THOMAS MORE

The Moving Gallery

Realisatiedocument

Tibo Vermunicht
Student Bachelor in de Toegepaste Informatica – Applicatieontwikkeling

2025 - 2026

Inhoudsopgave

1. Inleiding

Dit document geeft een uitgebreide beschrijving van de realisatie van The Moving Gallery, een project dat tijdens mijn stage bij Sodego werd ontwikkeld. Sodego is een relatief klein IT-bedrijf dat ongeveer tien jaar actief is en zich specialiseert in het maken van software op maat, maar eveneens sterk aanwezig is in design, IoT-toepassingen en databaseontwikkeling. De stageopdracht werd uitgevoerd voor fotoclub Studio2290, eveneens gevestigd in Vorselaar, waarvan de eigenaar van Sodego voorzitter is.

In het eerder opgestelde projectplan werden de doelstellingen, scope en geplande aanpak van The Moving Gallery vastgelegd. Het doel van de opdracht was het ontwikkelen van een desktopapplicatie die kan gebruikt worden voor zowel het beheren van data als voor de aansturing van een opstelling met vijf televisieschermen. Deze schermen zijn gemonteerd op een mechanische rail, waardoor ze zowel kunnen roteren als horizontaal bewegen. De applicatie moest een centrale en gebruiksvriendelijke manier bieden om reeksen aan te maken voor de leden van de fotoclub, foto's aan deze reeksen toe te voegen en de gewenste positie en rotatie van elk scherm te configureren.

Tijdens de tweejaarlijkse tentoonstelling van fotoclub Studio2290 zal deze opstelling worden ingezet. De aangemaakte reeksen, die telkens tot maximaal vijf foto's kunnen bevatten, worden dan automatisch op de schermen weergegeven. Elk scherm beweegt daarbij zelf naar de vooraf ingestelde positie en rotatie, zonder dat er botsingen ontstaan. De desktopapplicatie vormt dus de kern van zowel de inhoudelijke voorbereiding als de technische aansturing van The Moving Gallery.

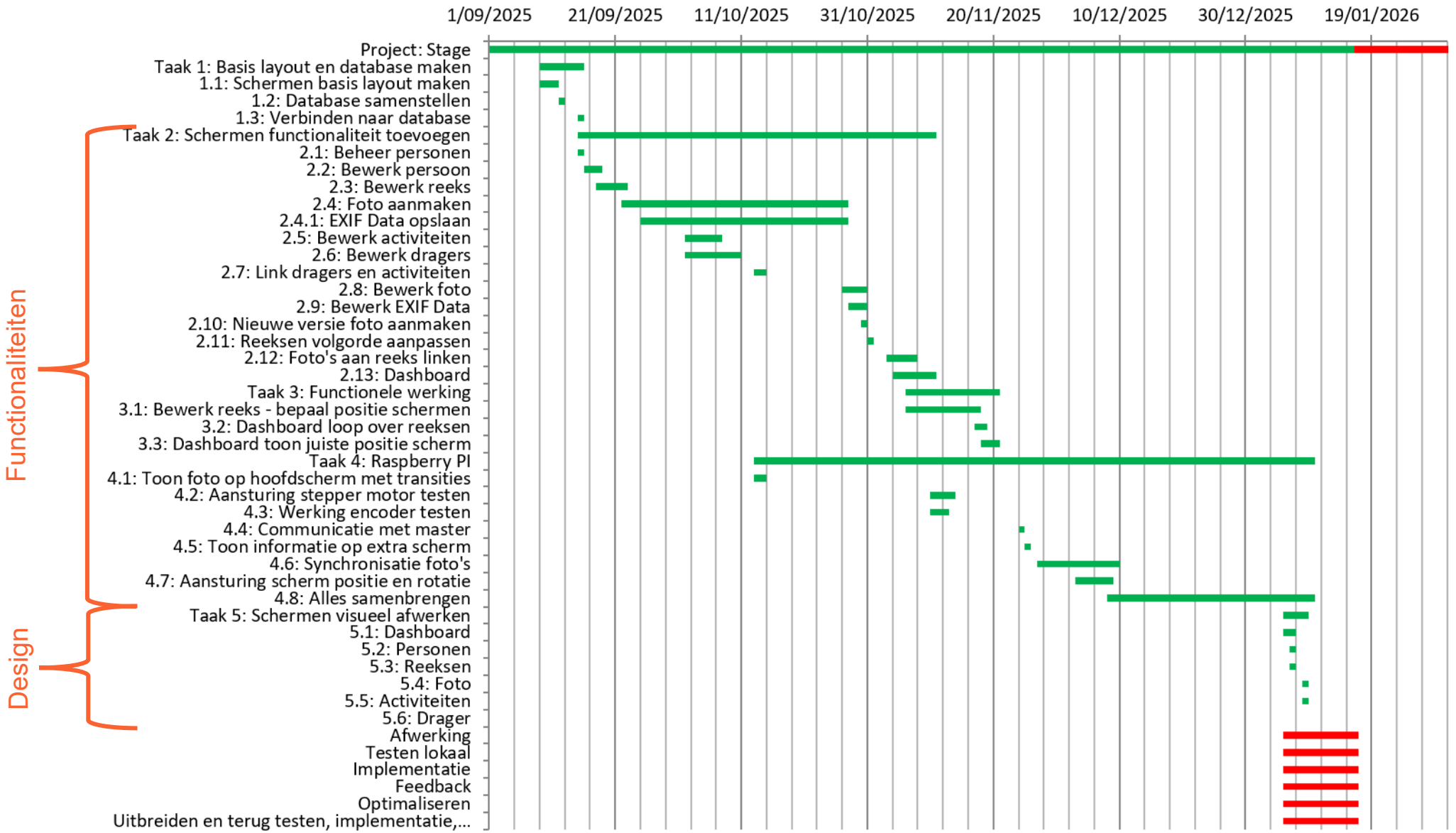
Dit realisatiedocument beschrijft de analyse en realisatie van de verschillende onderdelen en hoe deze samengebracht worden tot één werkend geheel. Ten slotte wordt het document afgesloten met een besluit dat het realisatiedocument samenvat.

2. Planning

Bij het begin van mijn stage heb ik een planning opgesteld waarin een overzicht gegeven wordt van de verschillende taken in het project. Deze planning geeft een duidelijke structuur voor het project en een geschatte tijdsduur van elke taak. Tijdens mijn stage zijn er enkele aanpassingen en uitbreidingen geweest aan de originele planning al is de algemene structuur hetzelfde gebleven. In de desktopapplicatie waren er nog enkele onderdelen van gegevensbeheer die toen nog niet ter sprake waren gekomen en nog niet in de planning stonden. Daarnaast zijn er ook enkele aanpassingen doorgevoerd aan de aansturing van de motoren, wat initieel niet tot mijn takenpakket behoorde.

