



Dwaaldetectie Documentatie

3^{de} fase IT Factory

Academiejaar 2021-2022

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Cornelissen Kieran

INHOUDSTAFEL

INHOUDSTAFEL	2
1 INLEIDING	4
2 PROJECT SCOPE	5
3 UI.....	6
3.1 Interface	7
3.2 Ondersteunende Functies van de Interface.....	15
3.2.1 Functie 'indicate_change'	15
3.2.2 Functie 'open_directory_dialog'	16
3.2.3 Functie 'open_file_dialog'	17
3.3 Opslaan en Validatie	18
3.3.1 IQ Messenger inlog gegevens	26
3.4 Configuratie Test.....	26
3.4.1 Functie: 'test_config'	26
3.4.2 Functie: 'handle_api_response'	27
4 DWAALDETECTIE.....	29
4.1 Logging	30
4.1.1 Logging configuratie	30
4.1.2 Toevoegen aan de log.....	31
4.1.3 Event loggen	31
4.1.3.1 Loggen met een begeleider	31
4.1.3.2 Loggen zonder begeleider	33
4.1.4 Error loggen	34

4.2	Programma Initialisatie	34
4.3	Hoofdkloop	36
4.4	Definitie 'get_data'	37
4.5	Definitie 'event_service'	38
4.6	Definitie 'get_event'	39
4.7	Definitie 'handle_response'	43
4.8	Definitie 'check_doors'	46
4.9	Definitie 'get_groups'	47
4.10	Definitie 'check_begeleider'	48
4.11	Definitie 'process_event'	50
4.12	Definitie 'validate_event'	53
4.13	Definitie 'event_at_same_time'	53
4.14	Definitie 'call_alarm'	54
4.15	Definitie 'process_done_list'	55
4.16	Definitie 'calculate_time'	55
4.17	Definitie 'clear_checked_list'	56
4.18	Definitie 'remove_threads'	57
5	AFBEELDING EVENT OPHALEN	58
5.1	Definitie 'create_image'	59
5.2	Definitie 'get_data'	62
5.3	Definitie 'load_image_from_response'	63
5.4	Definitie 'load_logo'	63
5.5	Definitie 'combine_images'	64
5.6	Definitie 'add_logo'	66
5.7	Definitie 'add_time_stamp'	67
6	API.....	70
6.1	Initialisatie	71
6.2	Autorisatie	72
6.3	'get_image' API	72
6.4	'reset_dwaaldetectie' API	74
7	AFBEELDING RESETTEN	76
7.1	Definitie 'replace_and_modify_image'	77
7.2	Definitie 'add_text_to_image'	80
8	IQ MESSENGER	83
8.1	Devices	84
8.2	Event flow	86
8.2.1	Volledige Flow	86
8.2.2	Event input	86
8.2.3	Condition	87
8.2.4	Alarm aansturen	87
8.2.5	Response	89
8.2.6	Afbeelding resetten	90

1 INLEIDING

Dit document is het realisatiedocument voor mijn stageopdracht bij Delta Technics. Delta Technics is gespecialiseerd in bekabeling en verschillende soorten datacommunicatie en heeft Delta Care opgericht om technologie in de zorgwereld te introduceren en de druk op het personeel te verlagen.

Voor mijn opdracht moest ik een systeem ontwikkelen dat ervoor zorgt dat wanneer een bewoner met dementie alleen naar buiten wil, er een alarm afgaat, maar niet wanneer er een begeleider bij is. Dit is cruciaal omdat het voor bewoners met dementie gevaarlijk is om alleen buiten te komen, terwijl ze wel de mogelijkheid moeten hebben om onder begeleiding naar buiten te gaan. De huidige oplossing zorgt ervoor dat er altijd een alarm afgaat, zelfs als er een begeleider bij is, wat leidt tot veel onnodige alarmen.

Mijn programma maakt gebruik van SAFR, een applicatie die eenvoudig verbinding maakt met een camera en gezichtsherkenning toepast op de beelden. In SAFR kunnen we personen toevoegen zodat ze herkend worden door de gezichtsherkenning. Ik houd bij wie een bewoner is en wie een begeleider is, en welke deuren een bewoner wel of niet mag gebruiken. SAFR maakt gebruik van API's die ik kan aanroepen om de detectie op te vragen.

Het systeem koppelt met IQ Messenger, dat meldingen en alarmen beheert, om ervoor te zorgen dat het alarm op het juiste moment wordt geactiveerd. Door deze integratie kunnen we de veiligheid van bewoners met dementie waarborgen door ongeautoriseerd verlaten van het gebouw te detecteren en alarmsignalen uit te zenden wanneer nodig.

2 PROJECT SCOPE

Het doel van dit project is om een applicatie te maken die ervoor gaat zorgen dat er enkel een alarm af zal gaan wanneer er een bewoner met dementie alleen naar buiten probeert te gaan door een deur waar deze persoon niet alleen buiten mag. De applicatie maakt gebruik van SAFR waar de gezichtsherkenning op zal gebeuren. En IQ Messenger waar het alarm mee geregeld zal worden. De applicatie zal volgende functionaliteiten bieden:

- Het verwerken van de geleverde herkenningen aan de hand van de SAFR API.
- Het nakijken of de bewoner met dementie alleen probeert naar buiten te gaan of dat er een begeleider bij is.
- Het activeren van een alarm als de bewoner met dementie alleen probeert naar buiten te gaan door een deur die hij niet alleen naar buiten mag nemen.

De stakeholders van dit project zijn:

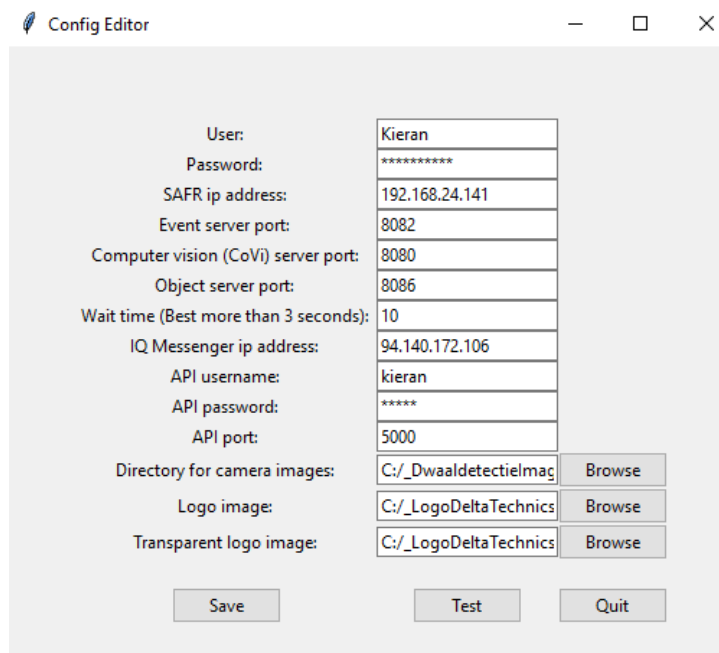
- De opdrachtgever: Delta Technics, zij hebben de vorige applicatie geplaatst en zouden graag een verbeterde versie hebben.
- De eindgebruikers: Medewerkers van het rusthuis, zij zullen minder alarmen krijgen waarbij er eigenlijk niets moet gebeuren.

3 UI

In de gebruikersinterface van dit systeem kunnen gebruikers essentiële configuratie-instellingen aanpassen die nodig zijn voor een optimale werking. Dit omvat het specificeren van de poortnummers voor de SAFR API, waaronder de Event server poort, de Computer Vision (CoVi) poort, en de Object server poort. Gebruikers kunnen ook de inloggegevens voor een SAFR-account invoeren, waarvoor minimaal monitorrechten vereist zijn.

Verder stelt de interface gebruikers in staat om IP-adressen in te voeren voor zowel de SAFR- als de IQ Messenger-servers. Er is ook een optie om de reactietijd te configureren, dat wil zeggen de tijdspanne waarbinnen een begeleider gedetecteerd moet worden na een gebeurtenis.

Wat betreft de beveiliging van onze eigen API, kunnen gebruikers hier een



The screenshot shows a 'Config Editor' window with the following fields and values:

User:	Kieran	
Password:	*****	
SAFR ip address:	192.168.24.141	
Event server port:	8082	
Computer vision (CoVi) server port:	8080	
Object server port:	8086	
Wait time (Best more than 3 seconds):	10	
IQ Messenger ip address:	94.140.172.106	
API username:	kieran	
API password:	*****	
API port:	5000	
Directory for camera images:	C:/_Dwaaldetectielmag	Browse
Logo image:	C:/_LogoDeltaTechnics	Browse
Transparent logo image:	C:/_LogoDeltaTechnics	Browse

At the bottom of the window are three buttons: 'Save', 'Test', and 'Quit'.

gebruikersnaam en wachtwoord instellen, en een specifieke poort toewijzen die gebruikt wordt door de API-server. Dit garandeert dat de communicatie tussen onze servers veilig verloopt.

Gebruikers hebben tevens de mogelijkheid om locaties te definiëren voor het opslaan van afbeeldingen van gebeurtenissen en voor het plaatsen van het DeltaTechnics logo, inclusief een versie met een transparante achtergrond. Dit kan gemakkelijk gedaan worden met een 'browse'-knop, waarmee mappen of bestanden geselecteerd kunnen worden.

Tot slot biedt de interface de mogelijkheid om de ingevoerde IP-adressen en poortnummers te testen om de juistheid en functionaliteit te verifiëren. Hoewel gebruikers deze configuraties ook handmatig in het config.json-bestand kunnen invoeren, biedt de interface het voordeel van validatie om te verzekeren dat alle ingevoerde gegevens correct zijn.

3.1 Interface

In de gebruikersinterface van dit programma maak ik gebruik van de Tkinter library, die het eenvoudig maakt om een grafische applicatie te ontwikkelen. De basis van de interface is een frame, gecreëerd met een padding van 50, wat zorgt voor een visuele marge rond de elementen, zodat deze niet direct tegen de rand van het venster staan. Ik gebruik het grid-systeem van Tkinter om de widgets gestructureerd en met gelijke tussenruimtes te plaatsen. Bij het initiëren van de interface laadt het programma eerst bestaande configuratiegegevens uit een configuratiebestand. Vervolgens worden op basis van deze gegevens de nodige widgets aangemaakt voor elk veld in het configuratiebestand.

```
def __init__(self, root):
    self.frm = ttk.Frame(root, padding=50)
    self.frm.grid()
    self.load_config()
    self.create_widgets()
```

In deze functie open ik het configuratiebestand 'config.json' en laad alle informatie op in de variabele 'config'. Dit gebeurt binnen een 'try'-blok om veilig om te gaan met mogelijke uitzonderingen. Mocht het bestand om welke reden dan ook niet kunnen worden geopend of gelezen, dan vangt de 'except'-clausule de fout op. In dat geval verschijnt er een foutmelding op het scherm die de aard van het probleem uitlegt, waardoor de gebruiker direct feedback krijgt over wat er mis is gegaan.

```
def load_config(self):
    try:
        with open('./config.json', 'r') as f:
            self.config = json.load(f)
    except Exception as e:
        messagebox.showerror("File Error", f"Failed to load configuration: {e}")
```

De functie 'create_widgets' is verantwoordelijk voor het aanmaken van de gebruikersinterface-elementen (widgets) binnen het frame dat eerder is gecreëerd. Elk widget wordt zorgvuldig gepositioneerd binnen een grid-layout, wat helpt bij het organiseren van de interface op een gestructureerde en visueel aantrekkelijke manier. Er is geen aanduiding of een veld ingevuld moet zijn of niet, maar hier moet alles ingevuld zijn.

1. Gebruikersnaam (User) Widget:
 - **Label:** Een label met de tekst 'User:' wordt gecreëerd en geplaatst in de eerste kolom (column 0) en de eerste rij (row 0) van het grid. Dit label dient als een aanduiding voor de gebruikersnaam invoerveld.
 - **Invoerveld:** Een invoerveld voor de gebruikersnaam wordt direct naast het label geplaatst, in de tweede kolom (column 1) van dezelfde rij. Dit veld wordt vooraf ingevuld met de gebruikersnaam vanuit het configuratiebestand ('config.json'). Om interactie te faciliteren, wordt een event 'KeyRelease' gekoppeld aan dit veld, waardoor elke toetsaanslag die de inhoud van het veld verandert, wordt gedetecteerd en verwerkt door de functie

'`indicate_change`', die het label van dit veld visueel markeert om veranderingen aan te geven.

User:

Het veld 'User' is de username van het SAFR account. Dit veld mag niet leeg zijn.

```
def create_widgets(self):
    self.user_label = ttk.Label(self.frm, text="User: ")
    self.user_label.grid(column=0, row=0)
    self.user = ttk.Entry(self.frm)
    self.user.insert(0, self.config["user"]["userId"])
    self.user.grid(column=1, row=0)
    self.user.bind("<KeyRelease>", lambda e: self.indicate_change(self.user_label))
```

2. Wachtwoord (Password) Widget:

- **Label:** Een label met de tekst 'Password:' wordt geplaatst in de eerste kolom van de tweede rij (row 1). Dit label identificeert het wachtwoord invoerveld voor de gebruiker.
- **Invoerveld:** Naast het wachtwoord label, in de tweede kolom van de rij, wordt een wachtwoordveld gecreëerd waarbij de ingevoerde tekens verborgen worden weergegeven (getoond als sterretjes '*'). Het wachtwoordveld wordt vooraf gevuld met het wachtwoord uit het configuratiebestand. Net als bij het gebruikersnaamveld, wordt er een 'KeyRelease' event aan gekoppeld, zodat elke wijziging in het veld leidt tot een visuele indicatie via de '`indicate_change`' functie.

Password:

Het veld 'Password' is het wachtwoord van het SAFR account. Dit veld mag niet leeg zijn.

```
self.password_label = ttk.Label(self.frm, text="Password: ")
self.password_label.grid(column=0, row=1)
self.password = ttk.Entry(self.frm, show="*")
self.password.insert(0, self.config["user"]["password"])
self.password.grid(column=1, row=1)
self.password.bind("<KeyRelease>", lambda e: self.indicate_change(self.password_label))
```

3. SAFR IP-adres Widget:

- **Label:** Een label met de tekst 'SAFR ip address:' wordt gecreëerd en geplaatst in de eerste kolom (column 0) en de derde rij (row 2) van het grid. Dit label dient als indicatie voor het invoerveld van het SAFR IP-adres.
- **Invoerveld:** Direct naast het label, in de tweede kolom (column 1) van dezelfde rij, wordt een invoerveld voor het SAFR IP-adres geplaatst. Dit veld wordt vooraf gevuld met de SAFR IP-adres waarde uit het configuratiebestand ('`config.json`'). Evenzo wordt een 'KeyRelease' event aan dit veld gekoppeld, waardoor veranderingen in de inhoud van het veld worden gedetecteerd. Wanneer er een toets wordt losgelaten, wordt de functie '`indicate_change`' aangeroepen, die het label van dit veld visueel markeert om aan te geven dat er een wijziging heeft plaatsgevonden.

SAFR ip address:

Hier moet het IP-adres van de SAFR server ingevuld worden. Er wordt nagekeken of dit de vorm heeft van een IP-adres.

```
self.ip_address_label = ttk.Label(self.frm, text="SAFR ip address: ")
self.ip_address_label.grid(column=0, row=2)
self.ip_address = ttk.Entry(self.frm)
self.ip_address.insert(0, self.config["settings"]["ipAddress"])
self.ip_address.grid(column=1, row=2)
self.ip_address.bind("<KeyRelease>", lambda e: self.indicate_change(self.ip_address_label))
```

4. Event Server-poort widget:

- **Label:** Er wordt een label met de tekst 'Event server port:' gemaakt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de vierde rij (row 3) van het grid. Het markeert duidelijk waar gebruikers het nummer van de event server-poort moeten invoeren.
- **Invoerveld:** Naast het label, in de tweede kolom (column 1) van dezelfde rij, wordt een invoerveld voor de event server-poort geplaatst. Dit veld wordt vooraf ingevuld met het nummer van de event server-poort uit het 'config.json' bestand, zoals opgegeven onder de instellingen. Net als bij andere velden, is dit invoerveld uitgerust met een 'KeyRelease' event, dat de functie 'indicate_change' activeert wanneer een toets wordt losgelaten. Deze functie licht het label op om aan te geven dat er een wijziging is aangebracht in de inhoud van het invoerveld.

Event server port: 8082 Hier moet de poort van de SAFR event server ingevuld worden. Deze kan je vinden door op de SAFR server de python file 'portcheck.py' op de locatie 'C:\Program Files\RealNetworks\SAFR\bin\' te runnen.

```
self.event_server_port_label = ttk.Label(self.frm, text="Event server port: ")
self.event_server_port_label.grid(column=0, row=3)
self.event_server_port = ttk.Entry(self.frm)
self.event_server_port.insert(0, self.config["settings"]["eventServerPort"])
self.event_server_port.grid(column=1, row=3)
self.event_server_port.bind("<KeyRelease>", lambda e: self.indicate_change(self.event_server_port_label))
```

5. Computer Vision (CoVi) Server-poort widget:

- **Label:** Er wordt een label met de tekst 'Computer vision (CoVi) server port:' geplaatst, wat aangeeft waar gebruikers het poortnummer van de Computer Vision server moeten invoeren. Dit label wordt in de eerste kolom (column 0) en de vijfde rij (row 4) van het grid geplaatst.
- **Invoerveld:** Naast het label bevindt zich in de tweede kolom (column 1) van dezelfde rij een invoerveld voor de Computer Vision server-poort. Dit veld wordt vooraf ingevuld met het poortnummer uit het 'config.json' bestand. Dit invoerveld is eveneens voorzien van een 'KeyRelease' event dat de functie 'indicate_change' activeert om het label te markeren wanneer er wijzigingen worden aangebracht.

Computer vision (CoVi) server port: 8080 Hier moet de poort van de SAFR computer vision (CoVi) server ingevuld worden. Deze kan je vinden door op de SAFR server de python file 'portcheck.py' op de locatie 'C:\Program Files\RealNetworks\SAFR\bin\' te runnen.

```
self.covi_server_port_label = ttk.Label(self.frm, text="Computer vision (CoVi) server port: ")
self.covi_server_port_label.grid(column=0, row=4)
self.covi_server_port = ttk.Entry(self.frm)
self.covi_server_port.insert(0, self.config["settings"]["computerVisionPort"])
self.covi_server_port.grid(column=1, row=4)
self.covi_server_port.bind("<KeyRelease>", lambda e: self.indicate_change(self.covi_server_port_label))
```

6. Object Server-poort widget:

- **Label:** Een label met de tekst 'Object server port:' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de zesde rij (row 5) van het grid.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor de Object server-poort geplaatst. Dit veld is vooraf ingevuld met het poortnummer uit het 'config.json' bestand, zoals opgegeven onder de instellingen. Dit veld is ook uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die het label oplicht bij wijzigingen.

Object server port: 8086

Hier moet de poort van de SAFR object server ingevuld worden. Deze kan je vinden door op de SAFR server de python file 'portcheck.py' op de locatie 'C:\Program Files\RealNetworks\SAFR\bin\' te runnen.

```
self.object_server_port_label = ttk.Label(self.frm, text="Object server port: ")
self.object_server_port_label.grid(column=0, row=5)
self.object_server_port = ttk.Entry(self.frm)
self.object_server_port.insert(0, self.config["settings"]["objectServerPort"])
self.object_server_port.grid(column=1, row=5)
self.object_server_port.bind("<KeyRelease>", lambda e: self.indicate_change(self.object_server_port_label))
```

7. Wachtijd Widget:

- **Label:** Een label met de tekst 'Wait time (Best more than 3 seconds):' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de zesde rij (row 6) van het grid.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor de wachttijd geplaatst. Dit veld is vooraf ingevuld met de wachttijd zoals gespecificeerd in het 'config.json' bestand. Dit veld is ook uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer wijzigingen zijn gemaakt.

Wait time (Best more than 3 seconds): 10

Hier moet je invullen hoeveel seconden dat je wil dat er tussen de begeleider en bewoner zitten. Als de wachttijd korter is dan 3 seconden dan kan het zijn dat er een aantal events worden overgeslagen.

```
self.wait_time_label = ttk.Label(self.frm, text="Wait time (Best more than 3 seconds): ")
self.wait_time_label.grid(column=0, row=6)
self.wait_time = ttk.Entry(self.frm)
self.wait_time.insert(0, self.config["waitTime"])
self.wait_time.grid(column=1, row=6)
self.wait_time.bind("<KeyRelease>", lambda e: self.indicate_change(self.wait_time_label))
```

8. IQ Messenger IP-adres Widget:

- **Label:** Een label met de tekst 'IQ Messenger IP-adres:' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de zevende rij (row 7) van het grid.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor het IQ Messenger IP-adres geplaatst. Dit veld is vooraf ingevuld met het IP-adres zoals gespecificeerd in het 'config.json' bestand onder de sectie 'iqMessenger'. Net zoals de andere velden is dit veld uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer wijzigingen zijn gemaakt.

IQ Messenger ip address: 94.140.172.106 Hier moet het IP-adres van de IQ Messenger server ingevuld worden. Er wordt nagekeken of dit de vorm heeft van een IP-adres.

```
self.iqmessenger_ip_label = ttk.Label(self.frm, text="IQ Messenger ip address: ")
self.iqmessenger_ip_label.grid(column=0, row=7)
self.iqmessenger_ip = ttk.Entry(self.frm)
self.iqmessenger_ip.insert(0, self.config["iqMessenger"]["ipAddress"])
self.iqmessenger_ip.grid(column=1, row=7)
self.iqmessenger_ip.bind("<KeyRelease>", lambda e: self.indicate_change(self.iqmessenger_ip_label))
```

9. API-gebruikersnaam Widget:

- **Label:** Een label met de tekst 'API gebruikersnaam:' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de achtste rij (row 8) van het grid. Hier kan je invullen wat je als gebruikersnaam wil om de API van dit programma te gebruiken.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor de API-gebruikersnaam geplaatst. Dit veld is vooraf ingevuld met de gebruikersnaam uit het 'config.json' bestand, zoals gespecificeerd in de sectie 'api'. Dit veld is ook uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer wijzigingen zijn gemaakt.

API username: kieran Hier moet je een gebruikersnaam voor onze API kiezen. Deze moet ook in IQ Messenger gebruikt worden om de afbeeldingen te laten zien en resetten. Dit veld mag niet leeg zijn.

```
self.api_user_label = ttk.Label(self.frm, text="API username: ")
self.api_user_label.grid(column=0, row=8)
self.api_user = ttk.Entry(self.frm)
self.api_user.insert(0, self.config["api"]["username"])
self.api_user.grid(column=1, row=8)
self.api_user.bind("<KeyRelease>", lambda e: self.indicate_change(self.api_user_label))
```

10. API-wachtwoord Widget:

- **Label:** Een label met de tekst 'API wachtwoord:' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de negende rij (row 9) van het grid. Hier kan je invullen wat je als wachtwoord wil om de API van dit programma te gebruiken.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor het API-wachtwoord geplaatst. Dit veld is vooraf ingevuld met het wachtwoord uit het 'config.json' bestand, zoals gespecificeerd in de sectie 'api'. Voor extra veiligheid wordt de inhoud van dit veld verborgen weergegeven ('*'). Ook dit veld is uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer wijzigingen zijn gemaakt.

API password: ***** Hier moet je een wachtwoord voor onze API kiezen. Deze moet ook in IQ Messenger gebruikt worden om de afbeeldingen te laten zien en resetten. Dit veld mag niet leeg zijn.

```
self.api_password_label = ttk.Label(self.frm, text="API password: ")
self.api_password_label.grid(column=0, row=9)
self.api_password = ttk.Entry(self.frm, show="*")
self.api_password.insert(0, self.config["api"]["password"])
self.api_password.grid(column=1, row=9)
self.api_password.bind("<KeyRelease>", lambda e: self.indicate_change(self.api_password_label))
```

11. API-poort Widget:

- **Label:** Een label met de tekst 'API poort:' wordt gebruikt om het bijbehorende invoerveld aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de tiende rij (row 10) van het grid. Hier kan je kiezen op welke poort van de server dat de API gaat draaien.
- **Invoerveld:** In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor de API-poort geplaatst. Dit veld is vooraf ingevuld met het poortnummer uit het 'config.json' bestand, gespecificeerd onder de 'api' sectie. Dit veld is uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer er wijzigingen zijn gemaakt.

