voert de tweede fase, `package_frontend`, een bundeling uit van alle relevante frontend-bestanden in een zipbestand. Dit zipbestand wordt gedefinieerd als een artefact en blijft gedurende één uur beschikbaar voor de volgende stap.

In de laatste fase, `publish_to_package_registry`, wordt het zipbestand geüpload naar de GitLab Package Registry van de repository.

Door de frontend-bestanden in een zipbestand te verpakken en op te slaan in de Package Registry, kunnen deze bestanden eenvoudig worden opgehaald door andere repositories. Dit maakt het mogelijk om de bestanden vervolgens in een ECS-omgeving te implementeren met de infrastructure repository.

### *Pipeline infrastructure*

Deze pipeline automatiseert het proces van het downloaden van de frontend-artifacten (zip bestand), implementeren van de infrastructuur met Terraform en het opschonen van de resources. De pipeline bestaat uit 4 fasen: `test` (standaard fase voor SAST), `download`, `apply` en `destroy`. Ook hier wordt gebruik gemaakt van het SAST template, en verwezen naar de `TF.gitlab-ci.yml` pipeline. Extra aan deze pipeline is de workflow. Dit zorgt ervoor dat de pipeline enkel runt wanneer de commits uitgevoerd worden op de “main” of “test” branch (*Applicatiecode – Infrastructuur/.gitlab-ci.yml*).

De eerste fase “kics-iac-sast” doorloopt de volledige code met de SAST tool. Hier wordt gecheckt op vulnerabilities.

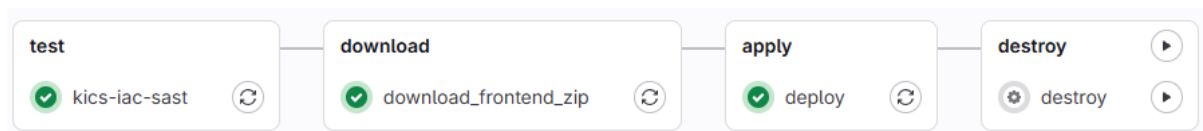
De tweede fase, 'download', richt zich op het ophalen van het zipbestand dat eerder is opgeslagen in de Package Registry van de frontend-repository. Dit zipbestand bevat de gebundelde frontend-bestanden en wordt gedownload met behulp van een privé-token. Deze token verleent de benodigde toegangsrechten om de bestanden te mogen ophalen uit de package registry. Na het downloaden wordt gecontroleerd of het bestand succesvol is opgehaald.

De derde fase, 'deploy', zorgt voor de daadwerkelijke implementatie van de infrastructuur en de frontend. De zip wordt uitgepakt, waarna de inhoud wordt voorbereid voor gebruik.

Vervolgens worden wijzigingen toegepast op de Terraform-configuratie met behulp van OpenTofu. Hierbij doorloopt de pipeline drie stappen (*Applicatiecode – Infrastructuur/TF.gitlab-ci.yml*):

1. **tofu init** – Initialiseert de Terraform-backend en zorgt dat alle configuraties correct worden geladen.
2. **tofu plan** – Genereert een plan waarin gedetailleerd wordt weergegeven welke wijzigingen op de infrastructuur worden toegepast.
3. **tofu apply** – Voert de geplande wijzigingen automatisch uit, waardoor de infrastructuur wordt bijgewerkt en de frontend wordt gedeployed.

De vierde fase, '**destroy**', is een handmatige functie die uitsluitend wordt uitgevoerd wanneer dit expliciet wordt geactiveerd via de pipeline-interface. Deze functie verwijdert alle geconfigureerde AWS-resources, waaronder instances, security groups, VPC's en andere infrastructuurelementen.



## Hardware (IoT)

Voor een betrouwbare monitoring van ons systeem maken we gebruik van verschillende IoT-componenten in de stroomkast. De kern van deze opstelling is een **Raspberry Pi 4B**, die verantwoordelijk is voor het aansturen en verzamelen van gegevens.

De belangrijkste functionaliteiten van dit systeem zijn:

- **Herstarten van de modem op aanvraag** bij verbindingsproblemen
- **Uitlezen van de status van de zekeringen** om stroomstoringen te detecteren
- **Monitoren van de stroomkwaliteit** om afwijkingen of problemen vroegtijdig op te sporen
- **Camera-observatie binnenin de kast** om op afstand fysieke problemen te controleren, zoals de statuslampjes van de modem of een uitgeschakelde zekering

## Communicatie met applicatie

Aangezien onze applicaties gehost worden op AWS, is er geen fysieke verbinding tussen deze infrastructuur en de hardware in de stroomkast. Naast de connectiviteit is ook centraal beheer van alle IoT-componenten cruciaal.

Amazon Web Services biedt hiervoor een oplossing: **AWS Greengrass / IoT Core**. Hiermee kunnen bijvoorbeeld certificaten geïnstalleerd worden op IoT-devices, zoals een Raspberry Pi, waardoor deze via MQTT kunnen communiceren met AWS IoT Core. AWS Greengrass maakt het bovendien mogelijk om deze apparaten op afstand te beheren. (Amazon, n.d.)

Voor onze **Proof-of-Concept** hebben we echter een oplossing nodig die flexibel en gebruiksvriendelijk is. AWS IoT Core voldoet hier niet aan om de volgende redenen:

1. We resetten de omgeving dagelijks om kosten te besparen, waardoor de certificaten op de Raspberry Pi telkens vervallen.
2. Na uitgebreid onderzoek vonden we geen manier om met AWS IoT Core snapshots naar de API te versturen – een beperking van MQTT.

Daarom kiezen we voor **Tailscale**, een gratis VPN-oplossing gebaseerd op WireGuard. Geen complexe configuratie of certificaten nodig: gewoon installeren, inloggen (via SSO of Auth Key) en klaar! (Tailscale, n.d.)