Voor deze algemene structuur heb ik gekozen om eerst de focus te leggen op de functionaliteiten van mijn project, en later pas te focussen op de afwerking van het design van de applicatie aangezien dit ook een lagere prioriteit had voor de opdrachtgever. Het project bestaat ook uit drie onderdelen die apart hun functionaliteiten hebben en uiteindelijk samen moeten werken om het project tot een goed geheel te brengen.

Op volgende pagina staat mijn finale versie van mijn planning die een goed overzicht geeft wanneer ik aan welk onderdeel gewerkt heb.



Figuur 1 Gantt-chart planning

3. Analyse

In dit onderdeel wordt toegelicht welke tools en technologieën tijdens de stage zijn gebruikt. Daarnaast worden ook de gebruikte hardwarecomponenten besproken. Tot slot geven de modellen een schematisch overzicht van de opdracht en de bijhorende database.

3.1. Gebruikte tools & technologieën

Tijdens mijn stage heb ik gebruik gemaakt van een brede set tools en programmeertalen die werden ingezet voor softwareontwikkeling, embedded aansturing en databeheer. De keuze voor de gebruikte tools voor de desktopapplicatie, alsook de database, werd volledig bepaald door de opdrachtgever. De tools die bij de Raspberry Pi en Arduino horen, lagen eveneens al vast door de infrastructuur van deze componenten. Enkel de gekozen API-library was volledig mijn keuze. Hieronder volgt een overzicht van de gebruikte technologieën.

3.1.1. VB.Net

VB.Net was de gekozen technologie om de desktopapplicatie te ontwikkelen, waarbij WinForms werd gebruikt voor de front-end. Dit was een keuze van de opdrachtgever. Aangezien ik nog geen ervaring had met deze programmeertaal, vormde dit een uitdaging voor mij.

3.1.2. Visual Studio

Visual Studio werd gebruikt als ontwikkelomgeving voor de desktopapplicatie, aangezien dit de officiële en meest complete IDE is voor .NET-ontwikkeling.

3.1.3. MSSQL

De bestaande database was gebouwd in Microsoft SQL Server. Bij mijn stagebedrijf werken ze uitsluitend met Microsoft SQL en hebben ze hier veel ervaring mee. Door SQL-queries te gebruiken kon ik data uit de database ophalen en manipuleren.

3.1.4. Azure Data Studio

Azure Data Studio werd gebruikt voor het beheren en analyseren van de MSSQL-database. Het bood een lichtere interface dan SQL Server Management Studio.

3.1.5. Visual Studio Code

VS Code was mijn primaire ontwikkelomgeving voor de Python-projecten op de Raspberry Pi, evenals voor het programmeren van de Arduino. Ik had al ervaring met deze IDE, die bovendien veel nuttige extensies bevat, zoals linting en SSH-ondersteuning.

3.1.6. Python

Op de Raspberry Pi's draaien applicaties geschreven in Python. Deze zijn verantwoordelijk voor de synchronisatie van fotobestanden, de aansturing van de tv en het infoscherm, en de communicatie met zowel de desktopapplicatie als de Arduino.

3.1.7. FastAPI

FastAPI werd gebruikt om een RESTful API te bouwen voor de synchronisatie van foto's vanuit de desktopapplicatie naar de Raspberry Pi's. Daarnaast werd FastAPI ingezet om een WebSocket op te zetten voor realtime communicatie tussen elke Raspberry Pi en de desktopapplicatie, zodat steeds de juiste foto wordt weergegeven. De keuze voor FastAPI is gebaseerd op het feit dat het een zeer hoge performantie biedt, een duidelijke en leesbare code-structuur stimuleert en automatisch documentatie genereert via Swagger UI en ReDoc. Daarnaast is het framework eenvoudig te schalen, waardoor het geschikt is voor zowel kleine toepassingen als grotere, groeiende systemen.

3.1.8. PlatformIO

PlatformIO werd gebruikt voor het ontwikkelen en uploaden van firmware naar de Arduino. Het maakte het programmeren via VS Code eenvoudiger en bood bovendien mogelijkheden voor seriële debugging.

3.1.9. C++

De Arduino's draaiden op C++-code die verantwoordelijk was voor het aansturen van de motoren die de rotatie en positie van de schermen bepaalden. Naast de motorsturing was het ook belangrijk dat de incrementele encoder correct werd uitgelezen om de exacte positie te bepalen.

3.1.10. Kicad

Kicad werd gebruikt voor het ontwerpen van het elektronische schema en de PCB-layout. Deze PCB zorgt voor een nette en betrouwbare verbinding tussen de stappenmotor-aanstuurmodules en de Arduino.

3.2. Hardware

In deze sectie worden de verschillende hardwarecomponenten beschreven die worden ingezet om de functionaliteiten van het systeem te realiseren.

3.2.1. Raspberry Pi

De Raspberry Pi fungeert als een minicomputer voor de aansturing van zowel de tv als het infoscherm. Daarnaast dient hij als tussenstation voor de communicatie met de hoofdapplicatie en verzorgt hij de seriële communicatie met de Arduino voor het aansturen van de beweging.

3.2.2. Arduino Nano Every

De Arduino Nano Every wordt gebruikt voor de aansturing van de motoren. Deze microcontroller ontvangt commando's van de Raspberry Pi via een seriële verbinding en vertaalt deze naar exacte stappen voor de motorsturing. Daarnaast leest hij de encoder uit om de positie van het mechanisme nauwkeurig te bepalen.

De keuze van dit Arduino-model is gebaseerd op de vereiste werkspanning van 5V en het feit dat dit type al in ruime hoeveelheid beschikbaar was.

3.2.3. Stappenmotoren

Voor de rotatie en de positionering van het mechanisme worden twee verschillende stappenmotoren gebruikt, elk afgestemd op hun specifieke functie binnen het systeem. Beide motoren zijn van het type NEMA 23 bipolaire stappenmotoren, maar verschillen in koppel en afmetingen.

De motor die verantwoordelijk is voor de positieaansturing is het model 23HS22-1504S. Deze motor levert een hoger koppel, wat nodig is voor het verplaatsen van het mechanisme.
Voor de rotatiesturing wordt het model 23HS16-0884S gebruikt. Deze motor levert minder koppel, maar is kleiner in de diepte wat hij geschikt maakt voor de beperkte ruimte die beschikbaar is. Hij levert wel voldoende koppel aangezien deze enkel de rotatie van de tv moet doen en niet het hele mechanisme.

3.2.4. Encoder

De DLS40E-S3AV00360 encoder wordt gebruikt voor de nauwkeurige bepaling van de positie van het mechanisme. Het betreft een incrementele encoder met een resolutie van 360 pulsen per rotatie.