API port:

Hier moet je een poort voor onze API kiezen. Deze moet ook in IQ Messenger gebruikt worden om de afbeeldingen te laten zien en resetten. Dit veld mag niet leeg zijn en moet een nummer zijn.

```
self.api_port_label = ttk.Label(self.frm, text="API port: ")
self.api_port_label.grid(column=0, row=10)
self.api_port = ttk.Entry(self.frm)
self.api_port.insert(0, self.config["api"]["apiPort"])
self.api_port.grid(column=1, row=10)
self.api_port.bind("<KeyRelease>", lambda e: self.indicate_change(self.api_port_label))
```

12. Locatie voor camerabeelden Widget:

- **Label:** Een label met de tekst 'Directory voor camera images:' dient om het bijbehorende invoerveld en de browse-knop aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de elfde rij (row 11) van het grid.
- **Invoerveld:** Direct naast het label, in de tweede kolom (column 1), wordt een invoerveld geplaatst waar de locatie van de opgeslagen camerabeelden staat aangegeven. Dit veld is vooraf ingevuld met het pad uit het 'config.json' bestand, zoals gespecificeerd onder de 'api' sectie. Het veld is eveneens uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, om aan te geven dat er wijzigingen zijn.
- **Browse-knop:** Naast het invoerveld bevindt zich een knop, geplaatst in de derde kolom (column 2), die gelabeld is met "Browse". Wanneer op deze knop wordt geklikt, wordt een dialoogvenster geopend waarmee gebruikers een map kunnen selecteren waarin de camerabeelden worden opgeslagen. Deze interactie zorgt voor een gebruiksvriendelijke manier om de locatie te wijzigen.

Directory for camera images:

Hier moet het pad naar de folder waar de afbeeldingen in opgeslagen gaan worden invullen. Je kan de 'Browse' knop gebruiken, deze gaat een file explorer openen waarin je een folder kan selecteren.

```
self.image_location_label = ttk.Label(self.frm, text="Directory for camera images: ")
self.image_location_label.grid(column=0, row=11)
self.image_location = ttk.Entry(self.frm)
self.image_location.insert(0, self.config["api"]["imageLocation"])
self.image_location.grid(column=1, row=11)
self.browse_button = ttk.Button(self.frm, text="Browse", command=lambda: self.open_directory_dialog(self.image_location, self.image_location_label))
self.browse_button.grid(column=2, row=11)
self.image_location.bind("<KeyRelease>", lambda e: self.indicate_change(self.image_location_label))
```

13. Widget voor Logo-afbeelding:

- Label: Een label met de tekst 'Logo image:' wordt gebruikt om zowel het invoerveld als de browse-knop aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de twaalfde rij (row 12) van het grid.
- Invoerveld: In de tweede kolom (column 1) van dezelfde rij wordt een invoerveld voor de locatie van het logo geplaatst. Dit veld is vooraf ingevuld met het pad uit het 'config.json' bestand, onder de 'api' sectie gespecificeerd. Dit veld is ook uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, die aangeeft wanneer er wijzigingen zijn gemaakt.
- Browse-knop: Naast het invoerveld bevindt zich een knop, geplaatst in de derde kolom (column 2), die gelabeld is met "Browse". Wanneer op deze knop wordt geklikt, wordt een dialoogvenster geopend waarmee gebruikers een afbeelding kunnen selecteren voor het logo. Deze interactie zorgt voor een gebruiksvriendelijke manier om het logo te wijzigen.

Logo image: Hier moet het pad naar de afbeelding van het logo invullen. Je kan de 'Browse' knop gebruiken, deze gaat een file explorer openen waarin je een afbeelding kan selecteren.

```
self.logo_location_label = ttk.Label(self.frm, text="Logo image: ")
self.logo_location_label.grid(column=0, row=12)
self.logo_location = ttk.Entry(self.frm)
self.logo_location.insert(0, self.config["api"]["logoLocation"])
self.logo_location.grid(column=1, row=12)
self.browse_logo_button = ttk.Button(self.frm, text="Browse", command=lambda: self.open_file_dialog(self.logo_location, self.logo_location_label))
self.browse_logo_button.grid(column=2, row=12)
self.logo_location.bind("<KeyRelease>", lambda e: self.indicate_change(self.logo_location_label))
```

14. Widget voor Transparante Logo-afbeelding:

- Label: Een label met de tekst 'Transparent logo image:' dient om het bijbehorende invoerveld en de browse-knop aan te duiden. Dit label wordt geplaatst in de eerste kolom (column 0) en de dertiende rij (row 13) van het grid.
- Invoerveld: Direct naast het label, in de tweede kolom (column 1), wordt een invoerveld geplaatst waar de locatie van de transparante logo-afbeelding staat aangegeven. Dit veld is vooraf ingevuld met het pad uit het 'config.json' bestand, zoals gespecificeerd onder de 'api' sectie. Het veld is eveneens uitgerust met een 'KeyRelease' event dat de functie 'indicate_change' activeert, om aan te geven dat er wijzigingen zijn.
- Browse-knop: Naast het invoerveld bevindt zich een knop, geplaatst in de derde kolom (column 2), die gelabeld is met "Browse". Wanneer op deze knop wordt geklikt, wordt een dialoogvenster geopend waarmee gebruikers een transparante logo-afbeelding kunnen selecteren. Deze interactie zorgt voor een gebruiksvriendelijke manier om de locatie te wijzigen.

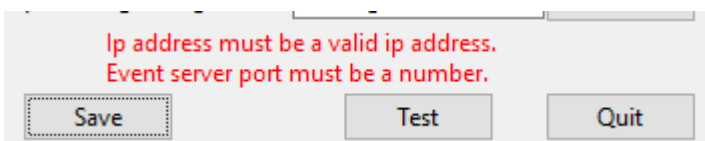
Transparent logo image: Hier moet het pad naar de afbeelding van het logo dat een transparante achtergrond heeft invullen. Je kan de

'Browse' knop gebruiken, deze gaat een file explorer openen waarin je een afbeelding kan selecteren.

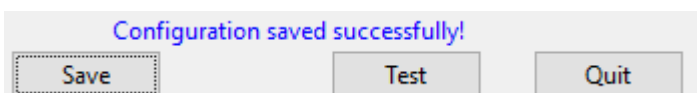
```
self.transparent_logo_location_label = ttk.Label(self.frm, text="Transparent logo image: ")
self.transparent_logo_location_label.grid(column=0, row=13)
self.transparent_logo_location = ttk.Entry(self.frm)
self.transparent_logo_location.insert(0, self.config["api"]["logoTransparentLocation"])
self.transparent_logo_location.grid(column=1, row=13)
self.browse_transparent_logo_button = ttk.Button(self.frm, text="Browse", command=lambda: self.open_file_dialog(self.transparent_logo_location, self.transparent_logo_location_label))
self.browse_transparent_logo_button.grid(column=2, row=13)
self.transparent_logo_location.bind("<KeyRelease", lambda e: self.indicate_change(self.transparent_logo_location_label))
```

15. Widget voor Foutmeldingen:

- **Label:** Een label zonder initiële tekst (tekst=""), maar met een rode tekstkleur (foreground="red"), wordt gebruikt om foutmeldingen weer te geven. Het label wordt over drie kolommen (columnspan=3) uitgestrekt en is geplaatst in de veertiende rij (row 14) van het grid. Dit label dient als feedbackmechanisme voor de gebruiker bij eventuele fouten tijdens het gebruik van de applicatie.



De foutmeldingen zijn standaard leeg. Maar als er een ingevuld veld niet correct is dan komt dit boven de knoppen te staan.



Als al de ingevulde waarden in orde zijn en alles is succesvol opgeslagen dan zal hier een blauwe tekst komen te staan om aan te tonen dat het opslagen gelukt is.

```
self.error_message_label = ttk.Label(self.frm, text="", foreground="red")
self.error_message_label.grid(column=0, row=14, columnspan=3)
```

16. Besturingsknoppen:

- **Opslaanknop ("Save"):** Een knop gelabeld 'Save' wordt geplaatst in de eerste kolom (column 0) en de vijftiende rij (row 15). Deze knop is gekoppeld aan de functie 'self.save', die bedoeld is om de huidige configuratie te valideren en op te slaan.
- **Testknop ("Test"):** Een knop met het label 'Test' bevindt zich in de tweede kolom (column 1) en dezelfde rij. Deze knop is verbonden met 'self.test_config', een functie bedoeld om de configuratie te testen, door de verschillende servers aan proberen te roepen.
- **Afsluitknop ("Quit"):** Een knop gelabeld 'Quit' staat in de derde kolom (column 2) en ook in rij 15. Deze knop is gekoppeld aan de functie 'root.destroy', die het hoofdvenster sluit en de applicatie afsluit.



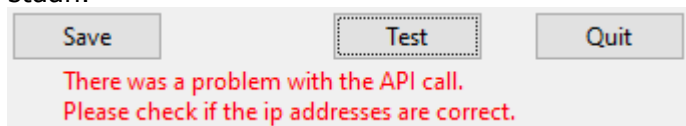
```
ttk.Button(self.frm, text="Save", command=self.save).grid(column=0, row=15)
ttk.Button(self.frm, text="Test", command=self.test_config).grid(column=1, row=15)
ttk.Button(self.frm, text="Quit", command=root.destroy).grid(column=2, row=15)
```

17. Widget voor Testmeldingen:

- **Label:** Net zoals bij de foutmeldingen, wordt hier een label gebruikt om berichten gerelateerd aan de testresultaten weer te geven. Dit label heeft geen initiële tekst (tekst=""), maar heeft een groene tekstkleur (foreground="green") om positieve feedback of geslaagde testresultaten aan te duiden. Het is eveneens uitgestrekt over drie kolommen (columnspan=3) en geplaatst in de zestiende rij (row 16) van het grid.



De berichten van de tests komen onder de knoppen te staan. Als al de API calls goed zijn zal dit in een groene kleur staan.



Als er een fout is dan zal deze informatie in het rood komen te staan.

```
self.test_message_label = ttk.Label(self.frm, text="", foreground="green")
self.test_message_label.grid(column=0, row=16, columnspan=3)
```

18. Applicatielus:

- **Mainloop:** Aan het einde van de widgetdefinities wordt 'root.mainloop()' aangeroepen, wat de Tkinter-eventloop start. Dit is essentieel voor het operationeel houden van de gebruikersinterface, waardoor deze kan reageren op gebruikersinput en andere gebeurtenissen.

```
root.mainloop()
```

Deze widgets vormen de basis voor het invoeren en wijzigen van de gebruikersgegevens, en de geïmplementeerde event handlers zorgen voor onmiddellijke feedback bij elke verandering, wat essentieel is voor een dynamische en responsieve gebruikerservaring.

3.2 Ondersteunende Functies van de Interface

In dit hoofdstuk behandelen we de verschillende hulp- en utility-functies die gebruikt worden binnen de interface van onze applicatie. Deze functies zijn essentieel voor het bieden van een dynamische en interactieve gebruikerservaring.

3.2.1 Functie 'indicate_change'

Doel:

De 'indicate_change' functie is ontworpen om visuele feedback te geven aan de gebruiker wanneer een wijziging is aangebracht in een invoerveld. Dit helpt om de aandacht van de gebruiker te vestigen op velden die gewijzigd zijn en mogelijk opgeslagen moeten worden.

Parameters:

label: Dit is het label widget dat gekoppeld is aan een specifiek invoerveld. Het label wordt gebruikt om visuele indicaties (zoals een asterisk) te tonen die aangeven dat er wijzigingen zijn gemaakt.

Functionaliteit:

De functie controleert of het tekstlabel al begint met een asterisk (*). Als dit niet het geval is, voegt het een asterisk toe aan het begin van de tekst van het label. Deze asterisk fungeert als een duidelijke indicator dat er wijzigingen zijn aangebracht in het invoerveld die nog niet zijn opgeslagen of bevestigd.

Toepassing:

'indicate_change' wordt gebonden aan het 'KeyRelease' event van een invoerveld. Wanneer de gebruiker een toets loslaat na het invoeren van tekst, activeert dit event de 'indicate_change' functie, die op haar beurt het gekoppelde label bijwerkt.

Werking:

Als deze functie wordt aangeroepen zal er eerst gekeken worden of het label al begint met een '*'. Als dit het geval is dan is er al een verandering gebeurd die nog niet is opgeslagen. In dit geval moeten we dus geen nieuwe asterisk toevoegen. Als we dit niet doen en er gebeuren veel 'KeyRelease' dan zullen er ook zeer veel asterisks staan, maar deze check lost dit op. Als er nog geen verandering was dan zal er een asterisk vooraan aan het label toegevoegd worden.

```
def indicate_change(self, label):
    if not label.cget("text").startswith("*"):
        label.config(text="*" + label.cget("text"))
```

3.2.2 Functie 'open_directory_dialog'**Doel:**

Deze functie stelt gebruikers in staat een map te kiezen door een dialoogvenster te openen. Het doel is om de interactie met bestandssystemen gebruiksvriendelijker te maken door een grafische interface te bieden voor het selecteren van mappen.

Functionaliteit:

Bij het activeren van deze functie opent het een dialoogvenster waar gebruikers een map kunnen selecteren. Na selectie wordt het pad van de gekozen map automatisch ingevuld in een vooraf gespecificeerd invoerveld binnen de gebruikersinterface van de applicatie. Dit maakt het proces van directoryselectie eenvoudig en foutvrij voor gebruikers.

Werkwijze:

1. **Openen van een dialoogvenster:** Wanneer de gebruiker op de bijbehorende 'Browse' knop klikt, wordt een dialoogvenster geopend waarin mappen kunnen worden gekozen.
2. **Update van het invoerveld:** Na het selecteren van een map, wordt het adres van deze map weergegeven in het desbetreffende invoerveld in de applicatie. Dit voorkomt dat gebruikers handmatig paden moeten typen.

3. **Indicatie van wijzigingen:** Om te signaleren dat het invoerveld is bijgewerkt, wordt een visuele indicator toegevoegd. Dit gebeurt door het label van het invoerveld te updaten, wat helpt om de aandacht van de gebruiker op deze verandering te vestigen.

'directory = filedialog.askdirectory()' gaat het file explorer venster openen.

'entry_widget.delete(0, tk.END)' verwijdert de huidige informatie die als de image locatie staat.

'entry_widget.insert(0, file_path)' vul de geselecteerde folder in het inputveld.

'self.indicate_change(label_widget)' roep de indicate_change definitie aan om aan te tonen dat er een verandering is gebeurd.

```
def open_directory_dialog(self, entry_widget, label_widget):
    directory = filedialog.askdirectory()
    if directory:
        entry_widget.delete(0, tk.END)
        entry_widget.insert(0, directory)
        self.indicate_change(label_widget)
```

Praktische Toepassing:

De functie is vooral nuttig in applicaties waar het beheren van bestanden en mappen een belangrijk onderdeel is van de functionaliteit. Het wordt meestal geactiveerd door een 'Browse' knop naast een invoerveld dat bedoeld is voor het invoeren van directorypaden.

3.2.3 Functie 'open_file_dialog'

Doel:

Deze functie biedt gebruikers de mogelijkheid om een bestandsdialoogvenster te openen waarmee zij afbeeldingsbestanden kunnen selecteren. Het is bedoeld om de gebruikerservaring te verbeteren door een grafische interface te bieden voor het selecteren van bestanden, waardoor het risico op fouten bij het handmatig invoeren van bestandspaden wordt verkleind.

Functionaliteit:

Bij het activeren van deze functie wordt een dialoogvenster geopend waar gebruikers een bestand kunnen kiezen uit de toegestane typen, zoals PNG, JPG, JPEG, BMP en GIF. Na het selecteren van een bestand wordt het pad ervan automatisch ingevoerd in een specifiek invoerveld in de applicatie.

Werkwijze:

1. **Openen van het dialoogvenster:** De functie wordt getriggerd door een 'Browse' knop, en opent een dialoogvenster dat de gebruiker toestaat om door bestanden te bladeren en een geschikt bestand te selecteren.
2. **Update van het invoerveld:** Zodra een bestand is geselecteerd, wordt het volledige pad van het bestand geplaatst in het bijbehorende invoerveld. Dit maakt het eenvoudig voor gebruikers om bestanden te selecteren zonder zich zorgen te maken over padfouten.

3. **Indicatie van wijzigingen:** Om gebruikers te informeren over de update, wordt het label van het invoerveld aangepast. Deze visuele indicator helpt om de aandacht van de gebruiker te vestigen op de wijziging.

'file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg;*.jpeg;*.bmp;*.gif")])' gaat het file explorer venster openen en zorgt ervoor dat je een afbeelding van een van de meegegeven types moet selecteren. 'entry_widget.delete(0, tk.END)' verwijdert de huidige informatie die als afbeelding pad staat. 'entry_widget.insert(0, file_path)' vul de geselecteerde afbeelding in het inputveld. 'self.indicate_change(label_widget)' roep de indicate_change definitie aan om aan te tonen dat er een verandering is gebeurd.

```
def open_file_dialog(self, entry_widget, label_widget):
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg;*.jpeg;*.bmp;*.gif")])
    if file_path:
        entry_widget.delete(0, tk.END)
        entry_widget.insert(0, file_path)
        self.indicate_change(label_widget)
```

Praktische Toepassing:

Deze functionaliteit is essentieel in applicaties waar de selectie van specifieke bestandstypen, vooral afbeeldingen, regelmatig voorkomt. Dit kan bijvoorbeeld gebruikt worden in applicaties voor mediabeheer, content creatie tools, of in instellingen waar gebruikers persoonlijke of configuratie-gerelateerde afbeeldingen moeten uploaden.

3.3 Opslaan en Validatie

Doel:

De 'save' functie is ontworpen om de configuratie-instellingen die een gebruiker heeft ingevoerd in de applicatie te valideren en op te slaan. Deze functie speelt een cruciale rol in het waarborgen van de integriteit van de gegevens voordat deze worden opgeslagen.

Functionaliteit:

De functie voert validatiechecks uit op alle invoervelden. Na validatie slaat het de waarden op in de configuratie als ze geldig zijn, of toont foutmeldingen als de invoer niet voldoet aan de vereisten.

Werkwijze:

Initialisatie: De functie begint met het instellen van 'valid_config' op True als aanname dat de configuratie geldig is. Het gebruikt een lijst genaamd 'error_messages' om foutmeldingen op te slaan die tijdens de validatie ontstaan.

```
def save(self):
    valid_config = True
    error_messages = []
```

Gebruikersnaamvalidatie:

- **Controle:** Eerst controleert het of het 'user' invoerveld niet leeg is.

- **Opslaan:** Als de 'user' invoer geldig is (niet leeg), wordt deze waarde opgeslagen in de configuratie onder 'user["userId"]'.
- **Label Reset:** Het bijbehorende label wordt gereset naar "User: ".
- **Foutafhandeling:** Als de invoer leeg is, wordt een foutmelding toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
if (self.user.get() != ""):
    self.config["user"]["userId"] = self.user.get()
    self.user_label.config(text="User: ")
else:
    error_messages.append("User can't be empty.")
    valid_config = False
```

Wachtwoordvalidatie:

- **Controle:** Vervolgens controleert de functie of het 'password' invoerveld niet leeg is.
- **Opslaan:** Als het wachtwoord geldig is, wordt deze waarde opgeslagen in de configuratie onder 'user["password"]'.
- **Label Reset:** Het bijbehorende label wordt gereset naar "Password: ".
- **Foutafhandeling:** Als het invoerveld leeg is, wordt op een soortgelijke wijze een foutmelding toegevoegd en 'valid_config' wordt op 'False' gezet.

```
if (self.password.get() != ""):
    self.config["user"]["password"] = self.password.get()
    self.password_label.config(text="Password: ")
else:
    error_messages.append("Password can't be empty.")
    valid_config = False
```

IP-adresvalidatie:

- **Controle:** Het systeem controleert eerst of het IP-adres invoerveld een geldig IP-adres bevat door gebruik te maken van de 'ip_address' methode uit de 'ipaddress' module.
- **Opslaan:** Als het IP-adres geldig is, wordt deze waarde opgeslagen in de configuratie onder 'settings["ipAddress"]'.
- **Label Reset:** Het bijbehorende label wordt gereset naar "SAFR ip address: ".
- **Foutafhandeling:** Als de invoer geen geldig IP-adres is, wordt een foutmelding "Ip address must be a valid ip address." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
try:
    ipaddress.ip_address(self.ip_address.get())
    self.config['settings']['ipAddress'] = self.ip_address.get()
    self.ip_address_label.config(text="SAFR ip address: ")
except ValueError:
    error_messages.append("Ip address must be a valid ip address.")
    valid_config = False
```

Event Server-poortvalidatie:

- Controle: De code controleert of het Event Server-poort invoerveld een getal bevat door te proberen deze om te zetten naar een geheel getal met 'int()'.
- Opslaan: Als de conversie succesvol is, wordt de poortwaarde opgeslagen in de configuratie onder 'settings["eventServerPort"]'.
- Label Reset: Het label wordt gereset naar "Event server port: ".
- Foutafhandeling: Als de conversie faalt (bijvoorbeeld als de invoer tekst bevat in plaats van enkel getallen), wordt een foutmelding "Event server port must be a number." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
try:
    int(self.event_server_port.get())
    self.config["settings"]["eventServerPort"] = self.event_server_port.get()
    self.event_server_port_label.config(text="Event server port: ")
except ValueError:
    error_messages.append("Event server port must be a number.")
    valid_config = False
```

Validatie van Computer Vision (CoVi) Server-poort:

- Controle: Deze validatiestap controleert of de waarde in het invoerveld voor de CoVi server-poort een geheel getal is door gebruik te maken van de functie 'int()'.
- Opslaan: Als de waarde succesvol naar een geheel getal wordt geconverteerd, wordt deze opgeslagen in de configuratie onder 'settings["computerVisionPort"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "Computer vision (CoVi) server port: ".
- Foutafhandeling: Als de waarde in het invoerveld geen geldig getal is, wordt "Computer vision (CoVi) server port must be a number." toegevoegd aan de lijst van foutmeldingen 'error_messages', en 'valid_config' wordt op 'False' gezet.

```
try:
    int(self.covi_server_port.get())
    self.config["settings"]["computerVisionPort"] = self.covi_server_port.get()
    self.covi_server_port_label.config(text="Computer vision (CoVi) server port: ")
except ValueError:
    error_messages.append("Computer vision (CoVi) server port must be a number.")
    valid_config = False
```

Validatie van Object Server-poort:

- Controle: Er wordt gecontroleerd of de invoer voor de Object server-poort een geheel getal is door te proberen deze om te zetten met 'int()'.
- Opslaan: Als de invoer succesvol wordt omgezet naar een geheel getal, wordt deze waarde opgeslagen in de configuratie onder 'settings["objectServerPort"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "Object server port: ".
- Foutafhandeling: Indien de conversie faalt en de invoer geen geheel getal is, wordt "Object server port must be a number." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
try:
    int(self.object_server_port.get())
    self.config["settings"]["objectServerPort"] = self.object_server_port.get()
    self.object_server_port_label.config(text="Object server port: ")
except ValueError:
    error_messages.append("Object server port must be a number.")
    valid_config = False
```

Validatie van Wachtijd (Wait Time):

- Controle: Er wordt gecontroleerd of de invoer voor de wachtijd een geheel getal is door te proberen deze om te zetten met 'int()'.
- Opslaan: Als de invoer succesvol wordt omgezet naar een geheel getal, wordt deze waarde opgeslagen in de configuratie onder 'waitTime'.
- Label Reset: Het bijbehorende label wordt gereset naar "Wait time: ".
- Foutafhandeling: Indien de conversie faalt en de invoer geen geheel getal is, wordt "Wait time must be a number." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
try:
    self.config["waitTime"] = int(self.wait_time.get())
    self.wait_time_label.config(text="Wait time: ")
except ValueError:
    error_messages.append("Wait time must be a number.")
    valid_config = False
```

Validatie van IP-adres voor IQ Messenger:

- Controle: De invoer voor het IP-adres van IQ Messenger wordt gevalideerd met behulp van 'ipaddress.ip_address()' om te verzekeren dat het een geldig IP-adres is.
- Opslaan: Als het IP-adres geldig is, wordt deze waarde opgeslagen in de configuratie onder 'iqMessenger[ipAddress]'.
- Label Reset: Het bijbehorende label wordt gereset naar "IQ Messenger ip address: ".
- Foutafhandeling: Als de invoer geen geldig IP-adres is, wordt "IQ Messenger ip address must be a valid ip address." toegevoegd aan de lijst van foutmeldingen 'error_messages', en 'valid_config' wordt op 'False' gezet.

```
try:
    ipaddress.ip_address(self.iqmessenger_ip.get())
    self.config["iqMessenger"]["ipAddress"] = self.iqmessenger_ip.get()
    self.iqmessenger_ip_label.config(text="IQ Messenger ip address: ")
except ValueError:
    error_messages.append("IQ Messenger ip address must be a valid ip address.")
    valid_config = False
```

Validatie van API Gebruikersnaam:

- Controle: Eerst controleert het of het API gebruikersnaam invoerveld niet leeg is.
- Opslaan: Als de API gebruikersnaam invoer geldig is (niet leeg), wordt deze waarde opgeslagen in de configuratie onder 'api["username"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "API username: ".
- Foutafhandeling: Als de invoer leeg is, wordt "API username can't be empty." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
if (self.api_user.get() != ""):
    self.config["api"]["username"] = self.api_user.get()
    self.api_user_label.config(text="API username: ")
else:
    error_messages.append("API username can't be empty.")
    valid_config = False
```

Validatie van API Wachtwoord:

- Controle: Er wordt gecontroleerd of het API wachtwoord invoerveld niet leeg is.
- Opslaan: Als het API wachtwoord geldig is (niet leeg), wordt deze waarde opgeslagen in de configuratie onder 'api["password"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "API password: ".