We vernemen dat Cipal Schaubroeck voor een andere oplossing zal kiezen aangezien zij applicaties hosten in hun eigen datacenter en een soort privé Proximus-netwerk hebben voor de stroomkasten.

## Componentenlijst

Naam	Beschrijving	Link	Alternatief
Raspberry Pi 4B Rev. 1.4 4 GB	Een microcontroller nodig voor de aansturing van de andere componenten. Vormt de communicatiebrug tussen cloud-infrastructuur en hardware in de kasten.	<a href="#">Raspberry Pi 4B 4 GB</a>	<a href="#">PiZero</a>
Raspberry Pi 4b batterij	Een batterij die ingeschakeld wordt bij een stroompanne om communicatie tussen hardware in de stroomkast en cloud te verzekeren.	<a href="#">Raspberry Pi 4b batterij</a>	Generic UPS
Shelly Pro EM-50	Een slimme zekering die gebruikt kan worden om componenten te schakelen en uit te meten.	<a href="#">Shelly Pro EM-50</a>	<a href="#">Waveshare Industrial 6-ch Relay Module for Raspberry Pi Zero</a> (enkel Relais, geen stroommeter)
Raspberry Pi 4b IR camera module	Een infraroodcamera voor in de stroomkast, zo is er geen nood meer aan een extra verlichtingsbron.	<a href="#">Raspberry Pi camera module</a>	USB-webcam met verlichting of externe lichtbron

Voor de aansturing van alle componenten en het verzenden van data naar de cloud maken we gebruik van een **Raspberry Pi 4B**, voorzien van een batterij als back-up om continuïteit te garanderen.

Voor metingen binnen de schakelkast, waarin de microcontroller zich bevindt, gebruiken we een **Shelly Pro EM-50**. Dit stelt ons in staat om cruciale parameters uit te lezen en de monitoring te optimaliseren. Zo kunnen we bijvoorbeeld op afstand de status van de zekeringen controleren.

Daarnaast is een camera geïnstalleerd die op aanvraag een foto van de binnenkant van de schakelkast maakt en deze naar de applicatie verstuurt. Dit vergemakkelijkt de visuele inspectie van alle componenten op afstand, waardoor een snelle en efficiënte beoordeling van de systeemstatus mogelijk is.

## Modemreset

Voor het op afstand resetten van de modem maken we opnieuw gebruik van de Shelly. Via de Raspberry Pi kunnen we de Shelly activeren om de stroomtoevoer naar de modem gedurende 10 seconden te onderbreken. Hierdoor verliest de modem tijdelijk zijn voeding en start hij automatisch opnieuw op. Dit proces wordt eenvoudig uitgevoerd door een request te sturen naar de Shelly met de volgende opdracht:

```
http://{self.shelly_ip}/rpc/Switch.Set?id={self.relay_index}&on={'true' if  
state else 'false'}
```

Om een volledige reset te garanderen, is het essentieel om enkele seconden te wachten voordat de stroomtoevoer wordt hersteld. Door een wachttijd van 10 seconden in te bouwen, zorgen we ervoor dat de reset correct wordt uitgevoerd. (*Applicatiecode – IoT/app/modem\_reset.py*)

Daarnaast kan de Shelly zo geconfigureerd worden dat deze standaard een gesloten circuit behoudt en dus continu voeding levert aan de modem. We hebben een script ontwikkeld dat ervoor zorgt dat de Shelly automatisch opnieuw inschakelt na 10 seconden, ongeacht wat er gebeurt—of iemand handmatig een knop indrukt of de microcontroller crasht. Hierdoor blijft de modem altijd operationeel.

Als alternatief kan de reset ook worden uitgevoerd via GPIO. Om dit te doen, dient de GPIO pin die aangesloten is op de sturing van de relais af gezet worden voor 10 seconden. Ook moeten we ervoor zorgen dat er een GPIO cleanup gebeurt voor en na het schakelen van de GPIO pin. Dit zorgt voor een optimale werking. (*Applicatiecode – IoT/app/modem\_reset\_gpio\_relay.py*)

## Zekeringenstatus

Om de status van een zekering uit te kunnen lezen, analyseren we de data die door de Shelly wordt geregistreerd. Wanneer de Shelly geen data meer doorstuurt, of wanneer zowel de spanning (*voltage*) als de stroom (*current*) 0 zijn, kunnen we concluderen dat de zekering is uitgeschakeld. In dat geval wordt de status op *"false"* gezet. (*Applicatiecode – IoT/app/event.py*)

Daarnaast wordt automatisch een foto van de binnenkant van de stroomkast genomen wanneer de status *"false"* is. Hierdoor kan op afstand een visuele controle worden uitgevoerd om te verifiëren of de zekering daadwerkelijk uitgeschakeld is. Dit helpt bij het beter in kaart brengen van de aard van het probleem, zodat de juiste diensten kunnen worden ingeschakeld om het euvel te verhelpen. (*Applicatiecode – IoT/app/sensor\_monitor.py*)

Uiteraard is het ook mogelijk om handmatig een foto op te vragen van de binnenkant van de stroomkast. Hierdoor kan de volledige stroomkast op elk gewenst moment visueel worden gemonitord. Dit biedt niet alleen inzicht in de status van de zekeringen, maar maakt het bijvoorbeeld ook mogelijk om de status van de modem te controleren—zoals of alle lampjes nog branden. (*Applicatiecode – IoT/app/snapshot.py*)

## Stroomkwaliteit

De **stroomkwaliteit** kunnen we meten met de Shelly, die verschillende parameters registreert, zoals spanning, stroom, actief vermogen, reactief vermogen, actief vermogen



en frequentie. Voor onze **proof of concept** maken we voorlopig geen gebruik van actief en reactief vermogen.