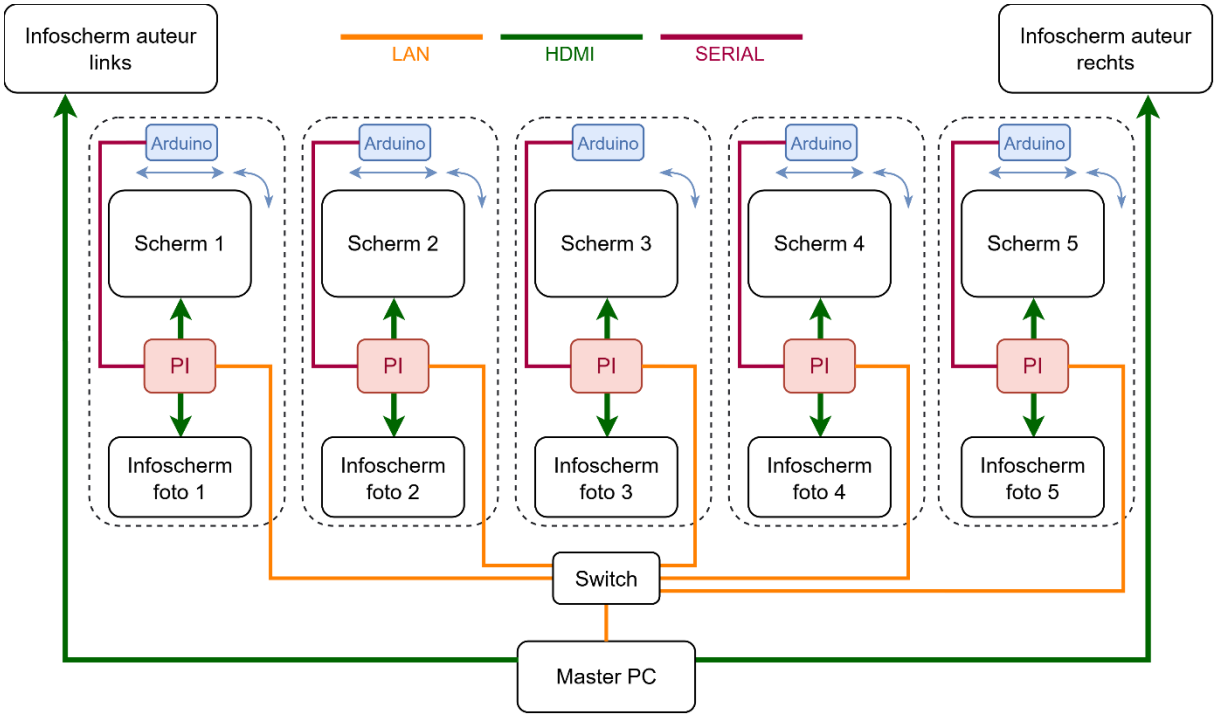
Door gebruik te maken van quadratuurdecodering op de twee uitgangssignalen kan zowel de draairichting worden bepaald als de effectieve resolutie worden verhoogd. Omdat elke puls vier detecteerbare flanken bevat (A-stijgend, A-dalend, B- stijgend, B- dalend), ontstaat er een effectieve resolutie van 1440 stappen per rotatie.

In combinatie met een poelie van 20 tanden resulteert dit in een praktische nauwkeurigheid van ongeveer ±0,1mm, wat ruim voldoende is voor de vereisten.

3.3. Modellen

3.3.1. Systeemarchitectuurdiagram

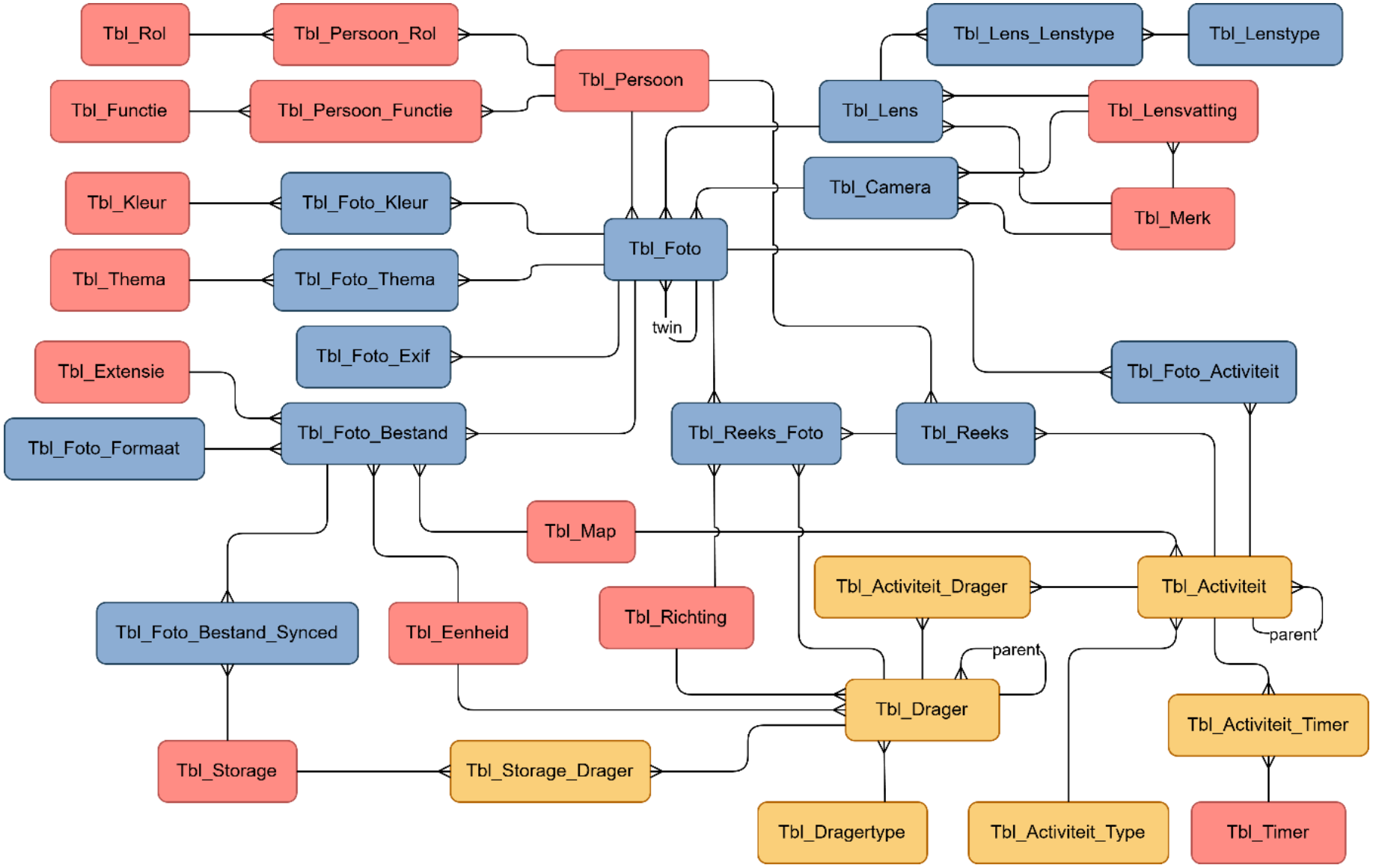
Het onderstaande diagram geeft een overzichtelijk beeld van de volledige opstelling en de onderlinge verbindingen tussen componenten. De installatie bestaat uit vijf identieke units, waarbij elke Raspberry Pi zowel het hoofdscherm als het bijhorende infoscherm aanstuurt en daarnaast een seriële verbinding onderhoudt met de gekoppelde Arduino.
De master-pc verzorgt de aansturing van beide auteur-infoschermen en is via een lokaal netwerk, opgebouwd rond een centrale switch, verbonden met elke Raspberry Pi. Op deze manier kan de master-pc alle units centraal beheren en synchroniseren.