- Foutafhandeling: Als het wachtwoordveld leeg is, wordt "API password can't be empty." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet.

```
if (self.api_password.get() != ""):
    self.config["api"]["password"] = self.api_password.get()
    self.api_password_label.config(text="API password: ")
else:
    error_messages.append("API password can't be empty.")
    valid_config = False
```

Validatie van API Poort:

- Controle: Er wordt gecontroleerd of de invoer voor de API poort een numerieke waarde is.
- Opslaan: Als de API poort een geldig nummer is, wordt deze waarde opgeslagen in de configuratie onder 'api["apiPort"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "API port: ".
- Foutafhandeling: Als de invoer geen geldig nummer is, wordt "API port must be a number." toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet. Dit zorgt ervoor dat de API poort altijd correct geformatteerd is voor netwerkcommunicatie.

```
try:
    int(self.api_port.get())
    self.config["api"]["apiPort"] = self.api_port.get()
    self.api_port_label.config(text="API port: ")
except ValueError:
    error_messages.append("API port must be a number.")
    valid_config = False
```

Validatie van Locatie voor Camera-afbeeldingen:

- Controle: Er wordt gecontroleerd of het opgegeven pad voor de opslag van camera-afbeeldingen bestaat op het bestandssysteem.
- Opslaan: Als het pad bestaat, wordt het pad opgeslagen in de configuratie onder 'api["imageLocation"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "Directory for camera images: ".
- Foutafhandeling: Als het opgegeven pad niet bestaat, wordt "Camera image location does not exist" toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet. Dit zorgt ervoor dat de applicatie verwijst naar een geldige map, waardoor fouten in bestandsbehandeling voorkomen worden.


```

try:
    image_location = self.image_location.get()
    if os.path.exists(image_location):
        self.config["api"]["imageLocation"] = image_location
        self.image_location_label.config(text="Directory for camera images: ")
    else:
        error_messages.append("Camera image location does not exist")
        valid_config = False
except ValueError:
    error_messages.append("Problem with the camera image location")
    valid_config = False

```

Validatie van Logo Afbeeldingslocatie:

- Controle: Er wordt gecontroleerd of het opgegeven pad voor de logo afbeelding bestaat op het bestandssysteem.
- Opslaan: Als het pad bestaat, wordt het pad opgeslagen in de configuratie onder 'api["logoLocation"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "Logo image: ".
- Foutafhandeling: Als het opgegeven pad niet bestaat, wordt "Logo image location does not exist" toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet. Dit verzekert dat de opgegeven pad naar een bestaande afbeelding verwijst, wat cruciaal is voor de representatie van de bedrijfsidentiteit.

```

try:
    logo_location = self.logo_location.get()
    if os.path.exists(logo_location):
        self.config["api"]["logoLocation"] = logo_location
        self.logo_location_label.config(text="Logo image: ")
    else:
        error_messages.append("Logo image location does not exist")
        valid_config = False
except ValueError:
    error_messages.append("Problem with the logo image location")
    valid_config = False

```

Validatie van Transparante Logo Afbeeldingslocatie:

- Controle: Er wordt gecontroleerd of het opgegeven pad voor de transparante logo afbeelding bestaat op het bestandssysteem.
- Opslaan: Als het pad bestaat, wordt het pad opgeslagen in de configuratie onder 'api["logoTransparentLocation"]'.
- Label Reset: Het bijbehorende label wordt gereset naar "Transparent logo image: ".

- **Foutafhandeling:** Als het opgegeven pad niet bestaat, wordt "Transparent logo image location does not exist" toegevoegd aan 'error_messages' en 'valid_config' wordt op 'False' gezet. Dit is belangrijk om ervoor te zorgen dat de transparante logo correct geladen kan worden voor gebruik in verschillende media.

```
try:
    transparent_logo_location = self.transparent_logo_location.get()
    if os.path.exists(transparent_logo_location):
        self.config["api"]["logoTransparentLocation"] = transparent_logo_location
        self.transparent_logo_location_label.config(text="Transparent logo image: ")
    else:
        error_messages.append("Transparent logo image location does not exist")
        valid_config = False
except ValueError:
    error_messages.append("Problem with the transparent logo image location")
    valid_config = False
```

Afhandeling en Opslaan van Configuratie:

- **Proberen te openen en schrijven:** Het systeem probeert het configuratiebestand 'config.json' te openen voor schrijftoegang met de 'with open()' statement. De 'json.dump()' functie wordt gebruikt om de configuratiegegevens (die zijn opgeslagen in de 'self.config'-variabele) naar dit bestand te schrijven. De 'indent=4' parameter zorgt voor een nette en leesbare opmaak van het JSON-bestand met een inspringing van 4 spaties.
- **Succesmelding:** Als het opslaan van de configuratie succesvol is, wordt het label 'self.error_message_label' geüpdatet met de tekst "Configuratie succesvol opgeslagen!" en de voorgrondkleur wordt blauw ingesteld om een positief resultaat aan te geven.
- **Foutberichten weergeven:** Als 'valid_config' 'False' is, wat betekent dat er een of meer problemen zijn geïdentificeerd, worden de foutberichten die verzameld zijn in 'error_messages' weergegeven in 'self.error_message_label'. Dit label toont alle foutberichten gescheiden door een nieuwe regel, en de tekstkleur wordt ingesteld op rood om de urgentie van de problemen aan te geven.
- **Succesvolle Opslag:** Als het opslaan succesvol is, wordt een bericht weergegeven in 'self.error_message_label' dat de configuratie succesvol is opgeslagen. De tekstkleur van dit label wordt veranderd naar blauw om het succes aan te geven.

```
try:
    with open('./config.json', 'w') as f:
        json.dump(self.config, f, indent=4)

    self.error_message_label.config(text="Configuration saved successfully!", foreground="blue")
except Exception as e:
    self.error_message_label.config(text=f"Failed to save the configuration: {e}", foreground="red")

if not valid_config:
    self.error_message_label.config(text="\n".join(error_messages), foreground="red")
```

3.3.1 IQ Messenger inlog gegevens

De gebruikersnaam en wachtwoord van IQ Messenger moet ook in de 'config.json' file opgeslagen worden maar ik heb niet genoeg tijd gehad om deze in de UI toe te voegen. Je kan deze waardes vinden in het IQ Messenger deel ('iqMessenger'). Bij 'username' kan je de gebruikersnaam invullen en bij 'password' kan je het wachtwoord invullen. Dit staat wel in de config file en kan hier ook aangepast worden maar heeft dus niet de validatie. Let wel op om de test knop te gebruiken moet dit ook ingevuld zijn anders zal de IQ Messenger server test een error geven.

```
"iqMessenger": {
  "ipAddress": "94.140.172.106",
  "username": "admin",
  "password": "admin"
},
```

3.4 Configuratie Test

3.4.1 Functie: 'test_config'

De 'test_config' functie is ontworpen om de configuratie-instellingen van de applicatie te testen door verzoeken te sturen naar verschillende API-eindpunten, gebruikmakend van de opgeslagen configuratieparameters zoals gebruikers-id, wachtwoord en IP-adressen.

Configuratie variabelen ophalen:

De functie haalt de benodigde configuratiegegevens op:

- Gebruikers-ID en wachtwoord voor authenticatie.
- IP-adres en poortnummers voor de Event Server, Computer Vision (CoVi) Server en Object Server.
- IP-adres voor de IQ Messenger service.

```
def test_config(self):
    user_id = self.config["user"]["userId"]
    password = self.config["user"]["password"]
    ip_address = self.config["settings"]["ipAddress"]
    event_server_port = self.config["settings"]["eventServerPort"]
    covi_server_port = self.config["settings"]["computerVisionPort"]
    object_server_port = self.config["settings"]["objectServerPort"]

    iq_messenger_ip = self.config["iqMessenger"]["ipAddress"]
```

API Aanroepen:

Met deze gegevens maakt het functie vervolgens meerdere HTTP GET-verzoeken naar de respectievelijke servers:

- Een aanvraag naar de Event Server om site-informatie op te halen.
- Een aanvraag naar de CoVi Server voor het ophalen van persoonstypegegevens.
- Een aanvraag naar de Object Server voor voorkeursinstellingen.

- Een aanvraag naar de IQ Messenger om de connectiviteit te testen.

Deze verzoeken gebruiken basisauthenticatie met gebruikers-ID en wachtwoord, en hebben een time-out van 5 seconden om te voorkomen dat de applicatie blijft hangen bij netwerkvertragingen.

```
response_event = requests.get(f"http://{ip_address}:{event_server_port}/sites?sinceTime=0", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
response_covi = requests.get(f"http://{ip_address}:{covi_server_port}/personTypes", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
response_object = requests.get(f"http://{ip_address}:{object_server_port}/preferences/list", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
iq_messenger = requests.get(f"http://{iq_messenger_ip}/IQHttpCommons/service/httpio?device=Any&message=test", timeout=5, verify=False)
self.handle_api_response(response_event, response_covi, response_object, iq_messenger)
```

Foutafhandeling:

De functie is voorzien van een algemene 'try-except' blok om fouten tijdens het netwerkverkeer op te vangen, zoals verbindingsproblemen of time-outs. Bij een uitzondering wordt een foutbericht weergegeven via 'self.test_message_label', wat aangeeft dat er een probleem was met de API-aanroepen.

```
try:
    response_event = requests.get(f"http://{ip_address}:{event_server_port}/sites?sinceTime=0", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
    response_covi = requests.get(f"http://{ip_address}:{covi_server_port}/personTypes", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
    response_object = requests.get(f"http://{ip_address}:{object_server_port}/preferences/list", headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"}, timeout=5)
    iq_messenger = requests.get(f"http://{iq_messenger_ip}/IQHttpCommons/service/httpio?device=Any&message=test", timeout=5, verify=False)
    self.handle_api_response(response_event, response_covi, response_object, iq_messenger)
except:
    print("error api call")
    self.test_message_label.config(
        text="There was a problem with the API call.\n"
        "Please check if the ip addresses are correct.",
        foreground="red"
    )
```

3.4.2 Functie: 'handle_api_response'

Deze functie wordt aangeroepen door 'test_config' om de antwoorden van de API-verzoeken te verwerken.

Statuscode Validatie:

De functie controleert de HTTP-statuscodes van alle ontvangen reacties:

- Als alle reacties de statuscode 200 (OK) hebben, wordt een succesmelding getoond.

```
if all(response.status_code == 200 for response in responses.values()):
    print("Successfully fetched the data")
    self.test_message_label.config(text="The test call was correct.", foreground="green")
    return
```

- Bij specifieke foutcodes (401, 500, 400, 404), wordt er een gericht foutbericht getoond afhankelijk van de aard van de fout:
 - **401 Unauthorized:** Probleem met autorisatie.

```

if response.status_code == 401:
    self.test_message_label.config(
        text="There was a problem with the authorization.\n"
        "Check if the user and the password are correct and if the user has the correct rights.",
        foreground="red"
    )
    return

```

- **500 Internal Server Error:** Serverfout aan de kant van de API.

```

if response.status_code == 500:
    self.test_message_label.config(
        text=f"There was a problem with the {api_name} server.",
        foreground="red"
    )
    return

```

- **400 Bad Request:** Onjuist verzoek, vaak gerelateerd aan configuratie-instellingen.

```

if response.status_code == 400:
    self.test_message_label.config(
        text=f"There was a bad request with the {api_name} server.\n"
        f"Please check if the {api_name} port is correct.",
        foreground="red"
    )
    return

```

- **404 Not Found:** De API kan niet gevonden worden, wat meestal duidt op een verkeerde URL of poortinstelling.

```

if response.status_code == 404:
    self.test_message_label.config(
        text=f"The {api_name} API could not be found.\n"
        f"Please check if the {api_name} port is correct.",
        foreground="red"
    )
    return

```

Uitzonderlijke situaties:

Als de antwoorden andere statuscodes bevatten of als er een ander type fout optreedt, wordt er een algemeen foutbericht weergegeven met details van de problematische serverresponsen.

```

self.test_message_label.config(text=f"An unexpected error occurred with one of the servers.\nEvent server: {responses['event']}\nComputer vision (CoV1): {responses['computer vision (CoV1)']}", foreground="red")

```

Deze methoden zorgen voor een grondige validatie van de netwerkconfiguratie en geven duidelijke, bruikbare feedback aan de gebruiker over de staat van de systeemverbindingen, wat cruciaal is voor het onderhouden van een betrouwbare netwerkinfrastructuur.

4 DWAALDETECTIE

Welkom bij het kerngedeelte van deze applicatie: het dwaaldetectieprogramma. Hier vindt de essentiële functionaliteit plaats die het hart van de applicatie vormt. Dit programma is ontworpen om te reageren op nieuwe gebeurtenissen die worden gedetecteerd door de SAFR-API. Het doel is om te bepalen of deze gebeurtenissen een potentiële dwaaldetectiesituatie vertegenwoordigen en zo ja, om passende acties te ondernemen.

In dit hoofdstuk duiken we dieper in op de logica en processen die achter de dwaaldetectie liggen. We beginnen met het bekijken van de recente gebeurtenissen die door de SAFR-API zijn vastgelegd. Vervolgens wordt elke gebeurtenis geanalyseerd om te bepalen of deze een bewoner vertegenwoordigt die mogelijk dwaalt. Dit omvat een nauwkeurige identificatie van de bewoner en het controleren van hun rechten om specifieke deuren te gebruiken.

Een belangrijk aspect van dwaaldetectie is het bepalen van de aanwezigheid van een begeleider bij de bewoner. Hierbij wordt rekening gehouden met de door de gebruiker ingestelde tijd in de interface, wat de maximale tijd is die een bewoner zonder toezicht kan doorbrengen voordat er een alarm wordt geactiveerd. Als er geen begeleider aanwezig is, wordt het alarm geactiveerd en worden de nodige stappen ondernomen om de situatie te beheren en de bewoner veilig te houden.

We zullen ook bekijken hoe het programma omgaat met het verzenden van meldingen naar de IQ Messenger-server, inclusief het verstrekken van relevante informatie en afbeeldingen die helpen bij het begrijpen van de situatie. Dit alles gebeurt naadloos en effectief, dankzij de nauwkeurige verwerking van gegevens en de gestroomlijnde communicatie tussen verschillende systemen.

Kortom, dit hoofdstuk biedt een diepgaand inzicht in hoe het dwaaldetectieprogramma werkt en hoe het bijdraagt aan de algemene functionaliteit en doelstellingen van de applicatie.

4.1 Logging

In dit hoofdstuk zullen we de functionaliteit van logboekregistratie verkennen, een cruciaal onderdeel van het systeem voor dwaaldetectie. Logboeken spelen een essentiële rol bij het vastleggen van gebeurtenissen, het volgen van activiteiten en het bieden van inzicht in de werking van de applicatie.

4.1.1 Logging configuratie

1. Logboekformaat: Hier wordt het formaat van het logboek gedefinieerd met behulp van een string die verschillende variabele placeholders bevat, zoals `'%(event_time)s'`, `'%(event)s'`, etc. Dit bepaalt hoe de informatie wordt opgemaakt wanneer deze wordt gelogd.

```
log_format = '%(event_time)s;%(event)s;%(person)s;%(alarm)s;%(reason)s;%(door)s;%(usable_doors)s'
```

De informatie die we bijhouden is de tijd van het event `'%(event_time)s'`, of dit een event is of een error `'%(event)s'` (In geval van een error komt hier het error bericht te staan.), of deze persoon een bewoner of begeleider is `'%(person)s'`, of er een alarm is of niet `'%(alarm)s'`, waarom er een alarm of geen alarm is `'%(reason)s'`, Op welke deur de persoon is gedetecteerd `'%(door)s'` en welke deuren deze persoon kan gebruiken `'%(usable_doors)s'`.

2. Logger-configuratie: Er wordt een aangepast logger-object gemaakt met de naam `'customLogger'` en ingesteld op het niveau `'INFO'`. Dit betekent dat alleen logberichten van het niveau `'INFO'` en hoger worden vastgelegd.

```
logger = logging.getLogger('customLogger')
logger.setLevel(logging.INFO)
```

3. Logboekhandler: Hier wordt een logboekhandler gemaakt, `'TimedRotatingFileHandler'`, die het logboek naar een bestand met de naam `'log.txt'` in de map `'./logs/'` schrijft. Het logbestand wordt dagelijks geroteerd (bij middernacht), met een interval van 1 dag en maximaal 30 back-upbestanden. Wat betekend dat we de logs van maximaal 30 dagen bijhouden.

```
log_handler = TimedRotatingFileHandler(
    './logs/log.txt',
    when='midnight',
    interval=1,
    backupCount=30,
)
```

4. Formattering: Een formatter wordt gemaakt met het eerder gedefinieerde logboekformaat (`'log_format'`) en een specifieke datumnotatie (`'datefmt'`). Deze formatter wordt vervolgens toegewezen aan de logboekhandler om de opmaak van de logberichten te configureren.

```
formatter = logging.Formatter(log_format, datefmt='%Y-%m-%d %H:%M:%S')
log_handler.setFormatter(formatter)
```

5. Toevoegen van de handler aan de logger: Tenslotte wordt de logboekhandler toegevoegd aan de logger, zodat deze wordt gebruikt om logboekberichten naar het opgegeven bestand te schrijven.

```
logger.addHandler(log_handler)
```

4.1.2 Toevoegen aan de log

Deze functie accepteert verschillende parameters die informatie bevatten over een specifieke gebeurtenis die moet worden vastgelegd in het logboek. Deze parameters omvatten 'event_time' (tijdstip van de gebeurtenis), 'event' (type gebeurtenis), 'person_type' (Bewoner of Begeleider), 'alarm' (alarmstatus), 'reason' (reden van het alarm of geen alarm), 'door' (deur waar de gebeurtenis plaatsvond) en 'usable_doors' (Deuren die de bewoner mag gebruiken).

```
def add_to_log(event_time, event, person_type, alarm, reason, door, usable_doors):
```

Binnen de functie wordt de 'info'-methode van de logger aangeroepen om een logboekbericht te maken en vast te leggen. Het logbericht bevat de informatie over de gebeurtenis die als parameters zijn doorgegeven. De 'extra'-parameter wordt gebruikt om extra informatie door te geven aan de formatter, zodat deze kan worden opgenomen in het logboekbericht.

```
logger.info('', extra={
    'event_time': event_time,
    'event': event,
    'person': person_type,
    'alarm': alarm,
    'reason': reason,
    'door': door,
    'usable_doors': usable_doors
})
```

4.1.3 Event loggen

Om de events te loggen wordt de 'log_line' definitie gebruikt. De parameters die meegegeven moeten worden zijn 'has_begeleider' (of er een begeleider bij de bewoner was), 'can_use_door' (of de bewoner de deur mag gebruiken), 'alarm' (of er een alarm is of niet), 'done' (een kopie van de done list waarin al de overlopen events staan), 'groups' (de lijst van dueren die de bewoner mag gebruiken), 'event_start_time' (de starttijd van het event).

```
def log_line(has_begeleider, can_use_door, alarm, done, groups, event_start_time):
```

Deze definitie wordt enkel gebruikt om events te loggen dus de 'events' variabele wordt op "events" gezet.

```
event = "event"
```

4.1.3.1 Loggen met een begeleider

We gaan kijken of er een begeleider bij was. In dat geval is de 'has_begeleider' gezet op 'True'. Aangezien er een begeleider is moeten we al de events in de 'done' list toevoegen aan de log. Aangezien deze events nadien niet meer worden nagekeken. Om hiervoor te zorgen lopen we over al de events in de 'done' list.

```
if (has_begeleider == True):
    for person in done:
```


Dan gaan we het tijdsverschil tussen het event waar we in de loop op zitten en het eerste event berekenen. Dit moeten we doen aangezien we enkel de tijd van het eerste event hebben. Maar aangezien we de tijd van dit event willen weten berekenen we het verschil. Dit doen we door de starttijd van het eerste event (`done[0]["startTime"]`) af te trekken van de starttijd van dit event (`person["startTime"]`). Dan moeten we dit ook nog delen door 1000 aangezien dit overeenstemt met 1 seconde.

```
sleep = ((person["startTime"]-done[0]["startTime"])/1000)
if (sleep > wait_time or sleep < 0):
    sleep = 0
```

Nadat we het tijdsverschil hebben berekend kunnen we de starttijd van dit event berekenen. Dit doen we door het tijdsverschil toe te voegen aan de starttijd van het eerste event en deze waarde slagen we op in de variabele `'event_time'`. Daarna gaan we de tijd berekenen tot waar het programma stopt met kijken naar een begeleider (`'wait_time'`). Als laatste gaan we deze tijden omzetten naar een label dat we kunnen lezen. Hierin wordt de tijd opgeslagen in het formaat dag/maand/jaar starttijd uren/minuten/seconden – eindtijd uren/minuten/seconden.

```
event_time = event_start_time + timedelta(seconds=sleep)
end_time = event_time + timedelta(seconds=wait_time)
date_time = event_time.strftime("%d/%m/%Y %H:%M:%S") + f" - {end_time.strftime('%H:%M:%S')}
```

In de variabele `'person_type'` gaan we opslaan of de persoon in dit event een begeleider of een bewoner is. Deze informatie staat in `'person["personType"]'`. In de variabele `'source_id'` gaan we opslaan op welke deur deze detectie is gebeurd. Deze informatie staat in `'person["sourceId"]'`. Dan zetten we de reden in `'reason'` naar "Begeleider" aangezien er geen alarm aangeroepen gaat worden omdat er een begeleider bij is.

```
person_type = person["personType"]
source_id = person["sourceId"]
reason = "Begeleider"
```

We gaan de `'get_groups'` definitie voor dit event aanroepen. Deze definitie gaat de deuren die de persoon mag gebruiken oproepen en terugsturen als een lijst. In de `'if'`-blok gaan we kijken of de lijst leeg is. Dit doen we door te kijken of de lengte van de lijst groter is dan 0. Dan zetten we de deuren als de lijst die we van de definitie hebben gekregen. Als dit niet het geval is dan kan deze persoon geen deuren gebruiken en zetten we de deuren naar de tekst "no doors".

```
groups = get_groups(person["personId"], event_start_time)
if (len(groups) > 0):
    doors = groups
else:
    doors = "no doors"
```

Nu hebben we al de informatie om dit event te loggen. Dus we sturen al de nodige informatie naar de definitie `'add_to_log'` wat dit in de log file gaat plaatsen.

```
add_to_log(date_time,event,person_type,alarm,reason,source_id,doors)
```

4.1.3.2 Loggen zonder begeleider

Als er geen begeleider bij is dan willen we enkel het eerste event loggen. Het is enkel het eerste event omdat dit het enige event is dat effectief volledig is nagekeken. De andere events worden achteraf nog nagekeken.