Voor de overige waarden kunnen we per component grenswaarden instellen die een event kunnen triggeren. Dit betekent dat wanneer een grenswaarde wordt overschreden, bijvoorbeeld wanneer de grenswaarde voor spanning 240V is en de gemeten spanning 250V bedraagt, alle metingen van dat moment worden opgeslagen en doorgestuurd in **JSON-formaat** naar de back-end van de applicatie. Hierdoor krijgen we meer inzicht in mogelijke problemen en kunnen we afwijkingen beter analyseren. (*Applicatiecode – IoT/app/event.py*)

Hoewel de stroomkwaliteitsmetingen momenteel beperkt zijn tot de basisparameters zoals spanning, stroom en het actief vermogen, kan in een volgende fase de **RMS-spanning** en **Total Harmonic Distortion (THD)** toegevoegd worden. Deze twee parameters bieden waardevolle inzichten in de kwaliteit van de netspanning en helpen bij het identificeren van stroomvervalsingen. Let echter wel op, dat er meerdere metingen nodig zijn om deze berekeningen te kunnen maken, wat wil zeggen dat de code ook herzien zal moeten worden.

De **RMS-spanning** is een maat voor de effectieve waarde van de spanning in het systeem. In een wisselspanningssysteem wordt de spanning continu aangepast, en de RMS-waarde biedt een nauwkeuriger beeld van de werkelijke spanning die door een apparaat wordt ontvangen. De berekening van de RMS-spanning gebeurt als volgt:

$$V_{rms} = \sqrt{\frac{1}{N} \sum_{i=1}^N V_i^2}$$

Waarbij  $V_i$  de gemeten spanningswaarden zijn en  $N$  het aantal metingen is binnen een bepaalde periode.

**Total Harmonic Distortion (THD)** meet de vervorming in het spanningssignaal door de aanwezigheid van hogere harmonischen, die kunnen wijzen op netvervuiling of niet-lineaire belastingen. De formule voor THD is als volgt:

$$THD_V = \frac{\sqrt{\sum_{n=2}^{\infty} V_n^2}}{V_1} \times 100\%$$

Waarbij  $V_1$  de RMS-waarde van de fundamentele frequentie is (bijvoorbeeld 50 Hz) en  $V_n$  de RMS-waarden zijn van de hogere harmonischen.

Deze metingen kunnen waardevolle inzichten bieden voor het detecteren van spanningsvervalsingen of netstoringen, en zouden in een latere fase van de implementatie toegevoegd kunnen worden, afhankelijk van de behoefte en de complexiteit van het systeem.

## Data

De oorspronkelijk aangeleverde data bestond voornamelijk uit eenvoudige netwerkmeldingen, zoals *"Ping loss too high"*. Deze gegevens waren te beperkt om een machine learning-model effectief op te trainen. Dit komt doordat de data nauwelijks uitzonderingen bevatte, waardoor een regelgebaseerd systeem al voldoende functioneert. Een machine learning-model wordt pas nuttig wanneer de data complex en divers genoeg is dat het schrijven van regels onpraktisch of onmogelijk wordt. (*Applicatiecode – AI/EDA\_events.ipynb*)

Door te kiezen voor een regelgebaseerd systeem in plaats van een machine learning-model, wordt de monitoring eenvoudiger en efficiënter. Dit elimineert de noodzaak voor extra infrastructuur om een model te trainen, hosten of onderhouden. Bovendien kunnen we de regelgebaseerde software direct integreren in de back-end van de applicatie, wat niet alleen kosten bespaart, maar ook de operationele eenvoud en betrouwbaarheid vergroot.

Zelfs met de toevoeging van IoT-componenten, die de monitoringscapaciteit van de behuizing aanzienlijk uitbreiden, blijft een regelgebaseerd systeem een haalbare en effectieve oplossing. Door gebruik te maken van een gewogen som om prioriteiten te bepalen, biedt dit systeem een flexibele en schaalbare aanpak voor monitoring die zowel huidige als toekomstige behoeften kan ondersteunen.

**Let op:** Dit is slechts een proof of concept. Dit betekent dat nog niet alle functionaliteiten in het systeem zijn geïmplementeerd.

Momenteel meten we slechts één fictief component, waardoor we de gewogen som voor de volledige stroomkast nog niet kunnen berekenen. Daarnaast zijn de grenswaarden en normaalwaarden die een event triggeren en afwijkingen detecteren, momenteel hardcoded. Hier moet rekening mee worden gehouden bij de implementatie van het systeem in een productieomgeving.

## Data Flow

Om de nieuwe data effectief in het systeem te integreren, moet deze vanuit de microcontroller worden doorgestuurd naar de back-end van de applicatie. Het is cruciaal dat de data event-based is, wat betekent dat deze alleen wordt verzonden wanneer een anomalie wordt gedetecteerd. Dit helpt om de infrastructuurkosten te beperken. In de back-end kan de data vervolgens worden verwerkt, waarbij er een prioriteitswaarde aan wordt toegekend en de informatie wordt weergegeven in de front-end.

De structuur van de event-based data is als volgt:

```
{
  "Status": true,
  "Voltage": 12,
  "Current": 0.34,
  "ActivePower": 3.0,
  "Frequency": 50,
  "Ip": "192.168.1.100",
  "ComponentId": "comp123",
  "Timestamp": "2025-01-16T14:20:00Z",
  "Flag": "stroom te hoog"
}
```

Dit bericht bevat meerdere belangrijke velden. De **status** geeft aan of het component operationeel is of niet, terwijl: **Voltage**, **Current**, **ActivePower**, en **Frequency**. Respectievelijk de gemeten spanning, stroom, actief vermogen, en frequentie op het moment van het event weergeven. Het veld **Ip** bevat het IP-adres van de stroomkast waarin het component zich bevindt, en **ComponentId** is een unieke identificatie die gebruikt wordt om individuele componenten te onderscheiden. Het tijdstip waarop het event heeft plaatsgevonden, wordt opgeslagen in het veld **Timestamp**, wat opgeslagen kan worden als string. Ten slotte geeft het veld **Flag** de oorzaak van het event weer, zoals “stroom te hoog” of “spanning te laag”.