Figuur 2 Systeemarchitectuurdiagram

3.3.2. Datamodel

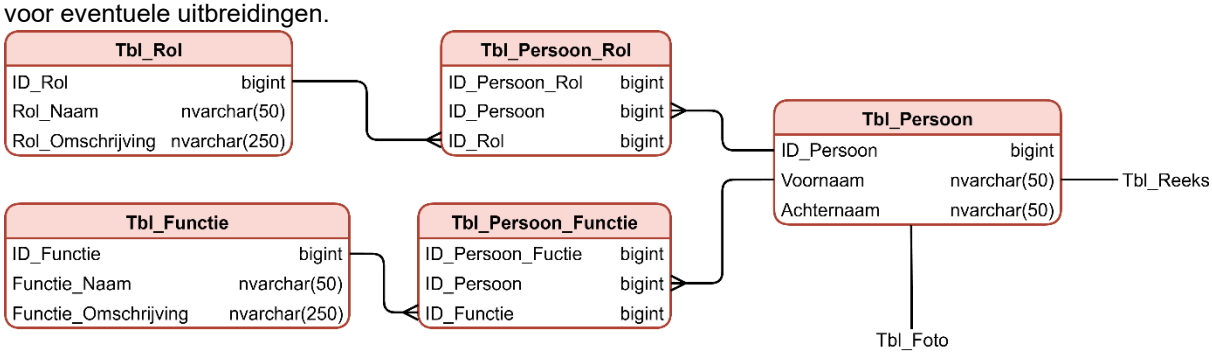
Het hoofd-ERD op de volgende pagina geeft een globaal overzicht van alle tabellen in de database. Deze database is afkomstig van eerder project, waarop ik verder gebouwd en uitgebreid heb. Het diagram toont enkel de tabelnamen en dient als een navigatiekaart om inzicht te krijgen in de onderlinge samenhang. Een volledig uitgewerkt diagram zou anders te onduidelijk worden.
Vervolgens heb ik het opgedeeld in groepen, zodat de tabellen per domein in meer detail kunnen toegelicht worden.



Figuur 3 ER diagram

3.3.2.1. Groep Persoon

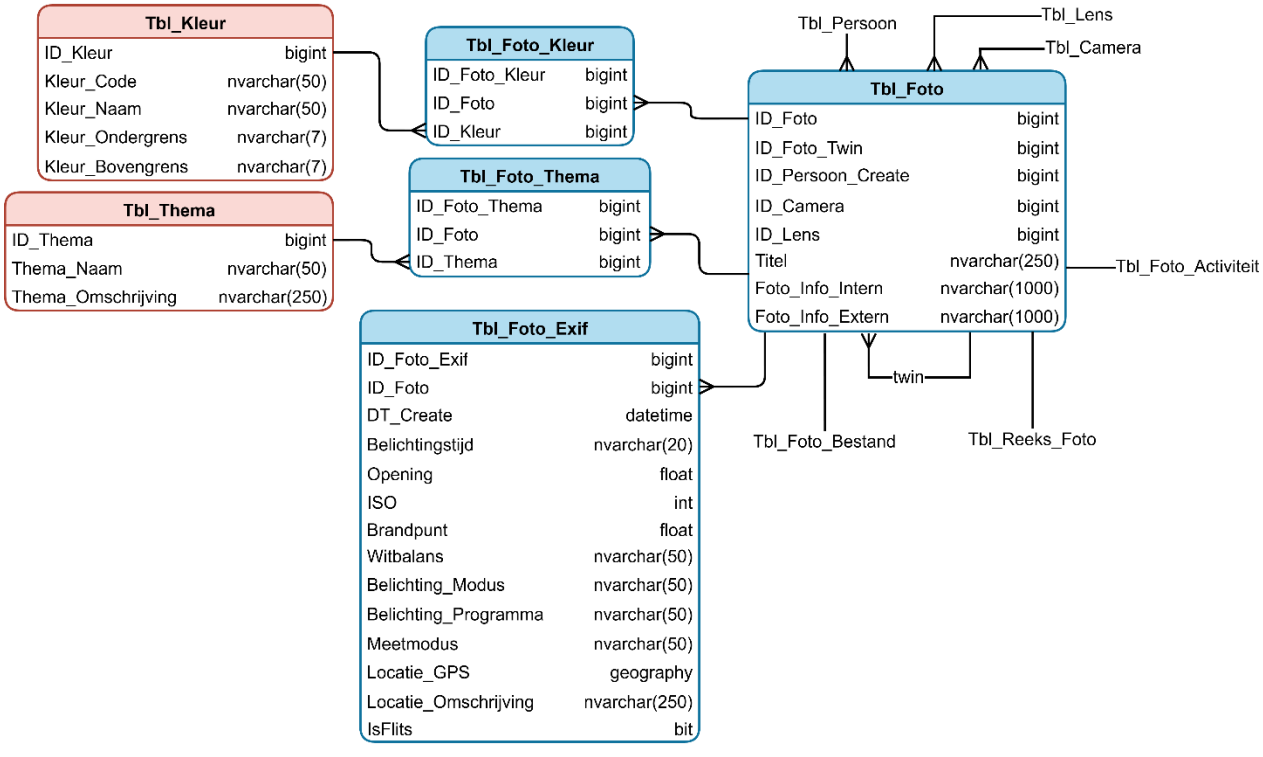
Dit onderdeel bevat de gegevens van personen, inclusief hun naam, rol en functie. De functie verwijst naar de positie die een persoon heeft binnen de fotoclub (bijvoorbeeld lid, voorzitter, …). De rol wordt gebruikt binnen de applicatie zelf. Hoewel deze momenteel nog geen actieve betekenis heeft, is dit wel interessant voor eventuele uitbreidingen.