Aangezien we enkel naar het eerste event kijken moeten we het tijdsverschil niet berekenen dus we kunnen de starttijd gebruiken. Daarna gaan we de tijd berekenen tot waar het programma stopt met kijken naar een begeleider ('wait_time'). Als laatste gaan we deze tijden omzetten naar een label dat we kunnen lezen. Hierin wordt de tijd opgeslagen in het formaat dag/maand/jaar starttijd uren/minuten/seconden – eindtijd uren/minuten/seconden.

```
else:
    event_time = event_start_time
    end_time = event_start_time + timedelta(seconds=wait_time)
    date_time = event_time.strftime("%d/%m/%Y %H:%M:%S") + f" - {end_time.strftime('%H:%M:%S')}
```

We gaan kijken of de persoon deze deur kan gebruiken. Als deze persoon de deur mag gebruiken dan zal deze waarde op 'True' staan. Als de persoon de deur mag gebruiken dan zetten we de reden ('reason') op "Can use door". Als de persoon de deur niet mag gebruiken dan zetten we de reden ('reason') op "No begeleider and can't use door".

```
if(can_use_door == True):
    reason = "Can use door"
else:
    reason = "No begeleider and can't use door"
```

In de variabele 'person_type' gaan we opslaan of de persoon van het eerste event een begeleider of een bewoner is. Deze informatie staat in 'done[0]["personType"]'. In de variabele 'source_id' gaan we opslaan op welke deur deze detectie is gebeurd. Deze informatie staat in 'done[0]["sourceId"]'.

```
person_type = done[0]["personType"]
source_id = done[0]["sourceId"]
```

Dan gaan we de deuren die deze persoon mag gebruiken opslaan in de variabele 'doors'. Aangezien het hier gaat over het eerste event moeten we de deuren niet meer opvragen. We kunnen hiervoor de 'groups' variabele gebruiken. Dan gaan we kijken of deze lijst is langer dan 0. Als dit het geval is dan zijn er deuren die de persoon mag gebruiken en dan slagen we deze op in de variabele. Als dit niet het geval is dan zijn er geen deuren die gebruikt mogen worden en dan zetten we de tekst naar "no doors".

```
if (len(groups) > 0):
    doors = groups
else:
    doors = "no doors"
```

Nu hebben we al de informatie om de 'add_to_log' definitie aan te roepen wat dit event in de log file gaat toevoegen.

```
add_to_log(date_time,event,person_type,alarm,reason,source_id,doors)
```

4.1.4 Error loggen

Om errors te loggen gebruiken we de 'log_error' definitie. Hiervoor moeten we de eventtijd meegeven en de foutmelding. Eerst zetten we de tijd om naar een leesbaar template. Daarna starten we een thread om deze foutmelding toe te voegen aan de log file. Dit doen we zodat het toevoegen aan de log file het hoofdprogramma niet gaat vertragen. In de 'args' moeten we al de argumenten meegeven die naar de 'add_to_log' definitie moet. Aangezien we enkel de tijd en de foutmelding willen moeten we enkel deze waardes meegeven, maar omdat de 'add_to_log' meerdere argumenten verwacht moeten we deze ook meegeven maar dan als een lege string. Dan voegen we deze thread toe aan de 'threads' lijst zodat als deze thread klaar is we deze later kunnen verwijderen. Nu kunnen we de thread effectief starten en wordt deze error toegevoegd aan de log file.

```
def log_error(event_time, message):
    date_time = event_time.strftime("%d/%m/%Y %H:%M:%S")
    t = threading.Thread(target=add_to_log, args=(date_time, message, '', '', '', ''), name="log_error")
    threads.append(t)
    t.start()
```

4.2 Programma Initialisatie

Als eerste creëren we de globale variabelen. We zetten de tijdszone op die van Berlijn. Anders gaat de tijdsconversie niet het juiste tijdstip geven. Dan creëren we de 'times' lijst waarin we de volgende tijden van events gaan houden die nog nageken moeten worden. Om de event die al eerder zijn nagekeken bij te houden heb ik gekozen voor een directory genaamd 'checked' aangezien dit makkelijker is om na te kijken dan een lijst. De 'done' lijst wordt gebruikt om bij te houden welke events dat we in deze iteratie hebben nagekeken. In de 'threads' lijst houden we al de verschillende threads bij die we hebben gestart zodat we deze later kunnen verwijderen. Als we de threads niet verwijderen dan gaan deze threads resources verbruiken. De variabele 'last_event_time' is de tijd van het laatste event dat we hebben gedetecteerd. Dit wordt gebruikt wanneer we de events gaan ophalen van de SAFR server en zorgt ervoor dat we geen events gaan overslaan.

```
local_tz = pytz.timezone('Europe/Berlin')

times = []
checked = {}

done = []
threads = []
last_event_time = 0
```

Er is ook een eigen API server die gestart moet worden. Aangezien we graag alles tegelijk starten gaan we een subprocess starten voor de API server. Dit gaat een 2^{de} process aanmaken waarop de API server zal kunnen draaien en zal dus het hoofdprogramma niet vertragen. De API server staat in dezelfde folder en noemt API.py en is dus ook in python gemaakt, deze gegevens moeten we meegeven aan het subprocess. Daarnaast gaan we ook een nieuwe console openen met de creationflag. Dit is niet echt nodig aangezien de API niets in de console gaat plaatsen. Maar het is wel handig om te zien dat de API server nog draait.

```
api_server = subprocess.Popen(['python', './API.py'], creationflags=subprocess.CREATE_NEW_CONSOLE)
```

Om de API's aan te gaan roepen moeten we de config file openen. Hiervoor gebruiken we een 'try-except' blok. We proberen de config.json file te openen in dezelfde folder als dit programma. Als dit lukt dan slagen we de informatie van deze file op in de 'config' variabele. Maar als de file niet gevonden kan worden dan gaat er een melding geprint worden en wordt het programma afgesloten.

```
try:
    with open('./config.json', 'r') as f:
        config = json.load(f)
except Exception as e:
    event_time = datetime.now()
    log_error(event_time, f"Can't find config file.")
    print("The Program is terminated! Could not open/find the config file!")
    raise SystemExit
```

Als de config.json file correct is ingelezen kunnen we al deze informatie er uit halen en opslaan in globale variabelen.

```
user_id = config["user"]["userId"]
password = config["user"]["password"]
ip_address = config["settings"]["ipAddress"]
event_server_port = config["settings"]["eventServerPort"]
computer_vision_port = config["settings"]["computerVisionPort"]
object_service_port = config["settings"]["objectServerPort"]
wait_time = config["waitTime"]
iq_messenger_ip_address = config["iqMessenger"]["ipAddress"]
logo_transparent_location = config["api"]["logoTransparentLocation"]
image_location = config["api"]["imageLocation"]
iq_messenger_username = config["iqMessenger"]["username"]
iq_messenger_password = config["iqMessenger"]["password"]
```

'user_id' is de gebruikersnaam van de SAFR gebruiker.
'password' is het wachtwoord van de SAFR gebruiker.
'ip_address' is het ip adres van de SAFR server.

'event_server_port' is de poort van de SAFR event server. 'computer_vision_port' is de poort van de SAFR computer vision server. 'object_server_port' is de poort van de SAFR object server. 'wait_time' is tijd die we gaan wachten tot op een begeleider. 'iq_messenger_ip_address' is het ip adres van IQ Messenger. 'logo_transparent_location' is de locatie van het transparante deltatechnics logo. Dit wordt niet gebruikt in dit programma maar wordt doorgestuurd naar het programma om de afbeelding op te halen. 'image_location' is de locatie van folder waar we de afbeeldingen tijdelijk gaan opslagen. Dit wordt niet gebruikt in dit programma maar wordt doorgestuurd naar het programma om de afbeelding op te halen. 'iq_messenger_username' is de gebruikersnaam van de IQ Messenger gebruiker. 'iq_messenger_password' is het wachtwoord van de IQ Messenger gebruiker.

```
if (wait_time <= 0):
    wait_time = 1
```

Als de wachttijd kleiner of gelijk is aan 0 dan gaan we dit toch nog op 1 zetten. Dit is zodat er toch nog iets van nakijken gaat gebeuren.

```
device = get_data(f"http://{ip_address}:{event_server_port}/sites?sinceTime=0").json()["sites"][0]
```

Voor een aantal API calls moeten we het device hebben. Deze gaan we ophalen met een API call. Hier worden al de devices gegeven die er zijn sinds het begin van deze server door 'sinceTime' op 0 te zetten. Aangezien er toch maar 1 device gaat zijn zorgen we ervoor dat we altijd een device gaan terugkrijgen. We gaan het eerste

device nemen dat we van de API terugkrijgen en slagen deze naam op in een globale variabele.

```
last_event_time = event_service(0)

since = event_service(0)
```

Om te starten zetten we eerst de 'last_event_time' en 'since' variabelen op de laatst bekende tijd. Hiervoor kunnen we 'event_service' met de tijd 0 aan te roepen. Normaal gaat deze functie wachten voor een nieuw event, maar omdat deze tijd in het verleden ligt gaat er meteen de huidige tijd doorsturen.

4.3 Hoofdloop

```
while(True):
    try:
```

Deze loop zal altijd blijven draaien aangezien we de conditie op 'True' hebben gezet. In deze loop gaan we dan ook de rest van het programma starten als er events zijn aangeroepen. In deze loop staat er een 'try-except' blok. In het 'try' gedeelte staat de aansturing van het programma. En het 'except' deel gaat ervoor zorgen dat als je het programma stopt (met 'ctrl-c') zal er geen error message komen maar een tekst dat het manueel gestopt is. Ook zal de API server hierdoor mee afgesloten worden.

```
if (len(checked) > 0):
    t = threading.Thread(target=clear_checked_list, args=(), name="clear-checked")
    t.start()
    threads.append(t)
```

Het eerste in de loop is deze 'if' functie. Hier gaan we kijken of er iets in de 'checked' dictionary staat aangezien het geen nut heeft om events te proberen verwijderen wanneer er geen events in deze dictionary staan. Dit is de lijst waar we de events die al volledig zijn nagekeken bijhouden. Uiteindelijk willen we deze events wel verwijderen. Anders als het programma al lang aan staat zullen er te veel resources gebruikt moeten worden voor deze dictionary. Dus we gaan hier een thread creëren zodat het verwijderen geen invloed op het hoofdprogramma gaat hebben. Deze thread gaat de functie 'clear_checked_list' aanroepen en deze functie verwacht geen argumenten dus de 'args()' moet leeg blijven. Nadat deze thread is aangemaakt gaan we deze starten en ook toevoegen aan de 'threads' list zodat we deze later ook kunnen verwijderen. Als we de thread niet verwijderen nadat deze klaar is dan zullen er ook resources gebruikt blijven worden, en dit willen we niet.

```
last_mod_date = event_service(since)
```

Hier roepen we de 'event_service' definitie aan. We geven de since tijd die we eerder hebben ingesteld of die we later terugkrijgen van het programma mee zodat de functie kan wachten voor een nieuw event. Als er een response is dan slagen we deze op in 'last_mod_date'. Maar hiermee gaan we eigenlijk enkel kijken of er een 204 error is of niet. Als het wachten op een event langer duurt dan 1 minuut dan zal hier een 204 error gestuurd worden, maar we willen dan wel verder gaan kijken.

```
if (last_mod_date != "204 error"):
    since = get_event(last_event_time, wait_time)
    times.clear()
```

Deze 'if' functie gaat kijken of de response die we hebben gekregen geen 204 error is. Als dit geen 204 error is dan zal de definitie 'get_event' aangeroepen worden. Dit is de definitie die de checks gaat doen en het alarm aansturen. Hiervoor moeten we de tijd van het laatste event meegeven alsook de tijd die we willen wachten op een begeleider. De definitie zal een nieuwe tijd retourneren waarmee we de 'since' tijd kunnen vervangen. Daarna maken we de 'times' lijst ook nog leeg om zeker te zijn dat deze leeg is voor de volgende keer te checken. Aangezien we hierin de tijden houden van events die nog nagekeken moeten worden. Maar als we hier terug zijn geraakt dan zijn deze allemaal nagekeken.

```
else:
    if(len(threads) > 5):
        t = threading.Thread(target=remove_threads, name=f"remove")
        threads.append(t)
        t.start()

    since = event_service(0)
```

Als de response wel een 204 error is of een andere error dan gaan we eerst kijken of we meer dan 5 open threads hebben staan. Dit doen we aan de hand van de 'if' functie. Hierin gaan we kijken of de lengte van de 'threads' lijst groter is dan 5. Als dit het geval is dan maken we een nieuwe thread die de functie 'remove_threads' aanroept die al de threads die klaar zijn gaat verwijderen. We maken hier een thread voor om ervoor te zorgen dat dit het hoofdprogramma niet gaat vertragen. Als deze thread is aangemaakt voegen we deze toe aan de 'threads' lijst en starten we de thread.

Aangezien we een error hadden klopt de 'since' tijd ook niet meer. Hiervoor kunnen we 'event_service' met de tijd 0 aan te roepen. En deze definitie gaat de huidige tijd terugsturen.

```
except KeyboardInterrupt:
    api_server.terminate()
    api_server.wait()
    print("The Program is terminated manually!")
    raise SystemExit
```

Als het programma door een keyboard interrupt ('ctrl-c') gestopt wordt dan zou er normaal een error message getoond worden, en de API server zou dan ook nog apart blijven opstaan. Maar door deze 'except' blok gaan we de API server die geconnecteerd is met 'api_server' kunnen afsluiten door een '.terminate()' te doen en te wachten tot deze is afgesloten. Daarna stoppen we het programma en gaan we een bericht schrijven dat het programma manueel afgesloten is. Als je de applicatie stopt maar het is bezig met te wachten op een event dan kan de applicatie niet stoppen. Maar als er dan iemand gedetecteerd wordt dan zal de applicatie stoppen.

4.4 Definitie 'get_data'

Om de SAFR API's aan te roepen wordt er gebruik gemaakt van de 'get_data' definitie. Om hiervan gebruik te maken zal je mee moeten geven welke API call er gemaakt moet worden en deze definitie zal de response retourneren.

```
def get_data(api):
```

Om de API requests te maken maken we gebruik van de 'request' library. Aangezien al de calls die we moeten doen 'get-requests' zijn kunnen we hier gewoon 'requests.get' zetten. Als eerste variabele die we moeten meegeven is de API call die we willen doen. Deze wordt opgesteld in de andere definities en meegegeven naar deze definitie. Voor de SAFR API moeten we ook een gebruikersnaam en wachtwoord meegeven alsook de directory waar we op werken. Deze moeten we in de headers meegeven met een specifieke tekst. Voor "X-RPC-AUTHORIZATION" is het de gebruikersnaam en wachtwoord dat gesplitst wordt met een ':'. En voor "X-RPC-DIRECTORY" geven we de tekst 'main' mee aangezien dit de enige directory is die we hebben.

```
response = requests.get(f"{api}", headers={"X-RPC-AUTHORIZATION" : f"{user_id}:{password}", "X-RPC-DIRECTORY" : "main"})
```

Als de call gedaan is dan gaan we kijken naar de response. Als de call gelukt is dan krijgen we een status code van 200. Als dit het geval is dan kunnen we de response gewoon retourneren.

```
if response.status_code == 200:
    return response
```

Als de statuscode 204 is dan duurde de call te lang. Deze error komt voor bij de call die gaat wachten op een event. Aangezien de error gewoon zegt dat er voor 1 minuut geen event is moeten we hier niet echt naar kijken, maar als deze error voorkomt moeten we er wel voor zorgen dat de checks niet gaan gebeuren maar dat we wel opnieuw gaan wachten voor nieuwe events. Hiervoor retourneren we "204 error" en de hoofdloop gaat hier dan rekening mee houden.

```
elif (response.status_code == 204):
    return "204 error"
```

Als er een andere response code is dan betekent dit dat er een effectieve fout is. In dit geval gaan we de tijd nemen en deze error in de log file stoppen. Daarna retourneren we het bericht "error" zodat het programma nog wel verder kan draaien.

```
else:
    event_time = datetime.now()
    log_error(event_time, f"{response.status_code} error")
    return "error"
```

4.5 Definitie 'event_service'

De definitie 'event_service' wordt gebruikt om de API aan te roepen die gaat wachten op een nieuw event. Om deze definitie te gebruiken moeten we de laatste tijd meegeven.

```
def event_service(since):
```

Hier gaan we de API call opstellen. We moeten eerst het IP adres en poort van de event server gesplitst door ':' achter de http zetten. Dan moeten we zeggen welke API het is.

In dit geval 'event/status' en we geven hier ook het device mee anders dan gaat de call niet correct werken. Daarnaast gaan we ook nog zetten dat we enkel willen dat het events zijn van deze site. We moeten ook de 'since' tijd meegeven. Anders dan gaat dit enkel de eerste keer werken. Dit ligt aan de manier waarop de SAFR API werkt. Ook als deze tijd in het verleden ligt dan gaat deze API niet wachten maar geeft meteen de nieuwste tijd terug.

```
API = f"http://{ip_address}:{event_server_port}/event/status/{device}?activity=added&since={since}"
```

Deze API kan dan doorgestuurd worden naar de 'get_data' definitie waardoor de call effectief wordt uitgevoerd. We gaan de response van de definitie opslaan in de locale variabele 'response'.

```
response = get_data(API)
```

Als de response een 204 error was dan gaan we dit ook verder doorsturen naar de hoofdloop. Deze zal hier dan rekening mee houden en opnieuw gaan wachten voor een event. We zouden ook hier opnieuw deze definitie aanroepen en dit zou werken, maar dan zal de definitie dieper in zichzelf gaan. Als er dan veel 204 errors na elkaar zijn dan gaat de definitie zeer ver zitten en dit zou onnodige resources gebruiken, terwijl op de manier hoe we dit nu doen gaat dit niet gebeuren.

```
if (response == "204 error"):
    return "204 error"
```

Dan gaan we kijken of dat de tijd (lastModDate) in de response zit. Als we dit niet doen en er is een probleem met de response waardoor de tijd er niet inzit dan zouden we een error krijgen. Daarna retourneren we deze tijd.

```
if ("lastModDate" in response.json()):
    return response.json()["lastModDate"]
```

Als de tijd niet in de response zit dan gaan we de nieuwste tijd ophalen en deze tijd retourneren.

```
else:
    new_time = event_service(0)
    return new_time
```

4.6 Definitie 'get_event'

De 'get_event' definitie is het hoofddeel van dit programma. Hierin gaan we de events ophalen en roepen we de definities die de checks doen aan. Om de definitie te kunnen gebruiken moeten de tijd en de wachttijd meegegeven worden. Ook gaan we ervoor zorgen dat we de globale variabelen 'checked', 'done' en 'last_event_time' kunnen gebruiken.


```
def get_event(last_mod_date, sleep):
    global checked
    global done
    global last_event_time
```

We gaan de API opbouwen. Hiervoor plaatsen we het ip adres en event server poort vooraan gesplitst door een `:`. De enige andere aanpassingen die we moeten doen aan de events API is het zetten van de minimum startdatum (`minStartDate`) en de tijd sinds waarna we willen checken (`sinceTime`).

```
API = f"http://{ip_address}:{event_server_port}/events?combineActiveEvents=false&minStartDate={last_mod_date}&rootEventsOnly=true&sinceTime={last_mod_date}&spanSources=false"
```

Voordat we de API effectief aanroepen moeten we eerst wachten voor de wachttijd. Anders gaan we niet kunnen kijken of er een begeleider bij is of niet.

We gaan hier altijd moeten wachten ook al is het event geen event dat we willen nakijken. We kunnen niet op voorhand al de API aanroepen en kijken of dit het geval is, als we voor de `sleep` de API aanroepen dan krijgen we geen event omdat SAFR hier een delay op heeft.

```
time.sleep(sleep)
```

Aangezien we de threads niet enkel in de hoofdloop willen verwijderen gaan we na de `sleep` een nieuwe thread starten om de threads te verwijderen. We doen dit na de `sleep` omdat we dan de grootste kans hebben dat de meeste threads klaar zijn en dus ook effectief verwijderd kunnen worden. Hier gaan we een thread aanmaken en om de `remove_threads` definitie te gebruiken. Daarna gaan we deze thread toevoegen aan de `threads` lijst en gaan we deze thread ook starten.

```
t = threading.Thread(target=remove_threads, name=f"remove")
threads.append(t)
t.start()
```

Na het wachten kunnen we de events die zijn voorgekomen na de laatste keer dat we events hebben opgehaald ophalen. Hiervoor roepen we de `get_data` definitie aan met de API call die we voor het wachten hebben opgesteld. En de we slagen de response op in de `response` variabele.

```
response = get_data(API)
```

We gaan een `try-except` blok gebruiken om de response naar json te zetten zodat we niet altijd `.json()` moeten zetten elke keer we de response willen aanroepen. Als de response geen json dan is er een probleem en gaan we dit loggen. We gaan dan ook `event_service` met de waarde 0 aanroepen om de nieuwste tijd te retourneren.

```
try:
    response = response.json()
except:
    event_time = datetime.now() - timedelta(seconds=wait_time)
    log_error(event_time, f"response has no json")
    return event_service(0)
```

Aangezien de events in het deel events van de response staat gaan we eerst kijken of dat deze tab effectief bestaat. Dit doen we aan de hand van de 'if' functie en als "events" niet in de response staat dan doen we een vroege return met de nieuwe tijd door de 'event_service' met waarde 0 aan te roepen.

```
if ("events" not in response):
    return event_service(0)
```

Om de events uit de response te halen die we niet hebben gaan we de last_event_time aanpassen. Dit moet gebeuren voordat we de events er uit halen die we niet nodig hebben, anders dan gaan we niet de laatste starttijd hebben wat problemen zal creëren bij het verwijderen van de checked lijst en dus ook het nakijken. We gaan proberen om de starttijd van het laatste event te nemen, de '-1' gaat het laatste event nemen. Als dit niet werkt dan nemen we de starttijd van het eerste event. En als dit ook niet werkt dan gaan we een error loggen aangezien het eerste en laatste event geen starttijd hebben of niet bestaan.

```
try:
    last_event_time = response['events'][-1]['startTime']
except:
    try:
        last_event_time = response['events'][0]['startTime']
    except:
        log_error(event_start_time, f"No startTime in first or last event.")
```

Daarna gaan we de 'handle_response' definitie aanroepen en de 'response' meegeven. Deze definitie gaat de events die we niet nodig hebben uit de response halen. We hoeven hier geen return voor te hebben aangezien de response die we meegeven een link is. Dus de veranderingen die gebeuren in de definitie gaan ook hier veranderen.

```
handle_response(response)
```

Dan gaan we de starttijd van het eerste event opslaan aan de hand van de starttijd. Ik gebruik de starttijd als tijd van het event omdat dit dan effectief gebeurd is en als de 'datetime' library gebruikt wordt kan er een delay op zitten dus dan zal het niet de tijd van het effectieve event zijn. We nemen de starttijd van het eerste event met door de [0]. Daarna gaan we dit omzetten in een effectieve tijd en dan zetten we dit naar de tijdzone van België die we hebben ingesteld. Als hier een probleem mee is dan zullen we wel gebruik maken van de 'datetime' library maar hier moeten we de wachttijd nog af trekken anders gaan we de zoiso niet de juiste tijd hebben.

```
try:
    event_start_time_response = response['events'][0]['startTime']
    start_time_utc = datetime.fromtimestamp(event_start_time_response / 1000.0, tz=timezone.utc)
    event_start_time = start_time_utc.astimezone(local_tz)
except:
    event_start_time = datetime.now() - timedelta(seconds=wait_time)
```

Als de lengte van de events 0 is dan is deze lijst leeg en dus was er geen event waarop we willen checken. Dus dan kunnen we gewoon de laatste tijd retourneren door 'event_service' van 0 aan te roepen.

```
if (len(response["events"]) == 0):
    return event_service(0)
```

Nu gaan we nakijken of de persoon die het event heeft getriggered deze deur mag gebruiken. Hiervoor gebruiken we de 'check_doors' definitie en we geven hiervoor het eerste event mee samen met de starttijd van het eerste event. Deze definitie gaat doorgeven of de persoon de deur mag gebruiken en de deuren die deze persoon mag gebruiken.

```
can_use_door, groups = check_doors(response["events"][0], event_start_time)
```

Nadat we hebben gekeken of dat de persoon de deur mag gebruiken gaan we kijken of er een begeleider is. Hiervoor gebruiken we de 'check_begeleider' definitie. Deze definitie gaat kijken of er een begeleider op dezelfde camera is gedetecteerd binnen de tijd. Hiervoor moeten we al de events meegeven alsook de starttijd van het eerste event. De definitie gaat teruggeven of er een begeleider mee gedetecteerd is door 'True' of 'False' te geven.

```
has_begeleider = check_begeleider(response["events"], event_start_time)
```

Nu we de checks hebben uitgevoerd kunnen we beslissen of er een alarm gestuurd moet worden of niet. Hiervoor dient de 'call_alarm' definitie. Voor deze definitie moeten we meegeven of er een begeleider was, of de deur gebruikt mag worden, de camera van het eerste event en de tijd van het eerste event. Deze definitie gaat teruggeven ofdat er een alarm was of niet zodat we dit kunnen gebruiken voor de logging.

```
alarm = call_alarm(has_begeleider, can_use_door, response["events"][0]["sourceId"], event_start_time)
```

De definitie 'check_begeleider' heeft de events die nog niet nagekeken konden worden toegevoegd aan de 'times' lijst. Maar om zeker te zijn dat het eerste event effectief eerst staat. Dus we gaan de 'sort()' functie gebruiken zodat de eerste tijd eerst staat.

```
times.sort()
```

Nu kunnen we de events processen. De nagekeken events staan in de 'done' lijst dus deze gaan we ook processen. Dit gebeurt in de 'process_done_list' definitie. Hiervoor geven we mee of er een begeleider was, of de deur gebruikt mag worden, of er een alarm was, een kopie van de 'done' lijst (als we gewoon 'done' zetten is dit een link naar de 'done' lijst wat niet gaat werken aangezien we deze leeg maken maar de ':' maakt een kopie en dit zal geen problemen geven met het loggen en de 'done' lijst leeg maken.), de deuren die de persoon kan gebruiken en de starttijd van het eerste event.

```
process_done_list(has_begeleider, can_use_door, alarm, done[:], groups, event_start_time)
```

Hierna kunnen we de 'done' lijst leeg maken door de 'clear' functie te gebruiken.

```
done.clear()
```

Als de 'times' lijst leeg is dan hoeven we geen ander event gaan nakijken. Dit doen we door te kijken of de lengte van de 'times' lijst 0 is. Als dit niet het geval is dan gaan we de nieuwe wachttijd berekenen zodat we even lang hebben gewacht als bij het eerste event. Hiervoor gebruiken we de 'calculate_time' definitie en we moeten hiervoor de eerste tijd en de response meegeven. Dan wordt er een nieuwe wachttijd teruggegeven. Nu kunnen we de 'get_event' definitie opnieuw aanroepen met de tijd van het eerste event in de 'times' lijst. We moeten hier 1 van aftrekken anders dan gaat dit event niet mee in de response zitten. Ook geven we de nieuwe wachttijd mee.

```
if (len(times) != 0):
    sleep = calculate_time(times[0],response)
    get_event(times[0] - 1, sleep )
```