Zoals eerder vermeld, zal de microcontroller er voor zorgen dat data alleen gestuurd wordt bij het detecteren van een anomalie in de gemeten waarden. Deze zal dan een flag toekennen zodat de oorzaak van het event duidelijk gelogd kan worden. Als de flag van het event “down” is, word er ook automatisch een foto van de binnenkant van de stroomkast genomen. Zo kan het probleem ook handmatig nageken worden vanop afstand – of de zekering daadwerkelijk uitstaat.

## Prioriteiten

Om de prioriteit van een melding te bepalen, wordt gebruikgemaakt van een gewogen som. Het is belangrijk om vooraf uitzonderingen te filteren en de juiste gewichten toe te kennen aan de relevante parameters. Er zijn twee soorten meldingen: **netwerkmeldingen** en **statusmeldingen**, die afzonderlijk worden geëvalueerd. Hun prioriteiten worden uiteindelijk gecombineerd om de totale prioriteit van een stroomkast te berekenen. Deze aanpak maakt het mogelijk om flexibel om te gaan met verschillende soorten problemen, waarbij zowel de ernst als de duur van een probleem in aanmerking worden genomen.

Netwerkmeldingen omvatten drie belangrijke categorieën: **packet loss**, **latency** en **status**. Voor meldingen waarbij de netwerkstatus down is of als er sprake is van 100% packet loss, wordt direct de hoogste prioriteit toegekend:

$$P_{netwerk} = 1.0 \text{ als } (status = 1 \text{ of } packetloss = 100)$$

Dit reflecteert de kritieke aard van deze problemen, omdat de stroomkast in deze gevallen niet bereikbaar is. Als er echter sprake is van minder ernstige netwerkproblemen, zoals gedeeltelijke packet loss of verhoogde latency, wordt een genormaliseerde prioriteit berekend. De packet loss wordt genormaliseerd naar een schaal tussen 0 en 1 door het percentage packet loss te delen door 100:

$$P_{netwerk} = \frac{packetloss}{100}$$

Voor latency wordt een vergelijkbare aanpak gebruikt, waarbij de gemeten latency wordt gedeeld door een vooraf gedefinieerde maximale waarde (voorlopig 10000ms maar kan aangepast worden naar wens), met een limiet van 1 om de prioriteit binnen een consistente schaal te houden: (*Applicatiecode – Backend/Helpers/ZabbixHelper.cs*)

$$P_{netwerk} = \min\left(\frac{latency}{latency_{max}}, 1\right)$$

Om prioriteiten toe te kennen aan meldingen van stroomkasten, wordt rekening gehouden met de status en werking van alle componenten binnen een stroomkast, zoals de modem, switch, verwarming, ... Elk component heeft een eigen status en prioriteit, en deze worden gecombineerd om de totale prioriteit van de stroomkast te bepalen. Hierbij is het belangrijk dat een probleem met één component niet automatisch de hoogste prioriteit aan de gehele stroomkast geeft. In plaats daarvan wordt de prioriteit van de stroomkast berekend op basis van de bijdragen van alle componenten.

Voor elk component binnen de stroomkast wordt een prioriteit berekend, afhankelijk van de status van de zekering, afwijkingen in spanning, stroom, actief vermogen, en frequentie. Wanneer de zekering van een component uitstaat (dus flag “down” heeft), krijgt dit component automatisch een basisprioriteit van 1.0:

$$P_{component} = 1.0 \text{ als } status = false$$

Dit weerspiegelt de kritieke aard van een uitgevallen zekering. Als de zekering actief is (dus geen flag “down” heeft), wordt de prioriteit van het component berekend op basis van de afwijkingen in spanning, stroom, actief vermogen, en frequentie. De afwijkingen worden genormaliseerd door het verschil met de normale waarde te delen door de normale waarde:

$$\Delta x = \frac{|x_{gemeten} - x_{normaal}|}{x_{normaal}}$$

Om grotere afwijkingen te benadrukken, maken we gebruik van exponentiele schaling van de afwijking. Dit kunnen we op de volgende manier toepassen:

$$Scaled \Delta x = (e^{\Delta x} - 1) \times 2$$

Deze afwijkingen worden gecombineerd met een gewicht dat de belangrijkheid van het component binnen de stroomkast weergeeft. De prioriteit van een actief component wordt berekend als: (*Applicatiecode – Backend/Helpers/PriorityHelper.cs*)

$$P_{component} = w_V \cdot \Delta V + w_I \cdot \Delta I + w_p \cdot \Delta p + w_{freq} \cdot \Delta freq$$

Na het berekenen van de prioriteiten van de individuele componenten wordt de prioriteit van de gehele stroomkast bepaald. Hierbij, kennen we elk component een gewicht toe, dat de belangrijkheid van het component binnen de stroomkast weergeeft. Zo kunnen we kritieke componenten (modem) een hoger gewicht geven. Dit kan worden gedaan door een gewogen gemiddelde te berekenen, zodat de prioriteit van de stroomkast representatief is voor alle componenten:

$$P_{status} = \frac{\sum_{i=1}^k P_{component,i} \cdot w_{component,i}}{\sum_{i=1}^k w_{component,i}}$$

Een alternatieve aanpak is het gebruik van de maximale prioriteit van de componenten:

$$P_{status} = \max(P_{component,1}, P_{component,2}, \dots, P_{component,k})$$

De keuze tussen het gewogen gemiddelde en de maximale prioriteit hangt af van de toepassing. Het gewogen gemiddelde biedt een evenwichtigere weergave van de stroomkast-status, terwijl de maximale prioriteit de meest kritieke problemen snel aan het licht brengt.