Figuur 4 ER diagram - groep persoon

3.3.2.2. Groep Foto

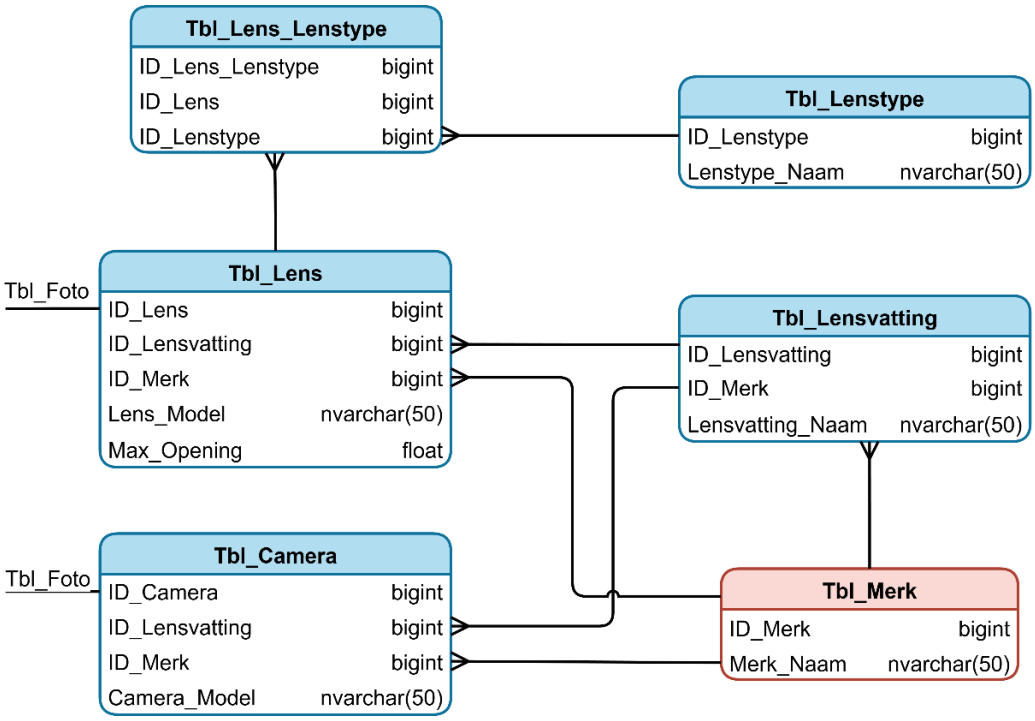
Deze groep bevat de informatie over de foto, waaronder de kleuren en thema's waartoe de foto behoort. Daarnaast wordt hier ook de EXIF-data opgeslagen, nadat deze uit het fotobestand is uitgelezen.



Figuur 5 ER diagram - groep foto

3.3.2.3. Groep Camera

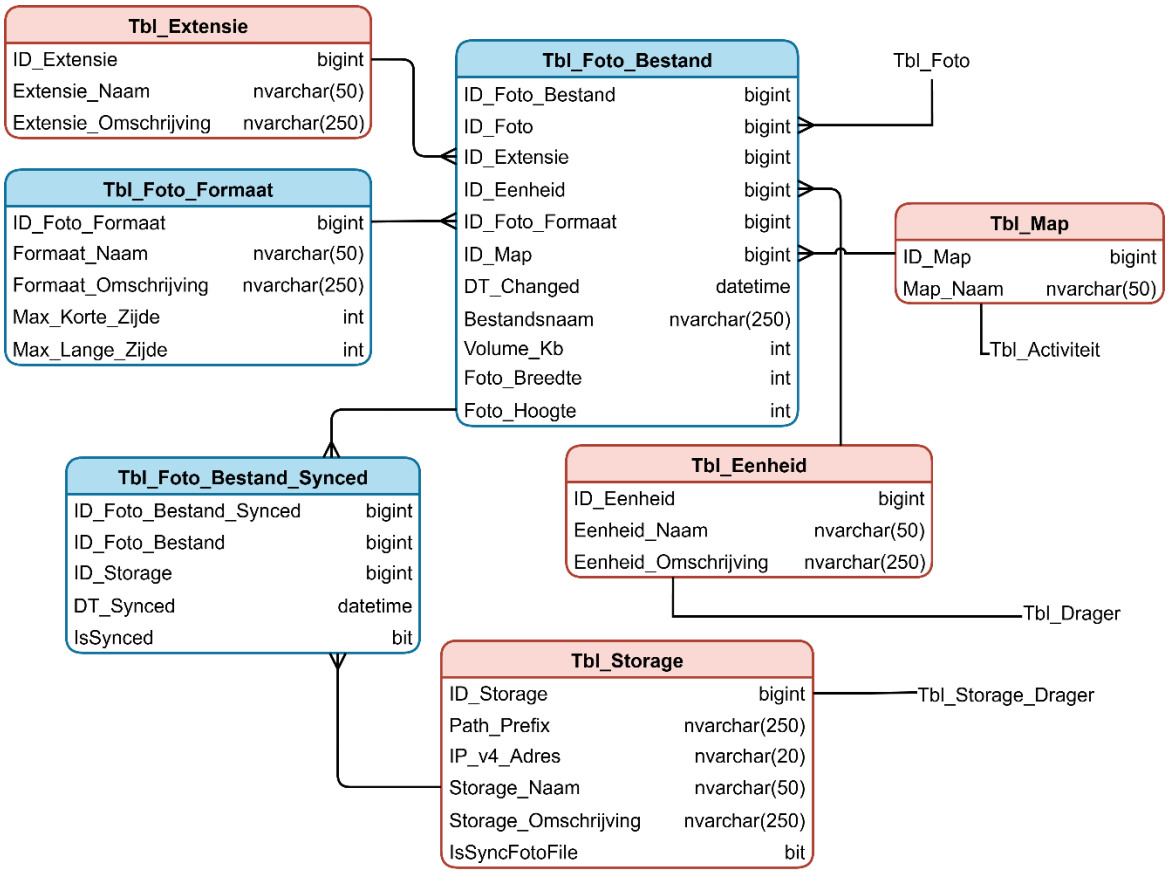
Dit onderdeel bevat aanvullende informatie over de foto. Het omvat onder andere het type camera en de lensgegevens die gebruikt werden bij het maken van de foto.