Als er geen event meer is dat nog nagekeken wordt geven we de laatste tijd terug met de definitie 'event_service' met waarde 0 en dan gaan we uit deze definitie terug in de hoofdloop.

```
return event_service(0)
```

4.7 Definitie 'handle_response'

In deze definitie gaan de events die we niet nodig hebben uit de response gehaald worden. Dit zijn de events van mensen die geen bewoner of begeleider zijn, of events van dezelfde persoon op dezelfde deur binnen de 20 seconden.

```
def handle_response(response):
```

De nagekeken events die we kunnen gebruiken houden we tijdelijk bij in de 'good_checked_events' variabele. Ook gaan we de starttijd van het eerste event bijhouden in de variabele 'current_reference_time'.

```
good_checked_events = {}
current_reference_time = response["events"][0]["startTime"]
```

Dan gaan we over al de events in de 'checked' lijst lopen. Hiervoor gaan we het tijdsverschil berekenen en elke rotatie van de loop resetten we deze waarde.

```
for event_id, event_info in checked.items():
    time_diff = 0
```

Dan kijken we of dit event een eindtijd heeft in de 'if' functie. Als dit het geval is dan berekenen we het verschil door de eindtijd van de referentietijd af te trekken.

```
if ('endTime' in event_info):
    time_diff = current_reference_time - event_info['endTime']
```

Als dit event nog geen eindtijd heeft dan gaan we met het event id de informatie van dit event ophalen. Hiervoor moeten we in de API het event id bij "eventId" zetten. Daarna kunnen we 'get_data' aanroepen met deze API call.

```
else:
    API = f"http://{ip_address}:{event_server_port}/events?combineActiveEvents=false&eventId={event_id}&rootEventsOnly=true&spanSources=false"
    data = get_data(API)
```

Dan gaan we kijken of de response een eindtijd bij dit event heeft. Als dat zo is dan gaan we het tijdsverschil berekenen door deze eindtijd van de referentietijd af te trekken. Als dit niet het geval is dan is het event nog steeds bezig en willen we dit in 'checked' houden.

```
if ("endTime" in data.json()["events"][0]):
    time_diff = current_reference_time - data.json()["events"][0]['endTime']
```

Dan gaan we kijken of het tijdsverschil meer dan 20 seconden is. Als dit het geval is dan gaat het niet in de 'if' functie en willen we hier ook geen rekening mee houden. Maar als het event binnen de 20 seconden is dan gaan we dit toevoegen aan de 'good_checked_events' opslaan. Want we willen met deze events wel rekening houden zodat als er iemand dubbel gedetecteerd wordt dan gaan we dit niet 2 keer nakijken.

```
if (time_diff < 20000):
    good_checked_events[event_id] = event_info
```

Nu gaan we de relevante events ophalen. Dit doen we door over al de events te gaan en kijken of het event een "personType" heeft en als dit zo is of dit begeleider of bewoner is.

```
relevant_events = [
    event for event in response["events"]
    if "personType" in event and event["personType"] in ["Begeleider", "Bewoner"]
]
```

Nu gaan we ervoor zorgen dat er geen dubbele events zijn. Deze worden in 'best_events' bijgehouden.

```
best_events = {}
```

Hiervoor gaan we over al de relevante events gaan. Voor de 'key' gaan we de naam, deur en persoonstype bijhouden. We gaan de key gebruiken om te kijken of er een ander event is met dezelfde key, als dit het geval is dan betekend dit dat het een dubbel event is en hier willen we geen rekening mee houden. Het persoonstype is niet echt nodig om dit te laten werken maar kan handig zijn om testen uit te voeren.

```
for event in relevant_events:
    key = (event.get("name"), event.get("sourceId"), event.get("personType"))
```

In de 'if' functie gaan we kijken of deze 'key' al in 'best_events' zit en of het event id niet in 'checked' zit. Als geen van beide waar is dan zullen we deze 'key' toevoegen aan de 'best_events' lijst samen met het even waarbij deze 'key' hoort.

```
if key not in best_events and event.get("eventId") not in checked:
    best_events[key] = event
```

Als de 'key' al in deze lijst zit dan gaan we kijken of het event id nog niet in de 'checked' lijst zit.

```
else:
    if (event.get("eventId") not in checked):
```

We gaan een 'try-except' blok gebruiken omdat sommige begeleiders geen naam hebben en anders gaan er problemen zijn. In de 'try' gaan we het event toevoegen aan de 'checked' lijst. Als de eindtijd in het event zit dan gaan we de naam, starttijd, eindtijd, deur en persoonstype bijhouden.

```
try:
    if ('endTime' in event):
        checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "endTime": event["endTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

Als de eindtijd niet in het event zit dan gaan we de naam, starttijd, deur en persoonstype bijhouden.

```
else:
    checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

Als er geen naam is dan gaan we de starttijd, deur en persoonstype bijhouden. En als naam gaan we dit op "No Name" zetten.

```
except:
    if ("name" not in event):
        checked[event["eventId"]] = {"name": "No Name", "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

Nu moeten we nog de events nog vergelijken met de al nagekeken events. De effectieve events die we gaan gebruiken gaan we in de 'filtered_events' lijst houden.

```
filtered_events = []
```

We gaan over al de events in de 'best_events' gaan. Het nakijken gebeurt in een 'try-except' blok om dezelfde reden als hiervoor. Als er iemand geen naam heeft dan gaan er problemen zijn.

```
for event in best_events.values():
    try:
```

Als het event ID niet in 'good_checked_events' zit dan gaan we over al de events in 'good_checked_events' gaan. Als de naam, deur, persoonstype en starttijd niet overeen komen met deze waarden in de 'good_checked_events' lijst.

```
if (event["eventId"] not in good_checked_events and
    not any(checked_event.get("name") == event["name"] and
            checked_event.get("sourceId") == event["sourceId"] and
            checked_event.get("personType") == event["personType"] and
            int(checked_event.get("startTime")) < int(event["startTime"])
            for checked_event in good_checked_events.values())):
```

Dan gaan we kijken of dit event niet in de 'filtered_events' zit. Als dit niet het geval is dan gaan we dit event aan deze lijst toevoegen.

```
if (event not in filtered_events):
    filtered_events.append(event)
```

In het 'except' deel gaan we nakijken voor de mensen zonder naam.

```
except:
    if ("name" not in event):
```

Als het event ID niet in 'good_checked_events' zit dan gaan we over al de events in 'good_checked_events' gaan. Als de naam gelijk is aan "No Name", deur, persoonstype en starttijd niet overeen komen met deze waarden in de 'good_checked_events' lijst.

```
if (event["eventId"] not in good_checked_events and
    not any(
        checked_event.get("name") == "No Name" and
        checked_event.get("sourceId") == event["sourceId"] and
        checked_event.get("personType") == event["personType"] and
        int(checked_event.get("startTime")) < int(event["startTime"])
        for checked_event in good_checked_events.values())):
```

Dan gaan we kijken of dit event niet in de 'filtered_events' zit. Als dit niet het geval is dan gaan we dit event aan deze lijst toevoegen.

```
if (event not in filtered_events):
    filtered_events.append(event)
```

Nu dat we al de checks hebben gedaan dan gaan we de events in de response vervangen met de gefilterde events.

```
response['events'] = filtered_events
```

4.8 Definitie 'check_doors'

In de 'check_doors' definitie gaan we kijken of deze persoon de deur waarop die gedetecteerd is mag gebruiken.

```
def check_doors(response, event_start_time):
```

De deuren die gebruikt mogen worden zijn opgeslagen in de groepen. Als SAFR een nieuwe versie is dan staan de groepen gewoon in de event response, maar als het een oude SAFR applicatie is dan staat dit niet in de event response. Maar moeten we met de persoon id de groepen van deze persoon ophalen. Hiervoor gebruiken we de 'get_groups' definitie. Hiervoor moeten we de persoon id en de starttijd meegeven. En de definitie geeft de lijst met groepen terug.

```
groups = get_groups(response["personId"], event_start_time)
```

We zetten de variabele waarin we bijhouden of de deur gebruikt mag worden als basis op 'False'.

```
can_use_door = False
```

Als de camera niet in de response zit dan is er een probleem hiermee. Dus we gaan dit in de log file zetten. En we gaan de deuren en de 'can_use_door' die 'False' is vroeg retourneren.

```
if ("sourceId" not in response):
    log_error(event_start_time, f"There is no source (camera) in the first event of the response.")
    return can_use_door, groups
```

De deuren in de 'groups' staan in lower case en we gaan kijken of de deur waarop gedetecteerd is in deze lijst zit. Als deze in de lijst zit gaan we 'can_use_door' op 'True' zetten.

```
if (response["sourceId"].lower() in groups):
    can_use_door = True
```

En daarna gaan we 'can_use_door' en de groep retourneren.

```
return can_use_door, groups
```

4.9 Definitie 'get_groups'

De definitie 'get_groups' wordt gebruikt om de groepen van een persoon op te halen van de computer vision poort. Hiervoor moet de persoons id en de starttijd meegegeven worden.

```
def get_groups(person_id, event_start_time):
```

Om de groepen op te halen gaan we de informatie van een persoon ophalen van de computer vision server. Als de SAFR server nieuw is dan hoeft dit niet en kunnen we de groepen uit de event response halen, maar als het een oude versie is dan moeten we op deze manier de groepen ophalen. Hiervoor moeten we de persoon id op het einde van de call zetten. Dan kunnen we deze API aanroepen met de 'get_data' definitie.

```
API = f"http://{ip_address}:{computer_vision_port}/people/{person_id}"
response = get_data(API)
```

Er wordt nagekeken of de response een json heeft en zetten de response als deze json. Als er geen json is dan loggen we dit. We moeten geen vroege return doen aangezien de groep lijst dan gewoon leeg gelaten wordt.

```
try:
    response = response.json()
except:
    event_time = event_start_time
    log_error(event_time, f"The Computer Vision response has no json. Please check the personId: {person_id}.")
```

Er wordt de lijst 'groups' gecreëerd om de deuren in te houden.

```
groups = []
```

Om errors te voorkomen gaan we kijken of de tab 'people' in de response zit. Hier worden de deuren in opgeslagen.

```
if ("people" in response):
```


Dan gaan we kijken of er iets in de response staat door te checken of de lengte van de response groter is dan 0.

```
if(len(response["people"]) > 0):
```

Als de response niet leeg is gaan we kijken of de response "groups" bevat. Als dit niet het geval is dan zijn er geen deuren aangeduid. Maar als dit wel zo is dan gaan we de deuren aan 'groups' toevoegen. Hiervoor gaan we door al de waardes in de response gaan en we zetten deze naar lowercase zodat als er een hoofdletter verkeerd is dit geen invloed op de werking van het programma gaat hebben.

```
if("groups" in response["people"][0]):
    groups = [door.lower() for door in response["people"][0]["groups"]]
```

Als de response leeg was dan is er een probleem met de response. Dus deze error gaan we opslaan in de logs. We hoeven geen vroege return te doen omdat de 'groups' lijst leeg zal blijven.

```
else:
    event_time = event_start_time
    log_error(event_time, f"The people list is empty. Please check the personId: {person_id}.")
```

Als de tab 'people' niet in de response zit dan is er een probleem met de response. Dus deze error gaan we opslaan in de logs. We hoeven geen vroege return te doen omdat de 'groups' lijst leeg zal blijven.

```
else:
    event_time = event_start_time
    log_error(event_time, f"There is no people list. Please check the personId: {person_id}.")
```

Als we de deuren hebben opgehaald dan retourneren we de 'groups' lijst.

```
return groups
```

4.10 Definitie 'check_begeleider'

In de 'check_begeleider' gaan we kijken of er een begeleider bij deze events is. Deze definitie gaat andere definities aansturen die deze checks gaan doen.

```
def check_begeleider(response, event_start_time):
```

We gaan de globale variabelen 'checked', 'times' en 'done' gebruiken in deze definitie. Dus we moeten ook tegen het programma zeggen dat we deze willen gebruiken.

```
global checked
global times
global done
```

Eerst wordt de variabele 'begeleider_detected' waar we bij gaan houden of er een begeleider was default op 'False' gezet. Als er geen begeleider is dan is deze variabele 'False' en als er wel een begeleider is dan zal deze variabele 'True' zijn.

```
begeleider_detected = False
```

Hier wordt over elk event in de response gegaan. Er wordt gebruik gemaakt van 'enumerate' zodat we heel gemakkelijk aan de index van het event kunnen aangezien we de index nodig gaan hebben voor het nakijken. In de loop gaan we een definitie aanroepen die gaat nakijken en de response houden we bij in de 'begeleider_detected'. We moeten hier niet kijken of dit op een moment op 'True' gezet worden aangezien de definitie dit enkel op 'True' kan zetten en gaat het nooit achteraf op 'False' zetten.

```
for index, event in enumerate(response):
    begeleider_detected = process_event(index, event, response, begeleider_detected, event_start_time)
```

De 'process_event' definitie gaat ook de nagekeken events toevoegen aan de 'done' lijst. Maar deze gaat de events nog niet in de 'checked' lijst zetten. Hiervoor gaan we over de 'done' lijst gaan en maken we opnieuw gebruik van 'enumerate' zodat we makkelijk aan de index kunnen.

```
for index, event in enumerate(done):
```

Als er geen begeleider is dan is enkel het eerste event volledig nagekeken aangezien de rest niet dezelfde tijd heeft gekregen als het eerste event. Dus we willen enkel het eerste event in de 'checked' lijst zetten. Maar als er wel een begeleider is dan willen we alles in de 'checked' lijst zetten aangezien het niet uitmaakt dat we extra nakijken omdat er een begeleider is dus er gaat geen alarm gestuurd worden.

Dus eerst gaan we kijken of er geen begeleider is. Dan zal 'begeleider_detected' op 'False' staan;

```
if begeleider_detected == False:
```

Aangezien we enkel het eerste event willen gaan we dit nakijken met een 'if' functie. Het eerste event gaat een index van 0 hebben.

```
if (index == 0):
```

We willen ook niet dat er dubbele events in de 'checked' lijst dus we gaan eerst kijken of dit event al in de lijst staat aan de hand van het event id.

```
if (event["eventId"] not in checked):
```

We gaan dit event toevoegen in een 'try-except' blok aangezien het kan zijn dat er geen naam is ingevuld en dit zou voor problemen zorgen als deze blok er niet is. Dan gaan we nog kijken of er een eindtijd is of niet want als we een eindtijd proberen in te vullen maar het event is nog gaande dan gaan er ook problemen zijn. Als er een eindtijd is dan houden we het event id, de starttijd, de eindtijd, de deur en het persoonstype bij. Als er geen eindtijd is dan houden we het event id, de starttijd, de deur en het persoonstype bij.

```
try:
    if ('endTime' in event):
        checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "endTime": event["endTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
    else:
        checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

In het 'except' deel gaan we eerst kijken of het effectief de naam is dat mist. Dit doen we aan de hand van de 'if' functie door te checken of de variabele 'name' in dit event

zit. Dan gaan we het event id, de starttijd, de deur en het persoonstype toevoegen aan de checked lijst.

```
except:
    if ("name" not in event):
        checked[event["eventId"]] = {"name": "No Name", "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

Ook moeten we de starttijd van het eerste event uit de 'times' lijst halen als deze hier in zit. Als we dit niet doen dan gaan we een loop creëren aangezien dit event dan constant opnieuw als eerste wordt gedetecteerd. Dus als de starttijd in de 'times' lijst zit gaan we deze verwijderen met deze 'if' functie.

```
if (event["startTime"] in times):
    times.remove(event["startTime"])
```

Als we wel een begeleider hebben gedetecteerd dan willen we wel al de events in de 'checked' lijst bijhouden. We gaan deze events toevoegen in een 'try-except' blok aangezien het kan zijn dat er geen naam is ingevuld en dit zou voor problemen zorgen als deze blok er niet is. Dan gaan we nog kijken of er een eindtijd is of niet want als we een eindtijd proberen in te vullen maar het event is nog gaande dan gaan er ook problemen zijn. Als er een eindtijd is dan houden we het event id, de starttijd, de eindtijd, de deur en het persoonstype bij. Als er geen eindtijd is dan houden we het event id, de starttijd, de deur en het persoonstype bij.

```
else:
    try:
        if ('endTime' in event):
            checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "endTime": event["endTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
        else:
            checked[event["eventId"]] = {"name": event["name"], "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

In het 'except' deel gaan we eerst kijken of het effectief de naam is dat mist. Dit doen we aan de hand van de 'if' functie door te checken of de variabele 'name' in dit event zit. Dan gaan we het event id, de starttijd, de deur en het persoonstype toevoegen aan de checked lijst.

```
except:
    if ("name" not in event):
        checked[event["eventId"]] = {"name": "No Name", "startTime": event["startTime"], "sourceId": event["sourceId"], "personType": event["personType"]}
```

Aangezien deze events allemaal goed zijn en we willen deze hierna niet meer nakijken. Dus we moeten de tijd van de events uit 'times' halen. In de 'if' wordt nagekeken of de starttijd in 'times' zit en dan wordt deze starttijd verwijderd.

```
if (event["startTime"] in times):
    times.remove(event["startTime"])
```

Er is nu nagekeken of er een begeleider was dus we gaan 'begeleider_detected' retourneren.

```
return begeleider_detected
```

4.11 Definitie 'process_event'

In de definitie 'process_event' gaan we kijken of dit event het event van een begeleider is.

```
def process_event(index, event, response, begeleider_detected, event_start_time):
```

We gaan de globale variabelen 'checked', 'times' en 'done' gebruiken in deze definitie. Dus we moeten ook tegen het programma zeggen dat we deze willen gebruiken.

```
global checked
global times
global done
```

Om errors te voorkomen gaan we eerst kijken of al de velden die we willen gebruiken effectief bestaan. Door het event mee te geven met de definitie 'validate_event' gaan we dit event nakijken.

```
is_valid, error_message = validate_event(event)
```

Als het event niet goed (niet al de velden zijn aanwezig) is dan gaan we dit loggen en retourneren we de waarde die 'begeleider_detectie' al had. Als we de return niet op 'begeleider_detectie' maar op 'False' zetten en er is eerder een begeleider gedetecteerd en er is daarna een fout dan zal het programma 'begeleider_detectie' op 'False' zetten denken dat er geen begeleider was terwijl er eigenlijk wel een begeleider was.

```
if not is_valid:
    log_error(event_start_time, error_message)
    return begeleider_detected
```

Als de id van dit event al in 'checked' zit dan is er ergens een fout gebeurd. Maar normaal zou dit nooit mogen zijn. Dit staat er gewoon voor de zekerheid. Maar als het event al in 'checked' zit dan gaan we kijken of de starttijd in 'times' zit. En als dat het geval is dan verwijderen we de tijd.

```
if (event["eventId"] in checked):
    if (event["startTime"] in times):
        times.remove(event["startTime"])
```

Als het event niet in 'checked' zit gaan we kijken of er een event is dat dezelfde starttijd heeft als het eerste event. We moeten dan zien dat dit niet het eerste event is aan de hand van de index (index != 0) en kijken of de starttijd gelijk is aan de starttijd van het eerste event (event["startTime"] == response[0]["startTime"]). Als we geen rekening houden met een event op dezelfde tijd en als er geen begeleider is of dit event is op een andere deur dan gaan we ook in een oneindige loop van deze events nakijken geraken. Want dan gaat deze starttijd in 'times' komen maar aangezien deze hetzelfde is als het eerste event gaan we exact dezelfde response krijgen. Daarom gaan we de definitie 'event_at_same_time' aanroepen. Dit gaat het event checken met dezelfde starttijd als het eerste event. Hiervoor moeten we de response meegeven maar we willen het eerste event hier niet bij hebben anders hebben we hetzelfde probleem. Door 'response[index:]' te doen gaan we enkel de response vanaf dit event nemen.

```
elif index != 0 and event["startTime"] == response[0]["startTime"]:
    event_at_same_time(response[index:], event_start_time)
```

Als dit niet het geval is gaan we kijken of de deur van dit event hetzelfde is als de deur van het eerste event. We willen kijken of er een bewoner op dezelfde deur is dus daarvoor moeten we dit doen. De deur staat in het 'sourceId' van het event

```
elif event["sourceId"] == response[0]["sourceId"]:
```

Als dit dezelfde deur is dan gaan we kijken of deze persoon een begeleider is. Dit staat in het 'personType' van het event.

```
if event["personType"] == "Begeleider":
```

Als deze persoon een begeleider is dan kunnen we 'begeleider_detected' op 'True' zetten aangezien er een begeleider is.

```
begeleider_detected = True
```

Als deze persoon een begeleider is dan gaan we kijken of dit het eerste event is of niet. Als het niet het eerste event is dan gaan we de starttijd van dit event toevoegen aan 'times' zodat we ook de wachttijd na het begeleider event nakijken. Als het event wel het eerste event is dan voegen we dit event wel toe aan de 'done' lijst zodat dit event in 'checked' wordt toegevoegd.

Ik heb ervoor gekozen om dit event opnieuw te checken tot dit event het eerste event is zodat er de wachttijd voor en na het begeleider event gaat nagekeken worden. Maar als dit niet nodig is kan je deze 'if-else' vervangen met 'done.append(event)'.

```
if index != 0:
    if (event["startTime"] not in times):
        times.append(event["startTime"])
else:
    done.append(event)
```

Als de persoon geen begeleider is dan gaan we dit event toevoegen aan de 'done' lijst aangezien dit event nog wel op dezelfde deur was. We voegen dit aan de lijst toe zodat als er een begeleider is we dit event kunnen toevoegen aan 'checked' lijst.

```
else:
    done.append(event)
```

Als een event niet op dezelfde deur is gebeurd dan willen we dit later nog nakijken. Hiervoor gaan we de starttijd toevoegen aan de 'times' lijst. Maar eerst kijken we dat deze tijd al niet in de 'times' lijst zit.

```
elif (event["startTime"] not in times):
    times.append(event["startTime"])
```

Als laatste retourneren we 'begeleider_detected'. Als dit event niet het event van een begeleider was dan zal hier de vorige waarde geretourneerd worden maar als dit een begeleider is dan zal deze waarde 'True' zijn.

```
return begeleider_detected
```

4.12 Definitie 'validate_event'

De definitie 'validate_event' gaat kijken of een event de nodige velden heeft om voor een begeleider na te kijken.

```
def validate_event(event):
```

De velden die we nodig hebben om te kijken voor een begeleider zijn de persoonstype, de deur, de starttijd en het event id. Hiervoor maken we een lijst van deze velden en gaan we over ieder veld om te kijken of dit veld in event zit.

```
missing_fields = [field for field in ["personType", "sourceId", "startTime", "eventId"] if field not in event]
```

Als er een veld is dat mist dan gaan we retourneren dat dit event niet 'valid' is en de error message waarin de missende velden staan. Het loggen van de error gebeurt in de definitie die 'validate_event' aanroept. Als er geen event mist dan retourneren we dat het event 'valid' is zonder een error message.

```
if missing_fields:
    return False, f"Event is missing fields: {'', '.join(missing_fields)}."
return True, ""
```

4.13 Definitie 'event_at_same_time'

De definitie 'event_at_same_time' gaat de events nakijken wanneer er een event dezelfde starttijd heeft als het eerste event. Aangezien het eerste event uit de response is geknipt voor deze definitie kunnen we al de definities voor de checks aanroepen.

```
def event_at_same_time(response, event_start_time):
```

We gaan de 'done' lijst gebruiken in deze definitie dus we moeten dit eerst definiëren.

```
global done
```

Eerst gaan we kijken of de persoon van dit event de deur mag gebruiken. Hiervoor roepen we de 'check_doors' definitie aan wat gaat nakijken of deze persoon de deur waarop die gedetecteerd was mag gebruiken.

```
can_use_door, groups = check_doors(response, event_start_time)
```

Nu gaan we kijken of er een begeleider bij deze events is. Hiervoor roepen we de definitie 'check_begeleider' aan. Deze definitie gaat kijken of er een begeleider op dezelfde deur is gedetecteerd.

```
begeleider = check_begeleider(response, event_start_time)
```

Daarna kunnen we het alarm laten afgaan door 'call_alarm' aan te roepen. Deze definitie gaat kijken of er een alarm aangestuurd moet worden en dit gaat ook een IQ Messenger alarm aansturen wanneer nodig.

```
alarm = call_alarm(begeleider, can_use_door, event_start_time)
```

Als het alarm aangestuurd is dan gaan we de events processen met de 'process_done_list' definitie. Deze definitie gaat de events die nagekeken zijn in de log file opslaan. Het meesturen van de 'done' lijst moet met 'done[:]' dit gaat een kopie van de 'done' lijst maken. Anders is dit een referentie naar de lijst en aangezien we deze lijst leeg moeten maken gaat dit een probleem zijn.

```
process_done_list(begeleider, can_use_door, alarm, done[:], groups, event_start_time)
```

Hier maken we de 'done' lijst leeg aangezien we hierna terug gaan naar de 'check_begeleider' definitie waar deze lijst ook gebruikt wordt. Maar dan moet dit leeg zijn.

```
done.clear()
```

4.14 Definitie 'call_alarm'

De definitie 'call_alarm' kijken of er een alarm moet aangestuurd worden. Gaat aansturen dat we de afbeelding van dit event ophalen en ook naar IQ Messenger het alarm aansturen.

```
def call_alarm(has_begeleider, can_use_door, source, event_start_time):
```

Het alarm dat we hebben gaan we opslaan als een string aangezien dit later makkelijker is voor de logging.

```
alarm = ""
```

Als er een begeleider is of de deur mag gebruikt worden dan zal het alarm op "no alarm" gezet worden en er wordt geen alarm naar IQ Messenger gestuurd.

```
if(has_begeleider == True or can_use_door == True):
    alarm = "no alarm"
```

Als de deur niet gebruikt mag worden en er is geen begeleider dan gaat het alarm op "alarm" gezet worden. Ook zal dan de afbeelding opgehaald worden en er wordt een alarm naar IQ Messenger gestuurd.

```
else:
    alarm = "alarm"
```

We gaan kijken of de naam in het event zit voor de zekerheid anders kan dit voor errors zorgen als voor een bewoner geen naam ingevuld is.