Tot slot wordt de totale prioriteit van de stroomkast berekend door de netwerkprioriteit en de statusprioriteit van de stroomkast te combineren met een gewogen som:

$$P_{totaal} = w_{netwerk} \cdot P_{netwerk} + w_{status} \cdot P_{status}$$

Met deze aanpak wordt een gebalanceerde en flexibele methode geboden om prioriteiten voor stroomkasten te berekenen, waarbij rekening wordt gehouden met de status van individuele componenten, afwijkingen in spanning en stroom, en de duur van het probleem. Dit maakt het mogelijk om snel en gericht in te grijpen bij kritieke situaties zonder onnodige meldingen te genereren.

# Applicatie

Deze applicatie is een krachtig en gebruiksvriendelijk platform ontwikkeld om de status en het beheer van ANPR-camera's, stroomkasten en bijbehorende meldingen te monitoren en te beheren. Hieronder vindt u een korte beschrijving van de belangrijkste onderdelen van de applicatie.

## Frontend

De frontend van de applicatie is ontwikkeld met het Angular-framework van Google, wat zorgt voor een snelle en responsieve gebruikerservaring binnen de webbrowser. De interface biedt verschillende pagina's die inzicht geven in de status van de ANPR-camera's, stroomkasten en alerts. Voor de kaartweergave wordt gebruikgemaakt van Leaflet, een open-source JavaScript-bibliotheek waarmee interactieve kaarten kunnen worden weergegeven en bewerkt.

## Login

Wanneer de applicatie wordt geopend, wordt de gebruiker eerst naar de loginpagina geleid. Hier is het noodzakelijk om een geldige gebruikersnaam en wachtwoord in te voeren om toegang te verkrijgen. De beschikbaarheid van functies binnen de applicatie is afhankelijk van de rol en rechten van de gebruiker. De authenticatie wordt verzorgd door **Auth0**, wat zorgt voor een veilige en efficiënte inlogervaring. (*Applicatiecode – Frontend/src/app/pages/signin*)

## Auth0

Voor de login op ons dashboard gebruiken we de [Auth0 SDK for Angular Single Page Applications](#) die door Auth0 ontwikkeld wordt. Deze library kan worden gebruikt via een AuthService zoals:

### Login

```
import { AuthService } from '@auth0/auth0-angular';

constructor(private auth: AuthService) { }

handleLogin(): void {
  this.auth.loginWithRedirect({
    appState: {
      target: '/',
    },
    authorizationParams: {
      prompt: 'login',
    },
  });
}
```

## Rollen

Volgende code laat zien hoe je de rollen kan ophalen van een gebruiker.

```
return authService.getAccessTokenSilently({ authorizationParams: { prompt: 'none' }
}).pipe(
  map((token) => {
    const decodedToken = jose.decodeJwt(token) as { [key: string]: any } | null;
    const rolesClaimKey = environment.rolesClaimKey;
    const roles: string[] = decodedToken?.[rolesClaimKey] || [];

    return roles.includes(role);
  }),
);
```

Al de dingen die te maken heeft met user management wordt dus door Auth0 voorzien zoals uitloggen, check of gebruiker is ingelogd, permissies, .... Zo zorgen we ervoor dat pagina's anders getoond worden op basis van de rechten van de gebruiker.

## Dashboard

Na een succesvolle login komt de gebruiker op het dashboard. Dit vormt het startpunt van de applicatie en biedt een overzichtskaart met alle ANPR-palen. De palen worden weergegeven als markers op de kaart, waarbij de kleur van de markers de status van de paal aangeeft. De ernst van eventuele meldingen bepaalt de kleurcodering van de markers, zodat kritieke meldingen direct opvallen. Dit helpt gebruikers snel de status van de ANPR-palen te beoordelen. Ook is er de mogelijkheid om te filteren op adres of klant. (*Applicatiecode – Frontend/src/app/pages/dashboard*)

## Meldingen

De meldingenpagina biedt een overzicht van alle openstaande meldingen, inclusief de prioriteit, melding en tijdstip. Dit maakt het mogelijk om snel een duidelijk overzicht te krijgen van alle meldingen en de voortgang van het oplossen van deze meldingen.

Meldingen zijn gesorteerd op prioriteit en tijd, waarbij de oudste meldingen met de hoogste prioriteit bovenaan worden weergegeven. Dit maakt het voor service-engineers eenvoudig om te zien welke meldingen als eerste moeten worden behandeld.

Daarnaast is er de mogelijkheid om meldingen te filteren. (*Applicatiecode – Frontend/src/app/pages/alert-list*)

De detailpagina van een melding toont extra informatie, inclusief een veld voor opmerkingen. Dit biedt de mogelijkheid om bij te houden welke stappen al zijn ondernomen om een melding op te lossen. (*Applicatiecode – Frontend/src/app/pages/alert-detail*)

## Kasten

De Kastenpagina biedt een overzicht van alle stroomkasten in het systeem en toont de status van de componenten in deze kasten. Hierdoor kunnen gebruikers in één oogopslag zien welke componenten nog functioneren en hoeveel zekeringen er nog actief zijn op een bepaalde locatie. (*Applicatiecode – Frontend/src/app/pages/cabinet-list*)

Door op een stroomkast te klikken, wordt de gebruiker doorverwezen naar een gedetailleerde pagina die een visualisatie toont van de kast. Deze visualisatie geeft de status van elk component weer, zodat gebruikers gemakkelijk kunnen zien welke onderdelen actie vereisen.