Figuur 6 ER diagram - groep camera

3.3.2.4. Groep Fotobestand

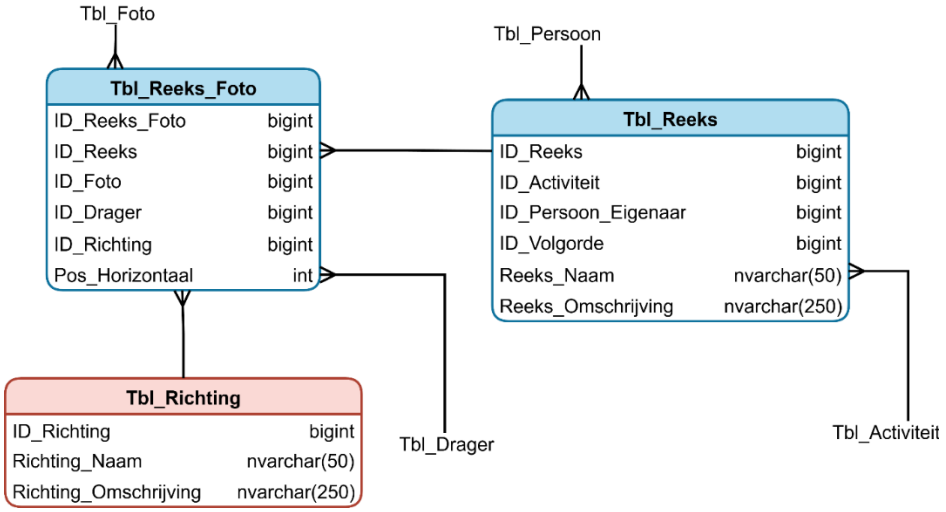
In deze groep worden de fotobestanden opgeslagen. Elk fotobestand is gekoppeld aan een map en een formaat. Na het uploaden worden de foto's automatisch omgezet naar meerdere groottes om de performantie van de applicatie te verbeteren. Daarnaast wordt hier ook de synchronisatiestatus van de fotobestanden bijgehouden voor elke storage waar dit vereist is.



Figuur 7 ER diagram - groep fotobestand

3.3.2.5. Groep Reeks

Deze groep bevat de reeksen van het systeem. Elke reeks heeft een volgorde-id, die niet gekoppeld is aan een andere tabel maar uitsluitend wordt gebruikt voor de afspeelvolgorde te bepalen. Daarnaast bestaat er een koppeling tussen de reeks en de bijhorende foto's. In deze relatie worden ook de positie van de drager en rotatie opgeslagen.

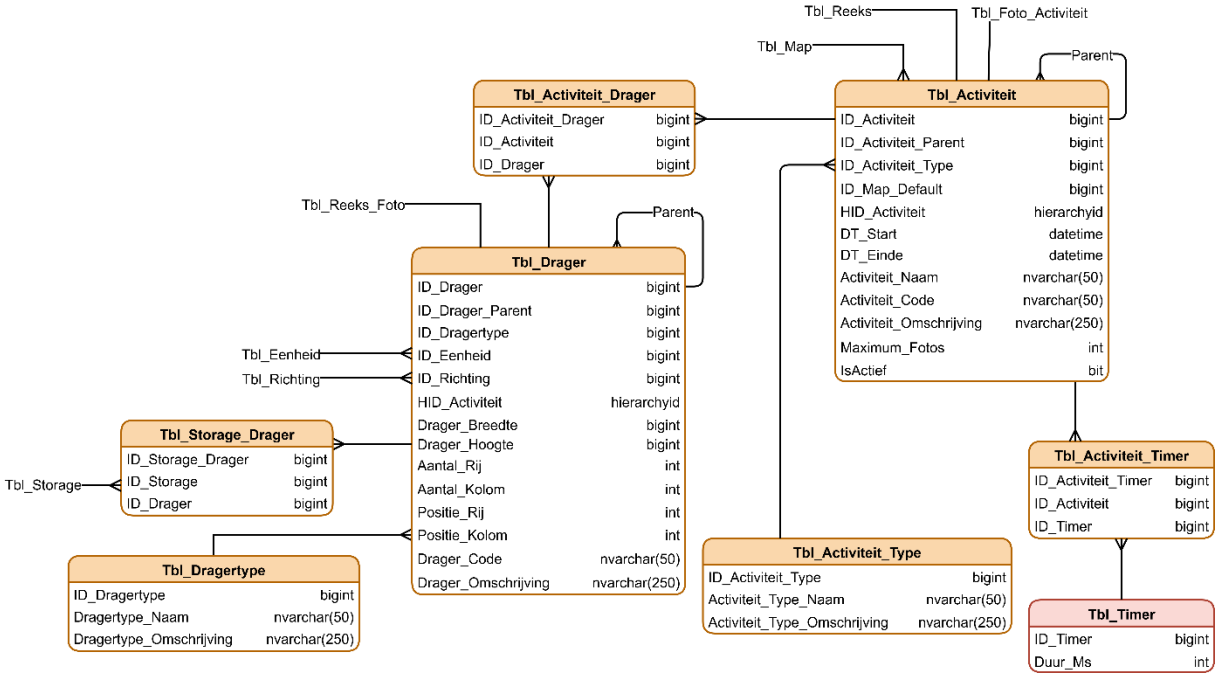


Figuur 8 ER diagram - groep reeks

3.3.2.6. Groep Activiteit

In deze groep bevinden zich de tabellen waarin zowel de activiteiten als de dragers worden opgeslagen. Een activiteit kan bijvoorbeeld de opstelling op een tentoonstelling zijn, maar ook een wekelijkse vergadering. Een drager verwijst naar een fysiek element zoals een scherm of een lichtbak.

Aan de activiteit kan een timer gekoppeld worden, waarmee de duur van het tonen van een reeks wordt bepaald.
Een drager kan bovendien gelinkt worden aan een storage-element. In deze applicatie wordt dit gebruikt om een koppeling te leggen tussen de storage (Raspberry Pi) en het fysieke scherm.