Eerst gaan we de afbeelding van dit event ophalen. Dit gebeurt in een andere file zodat de applicatie niet te onordelijk wordt. Deze file is 'GetEventImages' en hiervoor moeten we de 'create_image' definitie aanroepen. Aangezien in deze file de config file niet ingelezen wordt moeten we al deze informatie meegeven aan de definitie.

Daarna gaan we een request sturen naar de IQ Messenger server met de deur, de naam van de persoon en de tijd van het event

```
if "name" in done[0]:
    GetEventImages.create_image(user_id, password, ip_address, object_service_port, wait_time, done[0]['eventId'], source.lower(), done[0]['name'], image_location, logo_transparent_location, event_start_time)
    requests.get("http://(iq_messenger_username):(iq_messenger_password)@((iq_messenger_ip_address))/IQHttpCommons/service/httplo/device={source}&message={done[0]['name']} {event_start_time.strftime('%H:%M:%S')}" , verify=False)
```

Als er geen naam is ingevuld dan gaan we in plaats van de naam de tekst "geen naam" meegeven, maar voor de rest is dit hetzelfde als in de 'if' er boven.

```
else:
    GetEventImages.create_image(user_id, password, ip_address, object_service_port, wait_time, done[0]['eventId'], source.lower(), "geen naam", image_location, logo_transparent_location, event_start_time)
    requests.get("http://(iq_messenger_username):(iq_messenger_password)@(iq_messenger_ip_address)/IQHttpCommons/service/httplo?device={source}&message=geen naam {event_start_time.strftime('%H:%M:%S')}", verify=False)
```

Nadat het alarm wel of niet is aangestuurd kunnen we de tekst van het alarm retourneren.

```
return alarm
```

4.15 Definitie 'process_done_list'

De definitie 'process_done_list' gaat aansturen dat de events in de 'done' lijst gelogd gaan worden.

```
def process_done_list(has_begeleider, can_use_door, alarm, done, groups, event_start_time):
```

We gaan kijken of de 'done' lijst leeg is. Als de lengte van 'done' groter is dan 0 dan is de 'done' lijst niet leeg en kunnen we events gaan loggen. Hiervoor starten we een nieuwe thread en roepen we de 'log_line' definitie aan. We gaan deze thread aan de 'threads' lijst toevoegen en dan kunnen we ook deze thread starten.

```
if len(done) != 0:
    t = threading.Thread(target=log_line, args=(has_begeleider, can_use_door, alarm, done[:], groups, event_start_time), name="log")
    threads.append(t)
    t.start()
```

Als de 'done' lijst leeg is dan gaan we dit als een error loggen aangezien dit betekend dat er geen event nagekeken is wat niet de bedoeling is.

```
else:
    event_time = event_start_time
    log_error(event_time, "Done list is empty.")
```

4.16 Definitie 'calculate_time'

In de definitie 'calculate_time' gaan we het tijdsverschil tussen het eerste event en het volgende event dat we willen nakijken berekenen.

```
def calculate_time(time, response):
```

De variabele 'time' heft de tijd van het volgende event dat we willen nakijken. Hier trekken we de starttijd van het eerste event af. Dit geeft het tijdsverschil tussen het eerste en volgende event zodat we dezelfde wachttijd hebben voor het eerste event en het volgende event. Daarna delen we het verschil ook nog door 1000 aangezien 1 seconde een waarde van 1000 heeft.

```
sleep = ((time-response["events"][0]["startTime"])/1000)
```

Als deze delay groter is dan de wachttijd of als dit minder dan 0 is dan is het eerste event ouder of nieuwer dus dan zetten we de wachttijd op 0.


```
if (sleep > wait_time or sleep < 0):
    sleep = 0
```

Nu kunnen we de nieuwe wachttijd retourneren.

```
return sleep
```

4.17 Definitie 'clear_checked_list'

In de definitie 'clear_checked_list' gaan we de events die we niet meer nodig gaan hebben uit de 'checked' lijst halen.

```
def clear_checked_list():
```

Hier gaan we gebruik maken van de globale variabele 'last_event_time' dus we moeten deze initialiseren.

```
global last_event_time
```

Dan gaan we de meest recente starttijd ophalen door over de 'checked' lijst te gaan en de max starttijd te nemen.

```
most_recent_start = max(
    event_info['startTime']
    for event_id, event_info in checked.items()
)
```

Als de meest recente starttijd groter is dan de 'last_event_time' dan gaan we deze tijd als 'last_event_time' zetten.

```
if (most_recent_start > last_event_time):
    last_event_time = most_recent_start
```

Nu gaan we over de events van de 'checked' lijst gaan om de events die we niet meer nodig hebben uit de lijst kunnen halen. De 'list' functie zorgt ervoor dat er een kopie van de lijst gebruikt wordt anders gaan er problemen zijn als er events verwijderd worden. Dan gaan we het tijdsverschil tussen dit event en het laatste event berekenen. Dit doen we door de starttijd van dit event af te trekken van de 'last_event_time'.

```
for event_id, event_info in list(checked.items()):
    time_diff = last_event_time - event_info['startTime']
```

Als het tijdsverschil groter is dan 60000 (1 minuut) dan willen we kijken of het event nog gaande is.

```
if time_diff > 60000:
```

Dan gaan we kijken of de eindtijd al in de 'checked' lijst zit. Als dit het geval is dan is het event gedaan en kunnen we dit event verwijderen.

```
if 'endTime' in event_info and event_info['endTime'] <= last_event_time:
    del checked[event_id]
```

Als de eindtijd niet in de 'checked' lijst zit dan gaan we dit event aanroepen met de API aangezien het kan zijn dat het event geëindigd is maar de eindtijd was nog niet in 'checked' gezet. Hiervoor moeten we de SAFR API aanroepen met het event id.

```
else:
    API = f"http://{ip_address}:{event_server_port}/events?combineActiveEvents=false&eventId={event_id}&rootEventsOnly=true&spanSources=false"
    data = get_data(API)
```

Dan gaan we kijken of het event een eindtijd heeft. Als dit event een eindtijd heeft dan is dit geëindigd en kunnen we dit event verwijderen.

```
if ("endTime" in data.json()["events"][0]):
    del checked[event_id]
```

4.18 Definitie 'remove_threads'

In de definitie 'remove_threads' gaan we de threads die klaar zijn verwijderen. Als we dit niet doen gaan er heel veel openstaande threads zijn en deze verbruiken niet veel maar toch een aantal resources maar over een lange tijd loopt dit op. Hiervoor moeten we de threads die klaar zijn joinen met de hoofdthread.

```
def remove_threads():
```

We houden al de events in de 'threads' lijst. Dus we gaan over deze lijst lopen om de threads te joinen. Aangezien we dingen in de lijst gaan verwijderen gebruiken we de 'list' functie wat een kopie van de lijst gaat gebruiken.

```
for t in list(threads):
```

Eerst gaan we kijken of de thread nog actief is. Dit kunnen we doen met de 'is_alive' functie. Als de thread klaar is dan zal dit 'False' zijn. Daarna kunnen we de thread joinen met de 'join' functie en verwijderen deze thread van de 'threads' lijst.

```
if (t.is_alive() == False):
    t.join()
    threads.remove(t)
```

5 AFBEELDING EVENT OPHALEN

In dit project wordt het ophalen van afbeeldingen van een evenement niet rechtstreeks vanuit het hoofdprogramma uitgevoerd. In plaats daarvan is deze taak gedelegeerd aan de Python-file 'GetEventImages'. Dit script bevat alle noodzakelijke functionaliteiten om afbeeldingen van een evenement op te halen, te bewerken en op te slaan.

Het hoofdprogramma zal gebruik maken van een specifieke definitie binnen 'GetEventImages' om de afbeeldingen te verwerken. Deze aanpak zorgt voor een duidelijke scheiding van verantwoordelijkheden en maakt het hoofdprogramma overzichtelijker en eenvoudiger te onderhouden. Bovendien biedt het de flexibiliteit om de afbeeldingsverwerkingslogica onafhankelijk te ontwikkelen en bij te werken zonder het hoofdprogramma te beïnvloeden.

5.1 Definitie 'create_image'

De 'create_image' definitie is de hoofdfunctie van het ophalen van de afbeeldingen. Deze definitie gaat al de andere definities die we gebruiken om de afbeelding op te halen, aan te passen en op te slagen. Aangezien deze definitie aangestuurd gaat worden vanuit het hoofdprogramma gaat deze file niet op voorhand gerund worden waardoor we geen initialisatie kunnen doen van de config file. We zouden dit wel in een definitie kunnen doen maar aangezien ik niet elk event de volledige config file wil inlezen heb ik ervoor gekozen om al de nodige configuraties mee te geven bij het aanroepen van de definitie. We hebben de informatie van de config file ook effectief nodig aangezien we API calls moeten maken naar de SAFR server om de afbeelding van het event en van de persoon moeten ophalen.

```
def create_image(user_id, password, ip_address, object_service_port, wait_time, event_id, source_id, name, image_location, logo_transparent_location, event_start_time):
```

Om de afbeelding van het event en de persoon op te halen hebben we het id van het event nodig. Dit is meegegeven bij het aanroepen van de definitie. Het id is in 'utf-8' gecodeerd maar voor de API call moet dit in 'base64' gecodeerd zijn dus we moeten dit eerst omzetten.

Eerst gaan we de meegegeven waarde van het event id in een locale variabelen zetten. Dan gaan we de string van het id omzetten naar bytes aangezien het omzetten bytes gebruikt en geen string. Dan encoderen we deze bytes naar base64 door 'base64.b64encode(id_bytes)' te doen. Maar aangezien we voor de API call een string moeten hebben moeten we dit eerst nog decoden met 'utf-8' om deze 'base64' bytes om te zetten naar een string.

```
id_str = event_id
id_bytes = id_str.encode('utf-8')
base64_encoded = base64.b64encode(id_bytes)
base64_event_string = base64_encoded.decode('utf-8')
```

Nu hebben we de correct gecodeerde event id dus dan kunnen we de API calls maken om de afbeeldingen op te halen. Hiervoor gebruiken we de definitie 'get_data'. Deze definitie gaat de API call maken en retourneert de response van de API call die gemaakt is. Om de definitie te gebruiken moeten we de API call die we willen maken, de gebruikersnaam van de SAFR server en het wachtwoord meegeven.

De API call met "sceneThumb" in gaat de afbeelding van het event ophalen.

```
scene_response = get_data(f"http://{ip_address}:{object_service_port}/obj/{base64_event_string}/sceneThumb", user_id, password)
```

De API call met "face" in gaat de afbeelding van de gedetecteerde persoon ophalen.

```
face_response = get_data(f"http://{ip_address}:{object_service_port}/obj/{base64_event_string}/face", user_id, password)
```

Om de afbeeldingen te gebruiken moeten we deze eerst inladen als afbeelding. We gebruiken de definitie 'load_image_from_response' om deze afbeelding in te laden. De hoofdafbeelding is de afbeelding van het event dus we gaan deze eerst inladen en opslaan in de 'scene_image' variabele.

```
scene_image = load_image_from_response(scene_response)
```

Aangezien we de afbeelding van de gedetecteerde persoon op de afbeelding van het event willen plaatsen moeten we deze ook als afbeelding inladen en opslaan in de 'face_image' variabele.

```
face_image = load_image_from_response(face_response)
```

We gaan ook het DeltaTechnics logo toevoegen aan deze afbeelding. Dit gaat de afbeelding zijn met een transparante achtergrond. Eerst stoppen we het pad dat we meegekregen hebben van de definitie in een locale variabele.

```
logo_path = logo_transparent_location
```

Dan moeten we de afbeelding ook nog ophalen. Hiervoor wordt de definitie 'load_logo' gebruikt. Dit moet in een andere definitie gebeuren dan het inladen van de event afbeeldingen aangezien het logo een pad is dat we moeten ophalen en geen doorgestuurde afbeelding. Ook zal er in deze definitie het logo meteen op de juiste grote gezet worden.

```
logo_image = load_logo(logo_path)
```

Nu hebben we al de afbeeldingen die we nodig hebben om de volledige afbeelding te maken. Maar eerst gaan we kijken of de event afbeelding en gezicht afbeelding effectief bestaan zodat we niet tegen errors lopen als er een probleem was met deze afbeeldingen op te halen.

```
if scene_image and face_image:
```

Als de afbeeldingen bestaan kunnen we de gezichtsafbeelding boven op de afbeelding van de event afbeelding. Dit doen we aan de hand van de 'combine_images' definitie waar we de event afbeelding en gezicht afbeelding met meegeven.

```
combined_image = combine_images(scene_image, face_image)
```

Nu de event afbeelding en gezicht afbeelding zijn samengevoegd gaan we het logo toevoegen. Maar eerst gaan we weer kijken of we effectief een afbeelding hebben die we kunnen gebruiken.

```
if logo_image:
```

Als de afbeelding van het logo beschikbaar is, berekenen we eerst de positie waar het logo moet worden geplaatst. We willen het logo linksonder in de afbeelding positioneren, waarbij de coördinaten beginnen te tellen vanaf de linkerbovenhoek van de afbeelding.

De eerste variabele in de positie (10) geeft aan hoe ver het logo van de linkerrand van de afbeelding moet worden geplaatst. Dit betekent dat het logo 10 pixels vanaf de linkerrand komt te staan.

De tweede variabele in de positie wordt berekend met de uitdrukking 'combined_image.size[1] - logo_image.size[1] - 10'. Deze uitdrukking bepaalt de verticale positie van het logo als volgt:

- `'combined_image.size[1]'` is de totale hoogte van de gecombineerde afbeelding.
- `'logo_image.size[1]'` is de hoogte van het logo.
- Door de hoogte van het logo af te trekken van de totale hoogte van de gecombineerde afbeelding, bepalen we de positie waar de bovenkant van het logo moet beginnen om ervoor te zorgen dat de onderkant van het logo uitgelijnd is met de onderkant van de gecombineerde afbeelding.
- Vervolgens trekken we nog eens 10 pixels af om een kleine marge te creëren tussen de onderkant van het logo en de onderkant van de gecombineerde afbeelding.

Dus, de tweede variabele zorgt ervoor dat het logo 10 pixels boven de onderkant van de gecombineerde afbeelding wordt geplaatst. Hierdoor wordt het logo 10 pixels van de linkerrand en 10 pixels boven de onderkant van de gecombineerde afbeelding gepositioneerd.

```
logo_position = (10, combined_image.size[1] - logo_image.size[1] - 10)
```

Nu we de positie hebben kunnen we het logo toevoegen aan de afbeelding door de volledige afbeelding, het logo en de positie mee te geven in de `'add_logo'` definitie. Deze definitie gaat het logo op de afbeelding plaatsen op de positie die we net hebben berekend en gaat de nieuwe afbeelding retourneren.

```
combined_image = add_logo(combined_image, logo_image, logo_position)
```

We willen ook het tijdstip van het event op de afbeelding plaatsen. De tijd van het event is meegegeven met de definitie dus we stoppen deze waarde eerst in een locale variabele. Aangezien de tijd die meegegeven is niet in een leesbaar formaat staat gaan we eerst dit op een leesbare manier plaatsen. Dit gaat hier komen te staan als jaar-maand-dag uur-minuten-seconden.

```
current_time = event_start_time
formatted_time = current_time.strftime("%Y-%m-%d %H:%M:%S")
```

Nu kunnen we de tijd en naam van de persoon op de afbeelding plaatsen door de `'add_time_stamp'` definitie aan te roepen. Hiervoor moeten we de huidige afbeelding meegeven, alsook de tijd, naam, font grote en kleur van de tekst. De huidige grote van de tekst is 20 dus als je de tekst groter of kleiner wil kan je deze waarde aanpassen. Ik heb ook gekozen om de tekst in het zwart te doen maar als dit in het wit zou moeten dan kan je de waarde `'black'` naar `'white'` zetten.

```
timestamped_image = add_time_stamp(combined_image, formatted_time, name, 20, 'black')
```

Dan gaan we de volledige afbeelding converteren naar `'RGB'` zodat we zeker zijn dat de kleuren van de afbeelding normaal zijn.

```
timestamped_image = timestamped_image.convert('RGB')
```

Nu gaan we de afbeelding opslaan op de locatie die ingesteld is om de event afbeeldingen op te slaan met de naam van de deur. Ook willen we deze afbeelding als jpg opslaan dus we moeten na de naam van de deur `'.jpg'` zetten.

```
timestamped_image.save(f"{image_location}/{source_id}.jpg")
```

Als de afbeelding dan opgeslagen is gaan we een tekst laten zien dat de afbeelding is opgeslagen.

```
print("Image saved successfully.")
```

Als de event afbeelding of de gezichtsafbeelding niet beschikbaar was dan gaan we een bericht laten zien dat er een probleem was met het verkrijgen van de afbeeldingen.

```
else:
    print("Failed to fetch images or error occurred.")
```

5.2 Definitie 'get_data'

De definitie 'get_data' gaat de API call maken die is meegegeven bij het aanroepen van deze definitie. En als de call is gelukt dan retourneren we de response

```
def get_data(api, user_id, password):
```

Om de API call te maken gebruiken we de 'requests' library. Als eerste variabelen zetten we de API call die meegegeven wordt aan de definitie. Voor een SAFR API call te maken moeten we ook de inlog gegevens en de directory in de headers zetten.

```
response = requests.get(api, headers={"X-RPC-AUTHORIZATION": f"{user_id}:{password}", "X-RPC-DIRECTORY": "main"})
```

Als de call gelukt is dan zal er een response code van 200 zijn. Dan gaan we laten zien dat de call gelukt is en de response van de API retourneren.

```
if response.status_code == 200:
    print("Successfully fetched the data")
    return response
```

Als de call te lang duurt dan is er een 204 status code. Dan gaat er ook geen response zijn maar is er een probleem met de call dus we gaan hier een tekst voor laten zien dat er een probleem is en dan retourneren we 'None' om ervoor te zorgen dat de response leeg is zodat we niet tegen errors in de applicatie zitten.

```
elif response.status_code == 204:
    print("204 error: No data yet, trying again.")
    return None
```

Als er een andere error is met de call dan gaan we deze error printen. Dan retourneren we 'None' om ervoor te zorgen dat de response leeg is zodat we niet tegen errors in de applicatie zitten.

```
else:
    print(f"There's a {response.status_code} error with your request")
    return None
```

5.3 Definitie 'load_image_from_response'

In de definitie 'load_image_from_response' gaan we de afbeelding die we van de API hebben opgehaald inladen als een image zodat we deze afbeelding kunnen gebruiken om aanpassingen te maken.

```
def load_image_from_response(response):
```

Eerst gaan we kijken of de response effectief bestaat. Als deze bestaat gaan we de 'Image' library en de 'io' library gebruiken om de afbeelding in te laden. De 'BytesIO' gaat de response omzetten in bytes zodat we deze met de 'Image.open' als Image kunnen inladen zodat we de aanpassingen ook via de 'Image' library kunnen maken. Als de response niet bestaat (er was een error met de API call) dan gaan we 'None' retourneren zodat de afbeelding leeg is en geen errors gaat veroorzaken.

```
if response:
    return Image.open(BytesIO(response.content))
return None
```

5.4 Definitie 'load_logo'

In de definitie 'load_logo' gaan we het logo als afbeelding inladen. Ook gaan we de grote van de afbeelding aanpassen zodat deze juist op de afbeelding past.

```
def load_logo(logo_path, target_width=160):
```

We gaan al deze aanpassingen in een 'try-except' blok zodat we geen errors krijgen. Om het logo als afbeelding in te kunnen laden kunnen we gewoon 'Image.open' gebruiken omdat dit een pad is naar de afbeelding.

```
try:
    logo_image = Image.open(logo_path)
```

Om de afmetingen van een logo-afbeelding aan te passen terwijl de verhoudingen behouden blijven, volgen we deze stappen:

We halen eerst de oorspronkelijke breedte en hoogte van de logo-afbeelding op met 'original_width, original_height = logo_image.size'. Dit geeft ons de breedte ('original_width') en de hoogte ('original_height') van de afbeelding.

Vervolgens berekenen we de aspect ratio (verhoudingsverhouding) van de afbeelding met 'aspect_ratio = original_height / original_width'. De aspect ratio is de verhouding tussen de hoogte en de breedte van de afbeelding. Dit zorgt ervoor dat de afbeelding niet vervormd raakt wanneer we de grootte aanpassen.

Om de nieuwe hoogte van de afbeelding te bepalen terwijl de breedte wordt aangepast naar een bepaalde doelbreedte ('target_width'), gebruiken we de aspect ratio. Dit doen we met 'new_height = int(aspect_ratio * target_width)'. Door de aspect ratio te vermenigvuldigen met de nieuwe breedte ('target_width'), krijgen we de nieuwe hoogte ('new_height') die overeenkomt met de verhoudingen van de originele afbeelding.


```
original_width, original_height = logo_image.size
aspect_ratio = original_height / original_width
new_height = int(aspect_ratio * target_width)
```

Nu kunnen we de afbeelding de nieuwe afmetingen geven. Hiervoor geven we eerst de nieuwe hoogte en breedte mee. Ook moeten we 'Image.Resampling.NEAREST' hier bij zetten zodat als onze berekening op een kommagetal kwam we het dichtstbijzijnde even getal nemen aangezien we geen kommagetal voor pixels kunnen hebben.

```
logo_image = logo_image.resize((target_width, new_height), Image.Resampling.NEAREST)
```

Nu kunnen we het logo retourneren.

```
return logo_image
```

Als we het logo niet kunnen openen dan gaan we een error message printen en retourneren we 'None' zodat we later niet met errors komen te zitten.

```
except IOError:
    print("Unable to load the logo image.")
    return None
```

5.5 Definitie 'combine_images'

In de definitie 'combine_images' gaan we de afbeelding van de persoon ('overlay') op de afbeelding van het event ('background') zetten.

```
def combine_images(background, overlay):
```

In deze code passen we de afmetingen van een overlay-afbeelding aan zodat deze een nieuwe breedte van 160 pixels krijgt, terwijl de oorspronkelijke verhoudingen behouden blijven. Hier is hoe we dat doen:

We beginnen met het definiëren van de nieuwe breedte voor de overlay-afbeelding met 'new_width = 160'. Dit betekent dat we willen dat de overlay-afbeelding een breedte van 160 pixels heeft.

Vervolgens berekenen we de schaalverhouding die nodig is om de breedte van de overlay-afbeelding aan te passen naar de nieuwe breedte. Dit doen we met 'ratio = new_width / overlay.size[0]'. Hierbij is 'overlay.size[0]' de oorspronkelijke breedte van de overlay-afbeelding. De variabele 'ratio' vertegenwoordigt de factor waarmee we de afmetingen van de afbeelding moeten vermenigvuldigen om de nieuwe breedte te krijgen.

Om de nieuwe hoogte van de overlay-afbeelding te bepalen terwijl de oorspronkelijke verhoudingen behouden blijven, gebruiken we de schaalverhouding die we zojuist hebben berekend. Dit doen we met 'new_height = int(overlay.size[1] * ratio)'. Hierbij is 'overlay.size[1]' de oorspronkelijke hoogte van de overlay-afbeelding. Door de oorspronkelijke hoogte te vermenigvuldigen met de schaalverhouding, krijgen we de nieuwe hoogte die overeenkomt met de nieuwe breedte van 160 pixels.

```
new_width = 160
ratio = new_width / overlay.size[0]
new_height = int(overlay.size[1] * ratio)
```

Nu kunnen we de breedte en hoogte toepassen aan de gezichtsafbeelding door de 'resize' functie te gebruiken.

```
overlay = overlay.resize((new_width, new_height))
```

In deze code passen we de kleurmodus van een overlay-afbeelding aan en maken we een masker aan op basis van de transparantie (alpha-kanaal) van de afbeelding. Dit zijn de stappen:

De eerste regel 'if overlay.mode != 'RGBA':' controleert of de kleurmodus van de 'overlay'-afbeelding niet al 'RGBA' is. De 'RGBA'-kleurmodus staat voor rood (R), groen (G), blauw (B) en alpha (A), waarbij het alpha-kanaal de transparantie van de afbeelding aangeeft.