In de detailweergave van de stroomkast is het mogelijk om op afstand een **modem-reset** uit te voeren. Bovendien kunnen gebruikers foto's opvragen van de binnenkant van de stroomkast voor visuele inspectie, wat helpt bij het diagnosticeren van eventuele problemen. (*Applicatiecode – Frontend/src/app/pages/cabinet-details*)

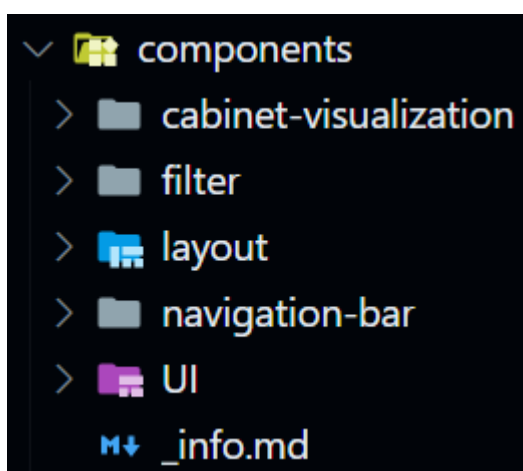
## Communicatie Backend

Om te communiceren met de backend maken we gebruik van services. Elk van deze services stelt een connectie op met de API, waaruit deze data ophaalt. Deze services zijn beschikbaar vanuit de hele applicatie. (*Applicatiecode – Frontend/src/app/services*)

Om deze interactie correct te laten verlopen, gebruikt de frontend dezelfde models als de backend. (*Applicatiecode – Frontend/src/app/models*)

## Components

Zoals gebruikelijk is in Angular, maakt onze applicatie gebruik van componenten. Componenten die op pagina's gebruikt worden zijn gegroepeerd op basis van hoe deze gebruikt worden. (*Applicatiecode – Frontend/src/app/components*)





## Paginas

Elke pagina is gecreëerd als een aparte component. Deze bevinden zich in een afzonderlijke folder. (*Applicatiecode – Frontend/src/app/pages*)

## Ruimte voor Aanpassingen

De applicatie is gemaakt met de gelaten ruimte voor aanpassingen. Tijdens de demomeetings werd er reeds aangegeven dat de effectieve data die getoond wordt eenvoudig kan worden aangepast naar de toekomst toe.

## Backend

De backend is ontwikkeld met het ASP.NET Core Framework en bevat de volledige businesslogica die nodig is om de data op een correcte en gestructureerde manier beschikbaar te stellen aan de frontend. Daarnaast is de backend beveiligd met Auth0, waardoor ongeautoriseerde acties worden voorkomen en alleen geverifieerde gebruikers toegang krijgen tot de applicatie.

In dit hoofdstuk wordt een beknopt overzicht gegeven van de belangrijkste componenten en functionaliteiten binnen de backend.

### Auth0 beveiliging

Voor de autorisatie van de backend maken wij gebruik van Auth0. Auth0 biedt een geavanceerd user management systeem, waarin alle gebruikers beheerd kunnen worden. De gebruikersgegevens worden veilig opgeslagen in de database van Auth0.

Voor de authenticatie en autorisatie maakt Auth0 gebruik van Bearer tokens, die worden meegegeven bij elke API-aanroep om de identiteit en rechten van de gebruiker te verifiëren.

Om de integratie met Auth0 correct te configureren, hebben we in de backend de volgende code toegevoegd in (*Applicationcode – Backend/program.cs*):

```
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.Authority = "https://team-said.eu.auth0.com/";
    options.Audience = "https://localhost:7280";
});

builder.Services.AddAuthorization();
app.UseAuthorization();
```

De bovenstaande code is essentieel om de authenticatie correct te configureren en de backend te koppelen aan de juiste Auth0 API's.

Daarnaast zorgen de onderste regels code ervoor dat de autorisatie correct functioneert. Hierdoor kunnen we later in de endpoints autorisatie afdwingen door gebruik te maken van de [Authorize]-attribuut boven de controllers of specifieke methodes.

```
app.UseCors(options =>
{
    options.AllowAnyHeader();
    options.AllowAnyMethod();
    options.WithOrigins("http://localhost:4200", "https://said.bestandje.zip",
"https://api.bestandje.zip", "http://dietpi");
});
```

Deze code is nodig om CORS (Cross-Origin Resource Sharing) correct in te stellen. CORS bepaalt welke URL's toegang krijgen tot de backend.

Door deze configuratie toe te passen, wordt alleen verkeer vanaf de gespecificeerde URL's toegestaan. Pogingen om de backend te benaderen vanaf andere domeinen worden automatisch geblokkeerd, wat bijdraagt aan de beveiliging van de applicatie.

## Helpers

Binnen de map Helpers (Applicationcode → Backend → Helpers) bevinden zich twee hulpprogramma's die specifieke berekeningen en dataverwerking uitvoeren:

### 1. PrioriteitHelper

- Deze helper wordt gebruikt voor de berekening van de prioriteit van events.
- Een gedetailleerde uitleg van de prioriteitsberekeningen is eerder in dit document beschreven.

### 2. ZabbixHelper

- Deze helper ondersteunt de verwerking van Zabbix-events die worden opgehaald uit de CIPAL API.
- De functionaliteiten van deze helper omvatten:
  - Het filteren en aanpassen van bepaalde velden.
  - Het opsplitsen van specifieke gegevens om ze bruikbaar te maken binnen onze applicatie.
  - Het uitvoeren van kleinere prioriteitsberekeningen.
- Een gedetailleerde uitleg van deze berekeningen en de bijbehorende code is te vinden in de notebook:  
(*Applicatiecode – AI/Data\_App.ipynb*)

Deze helpers zorgen ervoor dat de backend efficiënt en gestructureerd de juiste gegevens kan verwerken en prioriteiten correct kan toewijzen.

## Externe Services

Binnen de backend maken we gebruik van vier services, die elk een specifieke externe functionaliteit ondersteunen:

### 1. Swagger Extensions Service

Deze service voegt een Bearer Token-authenticatieknop toe aan Swagger, zodat gebruikers zich tijdens de ontwikkelfase kunnen authenticeren en geautoriseerde API-aanroepen kunnen uitvoeren.