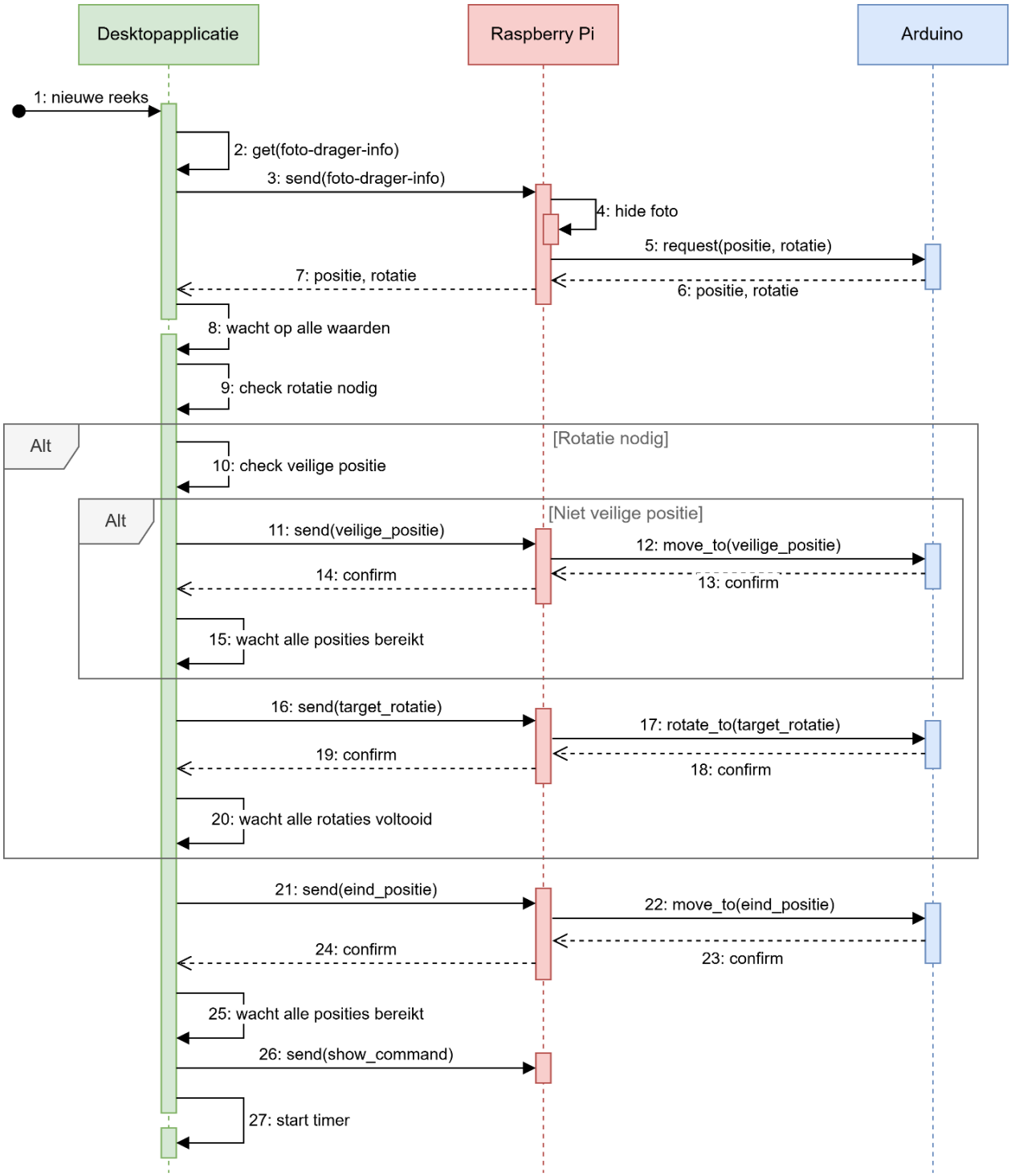
Figuur 9 ER diagram - groep activiteit

3.3.3. Sequentiediagram

Onderstaande sequentiediagram geeft een duidelijk overzicht van de volgorde van communicatie tussen de verschillende onderdelen van het systeem. Deze volgorde is essentieel om botsingen te vermijden bij het aansturen van de posities, en vooral bij de rotatie van de schermen.

Voor het deel met de Raspberry Pi en de Arduino toon ik slechts één exemplaar, hoewel dit proces vijf keer identiek wordt uitgevoerd. Om het diagram overzichtelijk te houden, heb ik de herhalingen weggelaten.

Het proces start wanneer in de desktopapplicatie een nieuwe reeks wordt geselecteerd op basis van een timer. Vervolgens worden alle positie en rotatie sturingen uitgevoerd. Zodra alle schermen hun eindpositie bereikt hebben, wordt een commando verstuurd om alle foto's te tonen. Daarna wordt een nieuwe timer gestart.



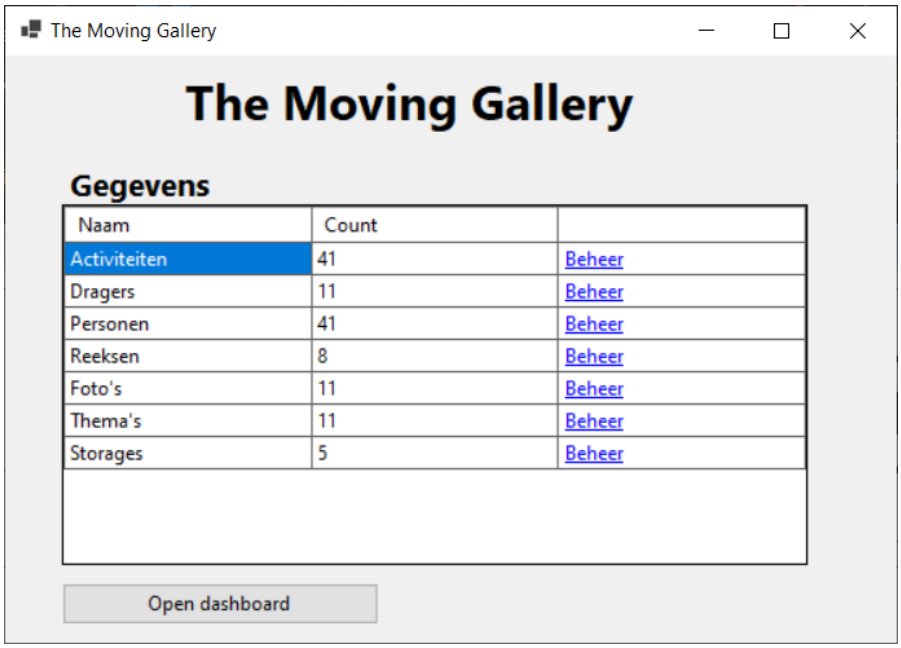
Figuur 10 Sequentiediagram

4. Realisatie

In dit hoofdstuk wordt de volledige realisatie van het project beschreven. De verschillende onderdelen - zowel softwarematig als hardwarematig - worden in detail toegelicht. Daarbij wordt uitgelegd hoe voor elk onderdeel deze uitwerking is gerealiseerd en hoe deze onderdelen samenwerken om het systeem tot één geheel te laten functioneren.

4.1. Beginscherm

In onderstaand voorbeeld is het beginscherm te zien. Dit scherm toont een overzicht van de verschillende beheerschermen, telkens aangevuld met het aantal aanwezige items. Onderaan bevindt zich een knop om het dashboard te openen, waarmee de aansturing van de schermen wordt uitgevoerd.