Als de kleurmodus van de 'overlay' niet 'RGBA' is, wordt deze geconverteerd met 'overlay = overlay.convert('RGBA')'. Dit zorgt ervoor dat de afbeelding een alpha-kanaal heeft, wat nodig is voor het volgende deel van de code. Het alpha-kanaal is belangrijk voor het werken met transparantie.

De regel 'mask = overlay.split()[3]' splitst de 'overlay'-afbeelding in zijn afzonderlijke kleurkanalen (R, G, B, en A). De 'split()'-methode retourneert een tuple van de afzonderlijke kanalen. Door '[3]' te gebruiken, selecteren we het vierde kanaal, wat het alpha-kanaal is. Dit alpha-kanaal wordt vervolgens toegewezen aan de variabele 'mask'.

```
if overlay.mode != 'RGBA':
    overlay = overlay.convert('RGBA')
mask = overlay.split()[3]
```

In deze regel berekenen we de positie waar een overlay-afbeelding op een achtergrondaafbeelding moet worden geplaatst. Het doel is om de overlay-afbeelding rechtsonder op de achtergrondaafbeelding te positioneren, met een marge van 10 pixels vanaf de rechter- en onderrand.

1. Horizontale Positie (x-coördinaat):
 - 'background.size[0]': Dit is de breedte van de achtergrondaafbeelding.
 - 'overlay.size[0]': Dit is de breedte van de overlay-afbeelding.
 - 'background.size[0] - overlay.size[0]': Door de breedte van de overlay af te trekken van de breedte van de achtergrond, krijgen we de x-coördinaat waar de linkerzijde van de overlay moet beginnen om rechts uitgelijnd te zijn.
 - 'background.size[0] - overlay.size[0] - 10': Door 10 pixels af te trekken, verschuiven we de overlay 10 pixels naar links, zodat er een marge van 10 pixels is tussen de overlay en de rechterrand van de achtergrond.
2. Verticale Positie (y-coördinaat):
 - 'background.size[1]': Dit is de hoogte van de achtergrondaafbeelding.
 - 'overlay.size[1]': Dit is de hoogte van de overlay-afbeelding.
 - 'background.size[1] - overlay.size[1]': Door de hoogte van de overlay af te trekken van de hoogte van de achtergrond, krijgen we de y-coördinaat waar de bovenkant van de overlay moet beginnen om onderaan uitgelijnd te zijn.

- `'background.size[1] - overlay.size[1] - 10'`: Door 10 pixels af te trekken, verschuiven we de overlay 10 pixels naar boven, zodat er een marge van 10 pixels is tussen de overlay en de onderrand van de achtergrond.

```
position = (background.size[0] - overlay.size[0] - 10, background.size[1] - overlay.size[1] - 10)
```

Dan gaan we de overlay op de event afbeelding plaatsen op de positie die we net hebben berekend. Als de overlay hier op is geplaatst dan kunnen we deze afbeelding retourneren.

```
background.paste(overlay, position, mask)
return background
```

5.6 Definitie 'add_logo'

In de definitie 'add_logo' gaan we het logo toevoegen aan de afbeelding. Ook is de positie al berekend en wordt meegegeven aan de definitie.

```
def add_logo(image, logo, position):
```

In deze code passen we de kleurmodus van het logo aan en maken we een masker aan op basis van de transparantie (alpha-kanaal) van de afbeelding. Dit zijn de stappen:

De eerste regel `'if logo.mode != 'RGBA':'` controleert of de kleurmodus van het 'logo'-afbeelding niet al 'RGBA' is. De 'RGBA'-kleurmodus staat voor rood (R), groen (G), blauw (B) en alpha (A), waarbij het alpha-kanaal de transparantie van de afbeelding aangeeft.

Als de kleurmodus van het 'logo' niet 'RGBA' is, wordt deze geconverteerd met `'logo = logo.convert('RGBA')'`. Dit zorgt ervoor dat de afbeelding een alpha-kanaal heeft, wat nodig is voor het volgende deel van de code. Het alpha-kanaal is belangrijk voor het werken met transparantie.

De regel `'mask = logo.split()[3]'` splitst de 'logo'-afbeelding in zijn afzonderlijke kleurkanalen (R, G, B, en A). De `'split()'`-methode retourneert een tuple van de afzonderlijke kanalen. Door `'[3]'` te gebruiken, selecteren we het vierde kanaal, wat het alpha-kanaal is. Dit alpha-kanaal wordt vervolgens toegewezen aan de variabele 'mask'.

```
if logo.mode != 'RGBA':
    logo = logo.convert('RGBA')
logo_mask = logo.split()[3]
```

Dan gaan we het logo op de event afbeelding plaatsen op de positie die is meegegeven bij het aanroepen van deze definitie. Als het logo hier op is geplaatst dan kunnen we deze afbeelding retourneren.

```
image.paste(logo, position, logo_mask)
return image
```

5.7 Definitie 'add_time_stamp'

In de definitie 'add_time_stamp' gaan we de tijd en de naam van de persoon op dit event toevoegen aan de afbeelding.

```
def add_time_stamp(image, text, name, font_size, color):
```

Eerst gaan we ervoor zorgen dat we op de afbeelding kunnen schrijven door 'ImageDraw.Draw' te gebruiken.

```
draw = ImageDraw.Draw(image)
```

Dan gaan we het lettertype ('arial') en de grote van de tekst ('font_size') instellen.

```
font = ImageFont.truetype("arial.ttf", font_size)
```

Hier berekenen we de bounding boxes (grensdozen) van twee tekststrings met behulp van de 'textbbox' methode uit de Python Imaging Library (PIL) of zijn opvolger, Pillow. Dit zijn de stappen:

- 'draw.textbbox((0, 0), text, font=font)':
 - Deze methode berekent de grenzen van de tekst 'text' wanneer deze wordt getekend op de positie '(0, 0)' met het opgegeven 'font'.
 - De '(0, 0)' coördinaten betekenen dat de berekening wordt gestart vanaf de linkerbovenhoek van de afbeelding.
 - 'text' is de string die de tijd vertegenwoordigt (bijv. "12:30 PM").
 - 'font' is een vooraf gedefinieerd lettertype object dat aangeeft welk lettertype en welke grootte moet worden gebruikt.
- 'draw.textbbox((0, 0), name, font=font)':
 - Net als bij de vorige regel, berekent deze methode de grenzen van de tekst 'name' wanneer deze wordt getekend op de positie '(0, 0)' met hetzelfde 'font'.
 - 'name' is de string die de naam vertegenwoordigt (bijv. "John Doe").

```
time_bbox = draw.textbbox((0, 0), text, font=font)
name_bbox = draw.textbbox((0, 0), name, font=font)
```

In deze code berekenen we de breedte en hoogte van twee tekststrings door gebruik te maken van de coördinaten van hun bounding boxes. Deze coördinaten zijn eerder verkregen met behulp van de 'textbbox' methode uit de Python Imaging Library (PIL) of zijn opvolger, Pillow. Hier zijn de stappen en uitleg:

- Berekenen van de Breedte:
 - 'time_text_width = time_bbox[2] - time_bbox[0]'
 - De breedte van de bounding box wordt berekend door het verschil te nemen tussen de rechter ('time_bbox[2]') en linker ('time_bbox[0]') coördinaten van de bounding box.
- Berekenen van de Hoogte:

- `time_text_height = time_bbox[3] - time_bbox[1]`
- De hoogte van de bounding box wordt berekend door het verschil te nemen tussen de onderste (`time_bbox[3]`) en bovenste (`time_bbox[1]`) coördinaten van de bounding box.

```
time_text_width = time_bbox[2] - time_bbox[0]
time_text_height = time_bbox[3] - time_bbox[1]
name_text_width = name_bbox[2] - name_bbox[0]
name_text_height = name_bbox[3] - name_bbox[1]
```

In deze code bepalen we de grootste hoogte van de twee tekststrings (tijd en naam) om deze waarde vervolgens te gebruiken als offset. Dit is handig voor het gelijkmatig positioneren van de tekst op een afbeelding. Hier is de uitleg en het doel van deze specifieke regel code:

1. Bepalen van de Maximale Hoogte:

- `max(time_text_height, name_text_height)`
- De `max` functie retourneert de grootste waarde van de twee opgegeven argumenten. In dit geval worden `time_text_height` en `name_text_height` vergeleken.
- `time_text_height` is de hoogte van de bounding box voor de tijdstekst.
- `name_text_height` is de hoogte van de bounding box voor de naamtekst.

2. Toewijzing aan de Variabele offset:

- De grootste waarde van `time_text_height` en `name_text_height` wordt toegewezen aan de variabele `offset`.
- Deze offset kan vervolgens worden gebruikt om de tekst op een consistente manier te positioneren, zodat ze niet overlappen en gelijkmatig verdeeld zijn.

```
offset = max(time_text_height, name_text_height)
```

In deze code bepalen we de posities voor de tijd- en naamtekst op een afbeelding, zodat ze gecentreerd en goed uitgelijnd zijn. De posities worden berekend op basis van de afmetingen van de tekst en de afbeelding zodat de tekst een beetje meer naar beneden staat anders gaat er iets over de tekst staan in de IQ Messenger app. Hier zijn de stappen en een gedetailleerde uitleg:

1. Berekenen van de Positie voor de Tijdstekst:

- `'time_position = ((image.width - time_text_width) // 2, offset)'`
- `'((image.width - time_text_width) // 2)'`: Deze berekening centreert de tijdstekst horizontaal op de afbeelding.
 - `'image.width'` is de breedte van de afbeelding.
 - `'time_text_width'` is de breedte van de bounding box van de tijdstekst.
 - Door het verschil te delen door 2, positioneer je de tekst precies in het midden van de afbeelding.

- `offset`: Dit is de verticale positie. Eerder hebben we de `offset` berekend als de maximale hoogte van de tijd- en naamtekst om ervoor te zorgen dat er voldoende ruimte is tussen de teksten.

2. Berekenen van de Positie voor de Naamtekst:

- `'name_position = ((image.width - name_text_width) // 2, offset + time_text_height)'`
- `'((image.width - name_text_width) // 2)'`: Deze berekening centreert de naamtekst horizontaal op de afbeelding.
 - `'name_text_width'` is de breedte van de bounding box van de naamtekst.
- `'offset + time_text_height'`: Deze berekening zorgt ervoor dat de naamtekst net onder de tijds tekst wordt geplaatst.
 - `'offset'` is de initiële verticale positie.
 - `'time_text_height'` wordt opgeteld bij de `offset` om de naamtekst direct onder de tijds tekst te plaatsen, met voldoende ruimte ertussen.

```
time_position = ((image.width - time_text_width) // 2, offset)
name_position = ((image.width - name_text_width) // 2, offset + time_text_height)
```

In deze code wordt de tijd- en naamtekst getekend op een afbeelding met behulp van de Python Imaging Library (PIL) of zijn opvolger, Pillow. De tekst wordt op specifieke posities geplaatst die eerder zijn berekend, en er wordt een bepaalde kleur gebruikt om de tekst te vullen.

```
draw.text(time_position, text, font=font, fill=color)
draw.text(name_position, name, font=font, fill=color)
```

Nu kunnen we de volledige afbeelding retourneren.

```
return image
```

6 API

Welkom bij de documentatie van onze Dwaaldetectie API. Hier wordt de essentiële functionaliteit beschreven. Deze API is ontworpen om te reageren op verzoeken met betrekking tot dwaaldetectie afbeeldingen, en biedt de mogelijkheid om deze afbeeldingen op te halen of te resetten.

In dit hoofdstuk duiken we dieper in op de logica en processen die achter de Dwaaldetectie API liggen. We beginnen met een overzicht van de configuratie en de endpoints die beschikbaar zijn. Vervolgens wordt elke functionaliteit in detail besproken, inclusief de stappen die de API onderneemt om te reageren op verzoeken en de manier waarop beveiliging is geïmplementeerd.

Deze API is nodig zodat we de afbeeldingen van de events in de SAFR applicatie kunnen krijgen. Ook moeten de afbeeldingen verwijderd worden omdat dit een persoonlijk gegeven is wat we niet mogen bijhouden. Het is de IQ Messenger die de afbeelding gaat resetten waardoor we hiervoor een API endpoint aanmaken.

6.1 Initialisatie

Aangezien we flask gebruiken om onze API endpoints te creëren moeten we eerst een flask applicatie aanmaken door 'app = Flask(__name__)' te doen. Ook willen we gebruik maken van authenticatie. Hiervoor moeten we ook de authenticatie initialiseren door 'auth = HTTPBasicAuth()' te runnen.

```
app = Flask(__name__)
auth = HTTPBasicAuth()
```

We moeten voor de applicatie ook een aantal gegevens hebben uit de config.json file. In deze 'try-except' blok gaan we proberen om de config.json file te openen en in te lezen onder de 'config' variabelen. Als we de config.json niet hebben kunnen openen dan gaan we de API afsluiten en een tekst laten zien die zegt dat we de config file niet konden vinden.

```
try:
    with open('./config.json', 'r') as f:
        config = json.load(f)
except Exception as e:
    print("The Program is terminated! Could not open/find the config file!")
    raise SystemExit
```

Voor de API endpoints hebben we de waarde van de pport nodig waarop we de API willen laten draaien, deze houden we bij in de 'api_port' variabele. We hebben ook de locatie waar we de afbeeldingen van events opslaan nodig, deze houden we bij in de 'image_location' variabele. Ook moeten we de locatie van het DeltaTechnics logo hebben wat we opslaan in de 'logo_image' variabele. Dan hebben we ook de gebruikersnaam en wachtwoord dat we voor de API willen gebruiken uit de config file halen. Deze slaan we op in de 'api_username' en 'api_password' variabelen.

```
api_port = config["api"]["apiPort"]
image_location = config["api"]["imageLocation"]
logo_image = config["api"]["logoLocation"]
api_username = config["api"]["username"]
api_password = config["api"]["password"]
```

Om de authenticatie te kunnen gebruiken moeten we een dictionary met gebruikers aanmaken. Hier komt dan de gebruiker uit de config file in te staan met de ingevulde gebruikersnaam en wachtwoord wat gesplitst in men een ':'.

```
users = {
    api_username: api_password
}
```

Om de API te laten draaien gaan we een definitie aanroepen die de API server opstart. Voor deze definitie moeten we de poort waarop de API moet starten meegegeven worden.

```
run(api_port)
```


De definitie 'run' gaat de API server opstarten. Om de server op te starten gebruiken we 'serve' van waitress aangezien de waitress server meer aan kan dan de development server van flask zelf. De 'host' zetten we op '0.0.0.0', dit gaat het ip adres gebruiken van het device waar deze applicatie op draait. En de 'port' moeten we de poort meegeven die in de config file stond zodat de API server effectief op deze poort draait.

```
def run(port):
    serve(app, host='0.0.0.0', port=port)
```

6.2 Autorisatie

In de definitie 'verify_password' gaan we de gebruikersnaam en wachtwoord verifiëren. Om ervoor te zorgen dat deze definitie voor de verificatie gebruikt gaat worden moeten we '@auth.verify_password' boven aan de definitie toevoegen.

```
@auth.verify_password
def verify_password(username, password):
```

In de definitie zelf gaan we kijken of de gebruikersnaam in de lijst van gebruikers zit en of het wachtwoord van deze gebruiker overeen komt met het wachtwoord in de dictionary. Als dit het geval is dan retourneren we de gebruikersnaam om te laten zien dat dit goedgekeurd is. Als dit niet het geval is dan retourneren we niets en dit laat zien dat de inloggegevens die ingegeven zijn niet correct zijn.

```
if username in users and users[username] == password:
    return username
```

6.3 'get_image' API

Om de afbeelding van het event op te halen via de API, gebruiken we de 'get_image' definitie. Om ervoor te zorgen dat deze definitie kan aangeroepen worden moeten we boven de definitie '@app.route' toevoegen. De eerste variabele is het pad dat gebruikt moet worden. Om dit endpoint aan te roepen moet je dus het ip en poort van de API server invullen + '/dwaaldetectie/' + de naam van de deur die gedetecteerd is, bijvoorbeeld 'http://127.0.0.1:5000/dwaaldetectie/voordeur' en dit gaat de afbeelding van de voordeur laten zien. Dit moet een 'get' endpoint zijn. Dit moeten we in de 'methods' variabele zetten. Daarnaast willen we ook ervoor zorgen dat er ingelogd moet worden om deze API endpoint te gebruiken. Hiervoor moeten we '@auth.login_required' toevoegen bovenaan de definitie.

```
@app.route('/dwaaldetectie/<name>', methods=['GET'])
@auth.login_required
def get_image(name):
```

Hier gaan we de folder waar we de afbeeldingen in bijhouden in een locale variabelen zetten zodat we zo weinig mogelijk globale variabelen moeten gebruiken.

```
directory = image_location
```

Dan gaan we de naam van de foto opslaan in een definitie. Aangezien het formaat van de afbeeldingen 'jpg' is kunnen we achter de naam van de deur '.jpg' zetten. De 'name'

variabele heeft de waarde van de naam van de deur die meegegeven is in de API call op de plaats van '<name>'.

```
file_path = f"{name}.jpg"
```

Dan gaan we eerst nakijken of het pad naar de afbeelding effectief bestaat. Dit kunnen we doen met 'os.path.exists'. Om dit na te kijken moeten we eerst de folder en de naam van de afbeelding samenvoegen door 'os.path.join' te gebruiken. Dit geeft een 'True' of 'False' dus als dit 'False' is dan bestaat de afbeelding niet.

```
if not os.path.exists(os.path.join(directory, file_path)):
```

Als het pad niet bestaat dan gaan we een nieuwe afbeelding creëren van het DeltaTechnics logo. Hiervoor roepen we de 'replace_and_modify_image' definitie in de 'ResetDwaaldetectieImages' file. Voor deze definitie moeten we het pad naar het logo meegeven, alsook het pad van de afbeelding waar we deze willen gaan halen. Hiervoor moeten het folder pad en afbeelding pad opnieuw samengevoegd worden. Ook gaan we de tekst die we op de afbeelding willen hebben ook meegeven. Aangezien de afbeelding niet beschikbaar was gaan we hier zetten "Image not available".

```
try:
    ResetDwaaldetectieImages.replace_and_modify_image(
        src_image_path=logo_image,
        dest_image_path=os.path.join(directory, file_path),
        text='Image not available'
    )
```

Als het niet lukt om de afbeelding te creëren gaan we hier een error message laten zien. We kunnen dit doen aangezien we het maken van de afbeelding in een 'try-except' blok gezet is.

```
except Exception as e:
    return jsonify(message=f"Error processing image: {str(e)}"), 500
```

Als de afbeelding wel gevonden kan worden dan gaan we deze als response nemen. Hiervoor gebruiken we de 'send_from_directory' van de flask library. Aangezien dit een afbeelding is moeten we ook het content type in de headers nog op afbeelding en jpg zetten ("image/jpg"). Nu is de response klaar en kunnen we deze retourneren naar de persoon dat de API heeft opgeroepen.

```
try:
    response = send_from_directory(directory, file_path)
    response.headers['Content-Type'] = 'image/jpg'
    return response
```

Als we voor een of andere manier de file toch niet meer kunnen vinden gaan we een error message laten zien dat de file niet gevonden kan worden en een 404 error retourneren.

```
except FileNotFoundError:
    return jsonify(message="File not found. If you believe this is an error, please contact support."), 404
```

Als er een ander probleem voorkomt dan gaan deze error als tekst retourneren als een 500 error.

```
except Exception as e:
    return jsonify(message=str(e)), 500
```

6.4 'reset_dwaaldetectie' API

Om de afbeelding van het event te resetten naar het DeltaTechnics logo via de API, gebruiken we de 'reset_dwaaldetectie' definitie. Om ervoor te zorgen dat deze definitie kan aangeroepen worden moeten we boven de definitie '@app.route' toevoegen. De eerste variabele is het pad dat gebruikt moet worden. Om dit endpoint aan te roepen moet je dus het ip en poort van de API server invullen + '/resetdwaaldetectie/' + de naam van de deur die gedetecteerd is, bijvoorbeeld 'http://127.0.0.1:5000/resetdwaaldetectie/voordeur' en dit gaat de afbeelding van de voordeur resetten naar het DeltaTechnics logo. Dit moet een 'get' endpoint zijn. Dit moeten we in de 'methods' variabele zetten. Daarnaast willen we ook ervoor zorgen dat er ingelogd moet worden om deze API endpoint te gebruiken. Hiervoor moeten we '@auth.login_required' toevoegen bovenaan de definitie.

```
@app.route('/resetdwaaldetectie/<name>', methods=['GET'])
@auth.login_required
def reset_dwaaldetectie(name):
```

We gaan de locatie van het DeltaTechnics logo in een locale variabele zodat we zo weinig mogelijk globale variabele gebruiken.

```
src_image_path = logo_image
```

Dan gaan we het pad naar de afbeelding die we willen resetten creëren. Dit doen we door de locatie van de folder waar we afbeeldingen opslaan samen voegen met de naam van de deur die meegegeven wordt in de API call, we hangen deze aan elkaar met een '\'. Ook moeten we achter de naam van de afbeelding '.jpg' zetten aangezien we met jpg afbeeldingen werken.

```
dest_image_path = f'{image_location}\{name}.jpg'
```

Om de afbeelding te resetten moeten we ook een tekst meegeven die we onderaan de afbeeldingen willen plaatsen. Hier gaan we de tekst "geen meldingen" plaatsen.

```
text = "geen meldingen"
```

Voor het resetten gebruiken we de 'replace_and_modify_image' definitie in de 'ResetDwaaldetectieImages' file. In deze definitie geven we de paden van het logo en van de event afbeelding mee alsook de tekst die we op de afbeelding willen zetten. We doen het resetten in een andere file omdat het resetten toch wel wat logica vraagt en als dit in de API zou gebeuren wordt de API file zeer lang en onordelijk.

```
ResetDwaaldetectieImages.replace_and_modify_image(src_image_path, dest_image_path, text)
```

Nu is de afbeelding gereset maar aangezien dit een get functie is moeten we ook nog iets retourneren. Hiervoor retourneren we een tekst dat de afbeelding van dit event succesvol is gereset.

```
return f'Image {name} reset'
```

7 AFBEELDING RESETTEN

In dit project wordt het resetten van de afbeeldingen niet rechtstreeks vanuit de API uitgevoerd. In plaats daarvan is deze taak gedelegeerd aan de Python-file 'ResetDwaaldetectieImages'. Dit script bevat alle noodzakelijke functionaliteiten om de afbeeldingen te resetten.

De API zal gebruik maken van een specifieke definitie binnen 'ResetDwaaldetectieImages' om de resetoperatie uit te voeren. Deze aanpak zorgt voor een duidelijke scheiding van verantwoordelijkheden en maakt de API overzichtelijker en eenvoudiger te onderhouden. Bovendien biedt het de flexibiliteit om de logica voor het resetten van afbeeldingen onafhankelijk te ontwikkelen en bij te werken zonder de API zelf te beïnvloeden.

7.1 Definitie 'replace_and_modify_image'

De 'replace_and_modify_image' is de definitie die aangeroepen wordt door de API. Deze definitie gaat ervoor zorgen dat de afbeelding gereset wordt. Om de definitie te gebruiken moeten we het pad naar het DeltaTechnics logo ('src_image_path') meegeven. Ook moeten we het pad naar de afbeelding die we willen vervangen ('dest_image_path') meegeven en de tekst die we op de afbeelding willen plaatsen.

```
def replace_and_modify_image(src_image_path, dest_image_path, text):
```

In deze code zorgen we ervoor dat een bestandsnaam en een directory-pad op een platformonafhankelijke manier worden gecombineerd en genormaliseerd. Dit is belangrijk omdat er soms zowel forward slashes (/) als backslashes (\) in het doorgegeven pad kunnen voorkomen, afhankelijk van waar het pad vandaan komt. Hier is een stapsgewijze uitleg van de code:

1. Ophalen van de Directory-naam:

- `'dest_dir = os.path.dirname(dest_image_path)'`
- Deze regel haalt het directory-gedeelte van het volledige pad 'dest_image_path' op. Bijvoorbeeld, als 'dest_image_path' `"/path/to/image.jpg"` is, dan zal 'dest_dir' `"/path/to"` zijn.
- `'os.path.dirname'` zorgt ervoor dat we alleen het pad tot de directory krijgen zonder de bestandsnaam.

2. Ophalen van de Bestandsnaam:

- `'dest_image_name = os.path.basename(dest_image_path)'`
- Deze regel haalt het bestandsnaamgedeelte van 'dest_image_path' op. Bijvoorbeeld, als 'dest_image_path' `"/path/to/image.jpg"` is, dan zal 'dest_image_name' `"image.jpg"` zijn.
- `'os.path.basename'` zorgt ervoor dat we alleen de naam van het bestand krijgen zonder het pad ernaartoe.

3. Samenvoegen van Directory en Bestandsnaam:

- `'new_image_path = os.path.join(dest_dir, dest_image_name)'`
- Deze regel voegt de directory en de bestandsnaam samen tot een volledig pad. `'os.path.join'` zorgt ervoor dat de juiste padseparator wordt gebruikt voor het besturingssysteem waarop de code wordt uitgevoerd.
- Dit betekent dat het juiste pad wordt gevormd, ongeacht of de padseparator een backslash of een forward slash is.