### 2. Pi API Service

Deze service is verantwoordelijk voor het versturen van requests naar de Raspberry Pi. Ze wordt gebruikt om:

- Een foto op te vragen van de Pi.
- Een modemreset aan te vragen.

### 3. Ciral API Service

Deze service faciliteert de communicatie met de Ciral API, die werkt op basis van workbenches en layers.

- We beschikken over een workbench-account van Ciral, waarmee we kunnen inloggen op hun API.
- De layers binnen de workbench vertegenwoordigen verschillende tabellen in het datamodel van Ciral.
- We maken gebruik van drie specifieke endpoints:
  1. Ophalen van alle fields.
  2. Ophalen van een specifiek field op basis van een ID.
  3. Updaten van comments.
- In de controllers kunnen we eenvoudig data ophalen door de juiste layerId mee te geven.

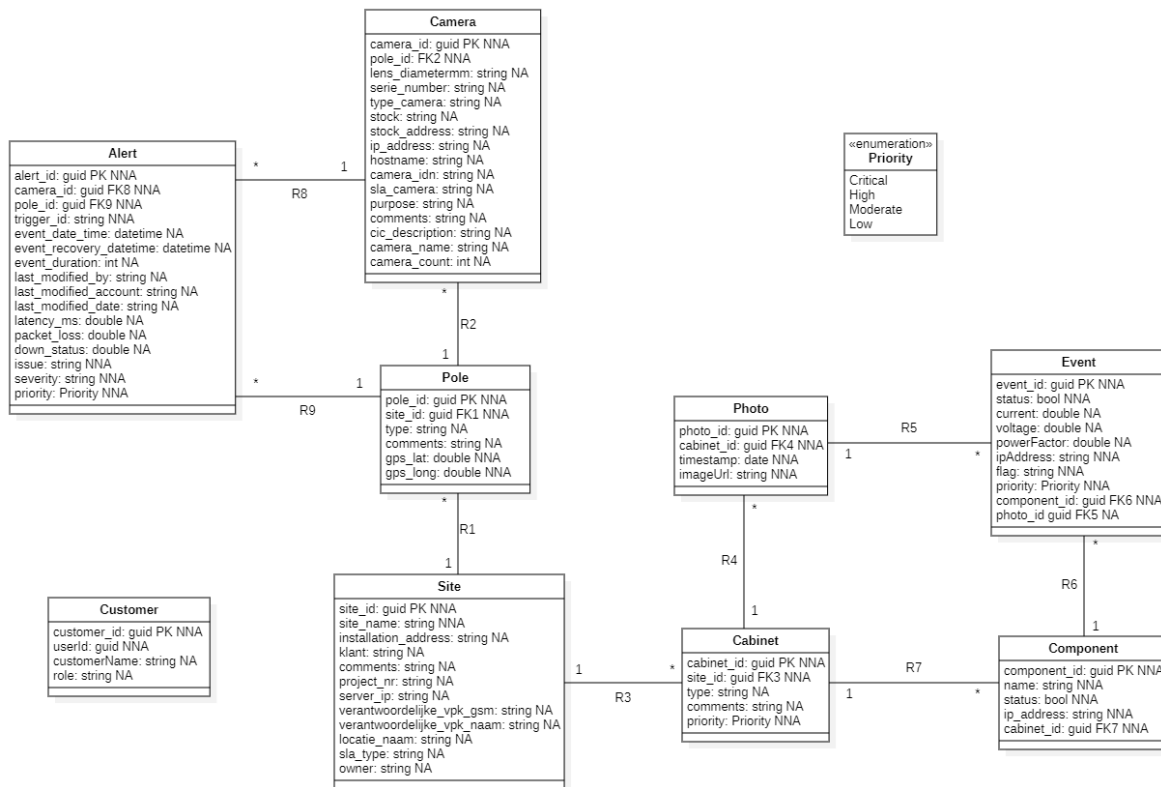
### 4. Auth0 Service

Deze service wordt gebruikt voor gebruikersbeheer en de synchronisatie met onze eigen database:

- Gebruikersgegevens ophalen via Auth0.
- Onze database vullen met klanten, waarbij we per klant de toegankelijke camera's, palen en kasten beperken.
- Ophalen van de userId en rollen van gebruikers via de Bearer Token van Auth0.
- De Bearer Token kan in deze service worden opgehaald en gebruikt voor verdere authenticatie.

Door deze services te integreren, kunnen we externe API's efficiënt benaderen en de autorisatie en dataflow binnen onze applicatie optimaliseren.

## Data Model



Ons datamodel bevat een overzicht van alle tabellen die binnen de applicatie worden gebruikt. Niet alle tabellen worden echter lokaal opgeslagen in onze database; sommige gegevens worden direct opgehaald uit de Ciral API via de services.

De tabellen die wel in onze eigen database worden opgeslagen:

- Cabinet
- Foto
- Event
- Component
- Customer
- Site (gedeeltelijk)

De overige tabellen worden niet lokaal bewaard, maar rechtstreeks opgehaald uit de Ciral API met behulp van de verschillende services. Dit zorgt ervoor dat we enkel de noodzakelijke gegevens opslaan en zoveel mogelijk up-to-date data direct uit de API kunnen verwerken.

## Models

De Models vertegenwoordigen de database-entiteiten en bevatten alle tabellen die in onze database worden opgeslagen. Binnen deze modellen worden de attributen gedefinieerd met de juiste datatypes, zodat de structuur van de database correct wordt nagebootst.

In sommige bestanden bevinden zich twee klassen/modellen. Dit is nodig om de data van de Cipal API correct om te zetten naar onze eigen database-modellen binnen de database-initialisatie.

De bijbehorende code is te vinden in: (*Applicationcode – Backend – Models*)

Deze models vormen de basis voor de gegevensstructuur en zorgen ervoor dat data correct kan worden beheerd en verwerkt binnen de applicatie.

## DTO's (Data Transfer Objects)

De DTO's (Data Transfer Objects) worden gebruikt om modellen om te vormen naar een gestructureerde request of response. Dit zorgt ervoor dat we enkel de relevante data verzenden en ontvangen binnen onze API-endpoints.

We gebruiken DTO's voornamelijk in de endpoints, omdat:

- Niet altijd alle data moet worden teruggegeven of opgevraagd.
- Voor een request bijvoorbeeld geen ID hoeft te worden meegegeven bij het aanmaken van een nieuw record.
- Sommige velden die niet relevant zijn voor een specifieke bewerking, worden weggelaten.

De DTO's zijn terug te vinden in: (*Applicationcode – Backend – DTOs*)

Door het gebruik van DTO's zorgen we voor een efficiënte en veilige data-uitwisseling tussen de frontend en de backend.

## Gebruik van AutoMapper

Voor het efficiënt converteren van modellen naar DTO's en vice versa maken wij gebruik van de AutoMapper-package. AutoMapper vereenvoudigt deze mapping, zodat we niet handmatig elk veld hoeven toe te wijzen. Door gebruik te maken van AutoMapper in onze code, kunnen we deze mappings eenvoudig toepassen zonder repetitieve code.

Om dit te realiseren, hebben we Mapping Profielen aangemaakt. Deze profielen zorgen ervoor dat:

- Modellen automatisch worden omgezet naar de bijbehorende DTO's.
- DTO's automatisch worden omgezet naar modellen.
- De code leesbaar en onderhoudbaar blijft, zonder handmatige toewijzingen van elk veld.

## Data Access Layer

De Data Access Layer (DAL) bevat alle logica die nodig is voor de interactie met de database. Om de code overzichtelijk te houden, is deze map opgesplitst in verschillende submappen.

### **(Applicationcode – Backend – DAL – Data)**

Deze map bevat de kerncomponenten voor het opbouwen en initialiseren van de database:

- Context: Bevat de databasecontext waarin de modellen worden gedefinieerd.
- Database Initializer: Zorgt voor het vullen van de database met data uit de Cipal API. Dit gebeurt via de Cipal API Service.

### **(Applicationcode – Backend – DAL – Repositories)**

Deze map bevat de repository-patronen voor gestructureerde data-opvragingen:

- IRepository.cs (interface): Definieert de basisfunctionaliteiten voor alle repositories.
- GenericRepository.cs: Bevat algemene functies die elke repository kan gebruiken, zoals CRUD-operaties.
- Voor elk specifiek model is er een bijhorende repository, waarin model-specifieke functies kunnen worden geïmplementeerd.
- De repositories zorgen ervoor dat data efficiënt wordt opgehaald via de databasecontext.

### **(Applicationcode – Backend – DAL – UnitOfWork)**

Deze map bevat een interface en een klasse die alle repositories initialiseren en beheren:

- UnitOfWork maakt het mogelijk om repositories eenvoudig op te roepen zonder direct gebruik te maken van de databasecontext.
- Dit biedt als voordeel dat meerdere repositories binnen één bewerking kunnen worden gebruikt zonder aparte database-aanroepen.

Door deze structuur zorgen we voor een duidelijke scheiding tussen de database-logica en de rest van de applicatie, wat onderhoud en uitbreidbaarheid vergemakkelijkt.

## Controllers

In deze map bevinden zich alle controllers, waarin de endpoints worden gedefinieerd. Om de code overzichtelijk en gestructureerd te houden, is er per tabel een aparte controller aangemaakt.

De naamgeving van de controllers volgt een vast patroon:

"Tabelnaam (meervoud) + Controller"

Bijvoorbeeld: AlertsController voor de Alert-tabel.

### Authenticatie en Autorisatie (Auth0)

- Globale autorisatie: Op de controllers wordt [Authorize] toegepast, zodat alleen geauthenticeerde gebruikers toegang krijgen.
- Specifieke autorisatie per request:
  - Met behulp van de Auth0Service worden de userId en rollen opgehaald.
  - Op basis van deze rollen wordt bepaald of de gebruiker toegang krijgt tot specifieke gegevens.
  - Rechtenverdeling:
    - Klant: Heeft enkel toegang tot zijn eigen palen.
    - Service Desk Medewerker: Kan alle palen inzien.

Door deze autorisatiestructuur zorgen we ervoor dat alleen bevoegde gebruikers toegang hebben tot de juiste API-functionaliteiten.

## Database en Docker-Setup

Voor de opslag van onze data maken wij gebruik van een PostgreSQL-database.

### Lokale omgeving

- Lokaal wordt de PostgreSQL-database opgezet met een Docker Compose-file.
- Dit zorgt ervoor dat de database eenvoudig kan worden gestart en beheerd binnen een containeromgeving.

### Productieomgeving

- In de productieomgeving wordt de database geconfigureerd via de Dockerfile.
- Dit automatiseert de database-initialisatie en het beheer binnen de containerized omgeving.

### Databasegebruik

- De database wordt gevuld via de database-initializer, die data ophaalt uit de Ciral API.
- Tijdens de werking van de applicatie wordt de database verder aangevuld met:
  - Events die binnenkomen via de applicatie.
  - Foto's die gegenereerd worden door onze Raspberry Pi.

Zo zorgen we voor een stabiele en schaalbare oplossing, zowel lokaal als in productie.



## Bronnen

Amazon. (n.d.). *Getting started with AWS IoT Core tutorials*. Retrieved from AWS:  
<https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>

ChatGPT. (2025). *Herschreven tekst met behulp van AI*. Retrieved from  
<https://chatgpt.com/>

Tailscale. (n.d.). *homepage*. Retrieved from tailscale: <https://tailscale.com/>