4. Normaliseren van het Pad:

- `'new_image_path = os.path.normpath(new_image_path)'`
- Deze regel normaliseert het pad, wat betekent dat eventuele redundante separaties en relatieve padsegmenten ('.' en '..') worden verwijderd.
- `'os.path.normpath'` zorgt ervoor dat het pad consistent en correct is voor het besturingssysteem waarop de code wordt uitgevoerd.

```
dest_dir = os.path.dirname(dest_image_path)
dest_image_name = os.path.basename(dest_image_path)
new_image_path = os.path.join(dest_dir, dest_image_name)
new_image_path = os.path.normpath(new_image_path)
```

De regel `'os.makedirs(dest_dir, exist_ok=True)'` wordt gebruikt om ervoor te zorgen dat de directory `'dest_dir'` bestaat. Indien deze directory (en eventuele bovenliggende directories) nog niet bestaan, worden ze aangemaakt. Hier is een gedetailleerde uitleg:

1. `'os.makedirs(dest_dir)'`:

- De functie `'os.makedirs()'` probeert de directorystructuur te maken die wordt gespecificeerd door `'dest_dir'`.
- Als `'dest_dir'` bijvoorbeeld `"/path/to/directory"` is, zal `'os.makedirs()'` proberen om zowel de directories `"/path"`, `"/path/to"` als `"/path/to/directory"` aan te maken als ze nog niet bestaan.

2. `'exist_ok=True'`:

- Het argument `'exist_ok=True'` betekent dat als de directory `'dest_dir'` al bestaat, er geen foutmelding zal worden gegenereerd. Dit voorkomt dat de code een fout gooit als de directory al aanwezig is.
- Zonder dit argument zou `'os.makedirs()'` een `'FileExistsError'` opwerpen als de directory al bestaat.

```
os.makedirs(dest_dir, exist_ok=True)
```

Deze code controleert of een bestand al bestaat op een bepaalde locatie (`'new_image_path'`) en verwijdert het indien nodig. Dit is belangrijk om ervoor te zorgen dat er geen ongewenste bestanden blijven bestaan, vooral wanneer je een bestand wilt overschrijven met nieuwe inhoud. Hier is een stap-voor-stap uitleg:

1. Controle of het Bestand Bestaat:

- `'if os.path.exists(new_image_path)'`:
 - Deze conditie controleert of er al een bestand of directory bestaat op het pad `'new_image_path'`.
 - `'os.path.exists()'` retourneert `'True'` als het pad bestaat, anders `'False'`.

2. Vergelijking van Paden:

- `'new_image_path.lower() != src_image_path.lower()'`:
 - Deze vergelijking controleert of de paden van het nieuwe bestand (`'new_image_path'`) en het bronbestand (`'src_image_path'`) niet hetzelfde zijn.

- 'lower()' wordt gebruikt om de paden naar kleine letters om te zetten, zodat de vergelijking niet hoofdlettergevoelig is. Dit is belangrijk op systemen waar bestandsnamen hoofdletterongevoelig zijn (zoals Windows), om vergissingen te voorkomen.

3. Verwijderen van het Bestand:

- 'os.remove(new_image_path)':
 - Als beide bovenstaande condities waar zijn (het bestand bestaat en de paden zijn niet hetzelfde), wordt 'os.remove()' aangeroepen om het bestand op 'new_image_path' te verwijderen.
 - Dit zorgt ervoor dat er geen oud bestand blijft bestaan op de locatie waar je een nieuw bestand wilt opslaan.

```
if os.path.exists(new_image_path) and new_image_path.lower() != src_image_path.lower():
    os.remove(new_image_path)
```

De regel 'shutil.copy(src_image_path, new_image_path)' kopieert een bestand van een bronpad ('src_image_path') naar een doelpad ('new_image_path'). Hier is een gedetailleerde uitleg:

1. 'shutil.copy()':

- 'shutil.copy()' is een functie uit de 'shutil' module in Python die wordt gebruikt om bestanden te kopiëren.
- Deze functie kopieert de inhoud van het bronbestand ('src_image_path') naar het doelbestand ('new_image_path').

2. Bronbestand ('src_image_path'):

- Dit is het pad naar het bestand dat je wilt kopiëren. Het kan een absoluut pad zijn (bijvoorbeeld '/home/user/source/image.jpg') of een relatief pad (bijvoorbeeld 'source/image.jpg').

3. Doelbestand ('new_image_path'):

- Dit is het pad waar je het bestand naartoe wilt kopiëren. Als 'new_image_path' een bestaande directory is, wordt het bestand met dezelfde naam in die directory geplaatst. Als het een pad naar een bestand is, wordt het bestand naar die locatie gekopieerd.

```
shutil.copy(src_image_path, new_image_path)
```

Nu hebben we de afbeelding veranderd met het DeltaTechnics logo maar we moeten ook nog de tekst toevoegen op deze afbeelding. Hiervoor gebruiken we de 'add_text_to_image' definitie waar we het pad voor de afbeelding en de tekst die er op moet komen moeten meegeven.

```
add_text_to_image(new_image_path, text)
```


7.2 Definitie 'add_text_to_image'

In de definitie 'add_text_to_image' gaan we de tekst die meegegeven is toevoegen aan de afbeelding die op het pad staat dat meegegeven is.

```
def add_text_to_image(image_path, text):
```

We openen de afbeelding die op het meegegeven pad staat door gebruik te maken van de 'Image' library.

```
with Image.open(image_path) as img:
```

Deze code probeert een specifiek lettertype te laden voor het tekenen van tekst op een afbeelding. Als het opgegeven lettertype niet beschikbaar is, wordt een standaardlettertype geladen als back-up. Dit is belangrijk om ervoor te zorgen dat de functionaliteit niet faalt als het gewenste lettertype niet kan worden gevonden of geopend. Hier is een stap-voor-stap uitleg:

1. Probeer het Specifieke Lettertype te Laden:

- `'font = ImageFont.truetype("arial.ttf", size=60)'`:
 - Deze regel probeert het TrueType-lettertype "arial.ttf" te laden met een grootte van 60 punten.
 - `'ImageFont.truetype()'` is een methode uit de Pillow-bibliotheek die een TrueType- of OpenType-lettertype laadt en een `'ImageFont'`-object retourneert dat kan worden gebruikt om tekst te tekenen op een afbeelding.
 - `"arial.ttf"` is de naam van het lettertypebestand. Het moet beschikbaar zijn op het systeem of in het pad waar het script zoekt.

2. Foutafhandeling met 'try...except':

- `'except IOError:'`:
 - Als er een `'IOError'` optreedt (bijvoorbeeld als het lettertypebestand niet kan worden gevonden of geopend), wordt de except-block uitgevoerd.
 - `'IOError'` is een veelvoorkomende fout die optreedt bij problemen met het lezen van bestanden.

3. Fallback naar Standaardlettertype:

- `'font = ImageFont.load_default()'`:
 - Als de `'IOError'` wordt opgevangen, wordt het standaardlettertype geladen met `'ImageFont.load_default()'`.
 - Dit zorgt ervoor dat er altijd een lettertype beschikbaar is, zelfs als het specifieke gewenste lettertype niet kan worden geladen.

```
try:
    font = ImageFont.truetype("arial.ttf", size=60)
except IOError:
    font = ImageFont.load_default()
```

Deze regel maakt een tekenobject (draw) dat wordt gebruikt om op een afbeelding (img) te tekenen.

```
draw = ImageDraw.Draw(img)
```

De regel `'width, height = img.size'` haalt de breedte en hoogte van een afbeelding ('img') op en slaat deze op in de variabelen 'width' en 'height'. Dit is een eenvoudige manier om de afmetingen van een afbeelding te verkrijgen met behulp van de Pillow-bibliotheek.

```
width, height = img.size
```

Deze code berekent de afmetingen van een tekststring door gebruik te maken van de 'textbbox' methode van de Pillow-bibliotheek. Dit helpt bij het nauwkeurig positioneren van tekst op een afbeelding. Hier is een stap-voor-stap uitleg:

1. Berekenen van de Bounding Box voor de Tekst:

- `'text_bbox = draw.textbbox((0, 0), text, font=font)'`:
 - Deze regel berekent de grenzen (bounding box) van de tekst 'text' als deze wordt getekend op de positie '(0, 0)' met het opgegeven font.
 - `'draw.textbbox()'` is een methode uit de Pillow-bibliotheek die de coördinaten van de bounding box retourneert als een tuple '(left, top, right, bottom)'.
 - '(0, 0)' geeft aan dat de tekst wordt geplaatst vanaf de linkerbovenhoek van de afbeelding.
 - 'text' is de tekststring waarvan de bounding box wordt berekend.
 - 'font' is het lettertype object dat aangeeft welk lettertype en welke grootte moet worden gebruikt.
 - De bounding box coördinaten worden toegewezen aan de variabele 'text_bbox'.

2. Berekenen van de Breedte van de Tekst:

- `'text_width = text_bbox[2] - text_bbox[0]'`:
 - Deze regel berekent de breedte van de tekst.
 - `'text_bbox[2]'` is de rechter coördinaat van de bounding box.
 - `'text_bbox[0]'` is de linker coördinaat van de bounding box.
 - Door het verschil te nemen tussen de rechter en linker coördinaat, wordt de breedte van de tekst berekend.
 - De berekende breedte wordt toegewezen aan de variabele 'text_width'.

3. Berekenen van de Hoogte van de Tekst:

- `'text_height = text_bbox[3] - text_bbox[1]'`:
 - Deze regel berekent de hoogte van de tekst.
 - `'text_bbox[3]'` is de onderste coördinaat van de bounding box.
 - `'text_bbox[1]'` is de bovenste coördinaat van de bounding box.

- Door het verschil te nemen tussen de onderste en bovenste coördinaat, wordt de hoogte van de tekst berekend.
- De berekende hoogte wordt toegewezen aan de variabele 'text_height'.

```
text_bbox = draw.textbbox((0, 0), text, font=font)
text_width = text_bbox[2] - text_bbox[0]
text_height = text_bbox[3] - text_bbox[1]
```

Deze code berekent de positie om een tekststring gecentreerd te plaatsen aan de onderkant van een afbeelding, met een kleine marge van 10 pixels van de onderrand. Dit is nuttig voor het dynamisch positioneren van tekst op een afbeelding. Hier is een stap-voor-stap uitleg:

1. Berekenen van de Horizontale Positie (X-coördinaat):

De horizontale positie wordt berekend met de uitdrukking `(width - text_width) / 2`.

- 'width' is de totale breedte van de afbeelding.
- 'text_width' is de breedte van de tekst, eerder berekend.
- Door het verschil tussen de breedte van de afbeelding en de breedte van de tekst te delen door 2, wordt de tekst horizontaal gecentreerd. Deze berekening plaatst de tekst zodanig dat er evenveel ruimte is aan zowel de linker- als de rechterkant van de tekst.

2. Berekenen van de Verticale Positie (Y-coördinaat):

De verticale positie wordt berekend met de uitdrukking `height - text_height - 10`.

- 'height' is de totale hoogte van de afbeelding.
- 'text_height' is de hoogte van de tekst, eerder berekend.
- '10' is een marge van 10 pixels van de onderkant van de afbeelding.
- Door de hoogte van de tekst en de marge van 10 pixels van de totale hoogte van de afbeelding af te trekken, wordt de tekst net boven de onderkant van de afbeelding geplaatst.

```
position = ((width - text_width) / 2, height - text_height - 10)
```

Nu gaan we de tekst op de afbeelding zetten met 'draw.text' hiervoor geven we de berekende positie, de tekst die we er op willen zetten en het font dat we willen gebruiken mee. Ook moeten we meegeven in welke kleur we de tekst willen hebben. Ik heb gekozen voor zwarte tekst aangezien de achtergrond wit is. Maar als de kleur veranderd moet worden kan je de "black" veranderen naar een andere kleur.

```
draw.text(position, text, font=font, fill="black")
```

Dan is de afbeelding klaar en kunnen we deze opslaan.

```
img.save(image_path)
```

8 IQ MESSENGER

IQ Messenger is verantwoordelijk voor het aansturen van de alarmen. Dit wordt gerealiseerd door een API aan te roepen die de event flow van de betreffende deur activeert. Wanneer de API wordt aangeroepen, start de event flow die gekoppeld is aan de specifieke deur. Deze flow stuurt vervolgens een alarm naar alle verbonden apparaten die door de medewerkers worden gebruikt.

Op deze apparaten ontvangen de medewerkers een melding met de afbeelding van het event. Ze krijgen de mogelijkheid om de afbeelding te bekijken en te beoordelen. Vervolgens kunnen ze het event accepteren of weigeren. Bij acceptatie van het event wordt de afbeelding direct gereset om te voorkomen dat persoonlijke gegevens onbedoeld worden gedeeld of bewaard. Dit is een belangrijke maatregel om de privacy en veiligheid van individuen te waarborgen.

In het geval dat het event niet wordt geaccepteerd door de medewerkers, zorgt het systeem ervoor dat de afbeelding en bijbehorende gegevens na een bepaalde tijd automatisch worden gereset. Specifiek, als het event niet binnen 15 minuten wordt geaccepteerd, zal de API van het dwaaldetectieprogramma worden aangeroepen om het event te resetten. Hierdoor worden alle relevante gegevens gewist, wat helpt om de integriteit en vertrouwelijkheid van de informatie te behouden.

8.1 Devices

De connectie met het dwaaldetectie programma wordt als 'HTTP I/O host' aangemaakt. Hiervoor moeten we het ip adress van het device waar het dwaaldetectie programma op draait als 'Code' zetten. Ook moeten we bij 'Port' de poort waar de dwaaldetectie API op draait zetten. We moeten ook bij 'Username' en 'Password' de gebruikersnaam en wachtwoord van de dwaaldetectie API zetten.

Code	<input type="text" value="10.20.0.117"/>
Name	<input type="text" value="Gezichtsherkenning-server"/>
Type	<input type="text" value="HTTP I/O Host"/>
Service	<input type="text" value="HTTP I/O"/>
Callback	<input type="text"/>
Related camera URL	<input type="text" value="Please select Related camera URL"/>
Related URL	<input type="text" value="Please select Related URL"/>
Related App	<input type="text" value="Please select Related App"/>
Location	<input type="text" value="Select location"/>
Scheme	<input type="text" value="HTTP"/>
Port	<input type="text" value="5000"/>
Username	<input type="text" value="gezichtsherkenning"/>
Password	<input type="password" value="••••••••"/>
<div><input type="button" value="Cancel"/> <input type="button" value="Save"/></div>	

We moeten ook voor elke deur een 'HTTP I/O device' maken zodat we deze kunnen gebruiken om tussen de verschillende flows te differentiëren. Hiervoor is het belangrijk dat de 'Code' dezelfde naam heeft als de deuren in de SAFR applicatie.

Code	<input type="text" value="Gang-Personeel"/>
Name	<input type="text" value="Gezichtsherkenning gang personeel"/>
Type	<input type="text" value="HTTP I/O Device"/>
Service	<input type="text" value="HTTP I/O"/>
Callback	<input type="text"/>
Related camera URL	<input type="text" value="Dwaaldetectie personeelsgang"/>
Related URL	<input type="text" value="Please select Related URL"/>
Related App	<input type="text" value="Please select Related App"/>
Location	<input type="text" value="Select location"/>

Ook moeten we voor het laten zien van de event afbeeldingen maken we een camera aan. Hiervoor gebruiken we een 'Camera URL' device. Dit zullen we moeten doen voor elke deur aangezien we de URL naar de afbeelding van elke deur moeten invullen bij 'Code'. In dit geval is de URL

'http://gezichtsherkenning:Rijkevorsel@10.20.0.117:5000/dwaaldetectie/gang-personeel' waar 'gezichtsherkenning:Rijkevorsel' de gebruikersnaam en wachtwoord van de dwaaldetectie API zijn. De '10.20.0.117:5000' zijn het ip adres en poort van de dwaaldetectie API. En 'dwaaldetectie' is het endpoint dat we moeten aanroepen voor de afbeelding. Dit deel is voor elke camera hetzelfde, maar dan moeten we achteraan ook nog de naam van de deur plaatsen om deze specifieke afbeelding aan te roepen.

Code	<input type="text" value="http://gezichtsherkenning:Rijkevorsel@10.20.0.117:5000/dwaaldetectie/gang-personeel"/>
Name	<input type="text" value="Dwaaldetectie personeelsgang"/>
Type	<input type="text" value="Camera URL"/>
Service	<input type="text" value="Camera URL"/>
Callback	<input type="text"/>

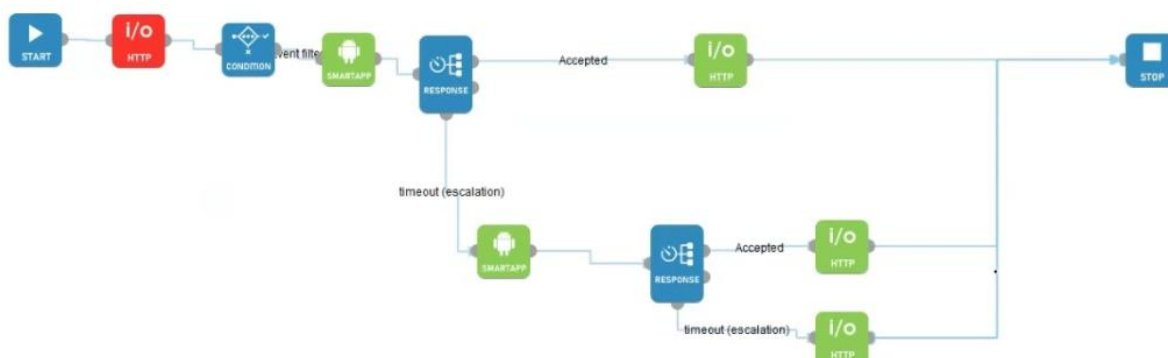
8.2 Event flow

De event flow stuurt het alarm aan en brengt de medewerkers op de hoogte van een dwaaldetectie bij een bewoner. Medewerkers kunnen vervolgens de afbeelding van dit event bekijken. Ze hebben de optie om het event te accepteren of af te wijzen. Bij acceptatie wordt de afbeelding onmiddellijk gereset om de privacy te waarborgen. Als het event niet wordt geaccepteerd, wacht het systeem 15 minuten voordat de afbeelding automatisch wordt gereset.

Voor elke deur moet een aparte event flow worden aangemaakt. Deze flows hebben allemaal dezelfde structuur, met als enige verschil dat de specifieke deurinformatie moet worden aangepast. Hierdoor kan het systeem nauwkeurig en efficiënt reageren op dwaaldetecties per individuele deur, terwijl de consistente structuur de implementatie en het beheer vereenvoudigt.

8.2.1 Volledige Flow

Hier is de volledige flow te zien. Eerst gaan we kijken of er een nieuw event is op de 'HTTP I/O' server is. Dan gaan we kijken of dit event niet 2 keer na elkaar gestuurd is. Nu kunnen we het event sturen naar de devices waar de afbeelding van het event ook kan bekeken worden. Als het event geaccepteerd wordt dan word er een API call gemaakt om de afbeelding te resetten en stopt de flow. Als het niet geaccepteerd wordt dan wordt het event nog eens doorgestuurd en als dit dan nog niet geaccepteerd wordt dan zal na 15 minuten de API call om de afbeelding te resetten gemaakt worden en stopt de flow ook.



8.2.2 Event input

TODO

Dit is de flow van 'Gang-Personeel' Hier moeten we in zetten van waar we de API call verwachten. Dit is een 'HTTP I/O host' device waarin we de connectie met de dwaaldetectie applicatie hebben ingesteld.

Dan hebben we ook nog het device dat we moeten invullen. Dit gaat ervoor zorgen dat we de juiste flow aanroepen aangezien we vanuit de dwaaldetectie applicatie het device moeten meegeven wat de deur is. Voor de devices hebben we 'HTTP I/O Device' aangemaakt met de namen van al de verschillende deuren.

Settings for: HTTP I/O event input

From: ⊕ Add

1 10.20.0.117 Gezichtsherkenning-server ✕

Device: ⊕ Add

1 Gang-Personeel Gezichtsherkenning gang ✕

Is wildcard location: ☐

Wildcard 1 (Is Char?): ☐

Wildcard 2 (Is Char?): ☐

Wildcard 3 (Is Char?): ☐

Wildcard 4 (Is Char?): ☐

Range wildcard is connected to the last wildcard

Range Wildcard:

Host available: ☐

Host unavailable: ☐

Scheduled alarm flow options

Schedule flow:

Run flow

Ok

8.2.3 Condition

Voor de condition hebben we gezet dat hetzelfde event niet binnen de 2 seconden opnieuw mag getriggered worden.

Settings for: Condition element

Condition type:

Not handle same events within:

Cancel

8.2.4 Alarm aansturen

Met de 'Smartapp event output' gaan we een event creëren. Hiervoor moeten we selecteren dat we dit naar al de devices willen sturen. Hiervoor gebruiken we degroup die hiervoor is aangemaakt. En als dit niet werkt dan sturen we het naar een andere groep wat dit ook naar alles dat lukt stuurt. Voor de rest kan je ook nog de ringtoon, de kleur en het icoon van het event aanpassen maar dit heeft niet veel invloed op de werking dus dit is meer persoonlijke preferentie.

Settings for: SmartApp event output (Android)

To: + Add

1 Group: SmartApp - Prinsenhof - Alle Smart ✕

2 System: Prinsenhof_ALL ✕

Dynamic recipient: ☐

Use initiator location: ☐

Use initiator location for Static groups: ☐

Send to the responder: ☐

Event type: Message ▼

Priority during voice call: High ▼

Ring volume: 100 ▼

Ringtone: Ringtone 34 ▼

Ring time: 60 ▼

Vibration: ☒

Display icon: Evacuation ▼

Color: Red ▼

Priority: 3 high priority ▼

Beep until reset: ☒

Display lightup time: 60 ▼

Periodic vibration: 30 ▼

Callback number 1:

Callback number 2:

Use callback number: ☐

Cancel
Ok

Wat wel ingesteld moet worden is dat de 'related camera URL' juist moet gezet worden. Dit is een camera URL device dat is aangemaakt. Ook moet de 'Include timestamp' uit staan omdat we de tijd van het event met de API call doorsturen.

Callback buttons:	Both
Related camera URL:	Dwaaldetectie personeelsgan
Related URL:	Please select item
Related app:	Please select item
Enable ETA:	<input type="checkbox"/>
Disable accept and reject buttons:	<input type="checkbox"/>
Enable close button:	<input type="checkbox"/>
Ignore if device is charging:	<input type="checkbox"/>
Force to event details:	<input checked="" type="checkbox"/>
Include timestamp:	<input type="checkbox"/>
Send auto generated message:	<input checked="" type="checkbox"/>
Use device related camera:	<input checked="" type="checkbox"/>
Use device related URL:	<input type="checkbox"/>
Use device related app:	<input type="checkbox"/>
Subject:	Dwaaldetectie Personeelsgan
Display message:	<div></div>
Publish to floor plan:	<input type="checkbox"/>

8.2.5 Response

Het event kan geaccepteerd worden of niet. Als het geaccepteerd is dan gaat dit door naar het aanroepen van de API om de afbeelding te resetten. Als het event niet geaccepteerd wordt of er is geen response na 60 seconden (1 minuut) dan zal de flow naar opnieuw het event doorsturen.

Settings for: Response element

Time out in second:

Erase on response: ☒

Erase on timeout: ☒

For the following devices: ⊕ Add

1 ⊗

Publish to floor plan: ☐

Publish to real-time console: ☐

Cancel Ok

De 'Smartapp event output' is exact hetzelfde als hiervoor.



Als er een response is of er is geen response voor 600 seconden gaat de API om de afbeelding te resetten aanroepen.

Settings for: Response element

Time out in second:

Erase on response: ☒

Erase on timeout: ☐

For the following devices: ⊕ Add

1 ⊗

Publish to floor plan: ☐

Publish to real-time console: ☐

Cancel Ok

8.2.6 Afbeelding resetten

Om de API voor het resetten van de afbeelding aan te roepen gebruiken we een 'HTTP I/O event output' waar we het 'HTTP I/O host' device gebruiken aangezien hier de connectie met de API staat. Bij 'Endpoint path' moeten we het pad naar het endpoint zetten. Voor het resetten is dit 'resetdwaaldetectie' samen met de naam van de deur van de flow, in dit geval is dit de 'gang-personeel' deur.

Settings for: HTTP I/O event output

To: ⊕ Add

1 10.20.0.117 Gezichtsherkenning-server ⊗

Use initiator location: ☐

Send to the responder: ☐

Endpoint path (relative): resetdwaaldetectie/gang-per

Include device and message: ☐

Include response code: ☐

Publish to floor plan: ☐

Publish to real-time console: ☐

Execution timeout: Please select timeout ⌵

Cancel Ok

De 3 'HTTP I/O event outputs' zijn allemaal hetzelfde.



Hierna stopt de flow.