Figuur 11 Beginscherm

4.2. Gegevensbeheer

In dit onderdeel wordt het gedeelte beschreven waarmee de gegevens van de desktopapplicatie worden beheerd. Dit omvat voornamelijk het toevoegen en bewerken van personen, reeksen, foto's en andere entiteiten. Dit onderdeel vormt een groot deel van de desktopapplicatie en is essentieel voor de aansturing van de fysieke opstelling.

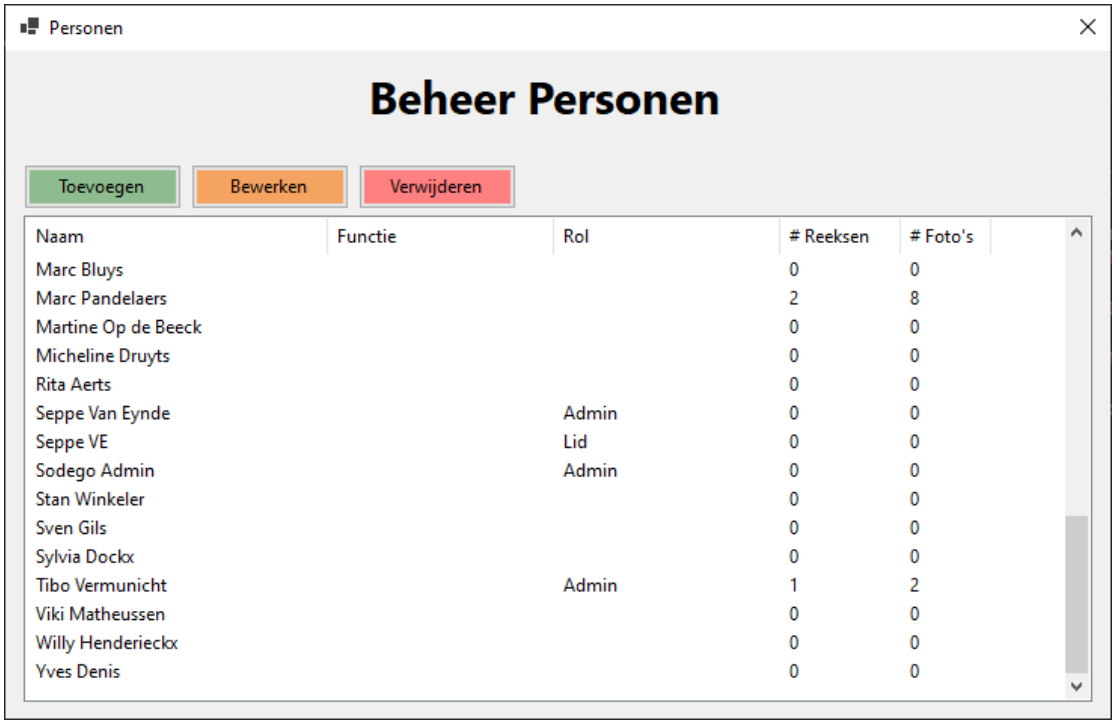
4.2.1. Personen

Voor het beheer van personen binnen de applicatie is een form voorzien met een lijst van alle personen. De lay-out van dit form wordt doorheen de volledige applicatie hergebruikt voor het weergeven van lijsten met data.

De lijst bestaat uit een ListView-component, waarbij de gegevens in tabelvorm worden gerepresenteerd. Boven de ListView bevinden zich minimaal drie knoppen:

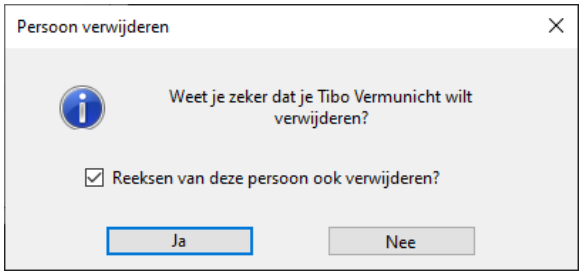
- Een knop om een nieuw item toe te voegen
- Een knop om het geselecteerde item te bewerken
- Een knop om het geselecteerde item te verwijderen.

Bij het verwijderen van een item wordt steeds een bevestigingspop-up getoond om onbedoelde acties te voorkomen.



Figuur 12 Form met lijst van personen

Bij het verwijderen van een persoon wordt er in de bevestigingspop-up nog een extra selectievakje getoond waarmee de gebruiker kan kiezen om de bijhorende foto's en reeksen van deze persoon ook te verwijderen.



Figuur 13 Pop-up bevestiging persoon verwijderen

Bij het openen van het "Beheer Personen" form, wordt de benodigde data opgehaald uit de database. Dit gebeurt via een SqlConnection waarna de gegevens in een DataTable worden geplaatst. Vervolgens worden de rijen van deze DataTable uitgelezen en gebruikt om hiermee de correcte informatie in de ListView weer te geven.

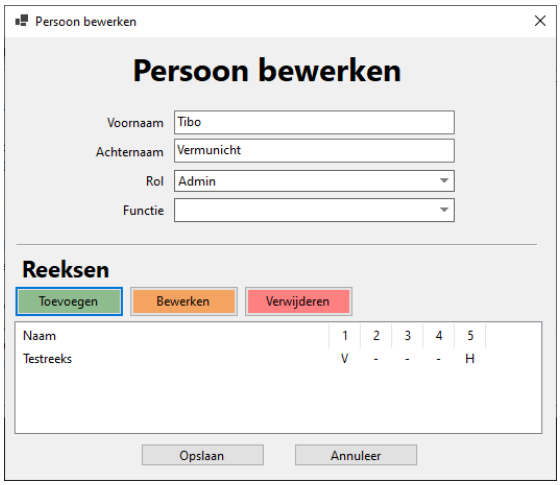
```
Dim DT_Personen As New DataTable()
Using conn As New SqlConnection(Cnst_DB_Connection)
    ' Database list alle personen en plaats in datatable
    Dim sql As String = "
       SELECT p.ID_Persoon,
              CONCAT(p.Voornaam, ' ', p.Achternaam) AS Naam,
              STRING_AGG(r.Rol_Naam, ', ') AS Rol,
              STRING_AGG(f.Functie_Naam, ', ') AS Functie,
              (SELECT COUNT(*) FROM foto.Tbl_Foto fp WHERE fp.ID_Persoon_Create=p.ID_Persoon AND fp.IsDele=0) AS Fotos,
              (SELECT COUNT(*) FROM foto.Tbl_Reeks re WHERE re.ID_Persoon_Eigenaar=p.ID_Persoon AND re.IsDele=0) AS Reeksen
       FROM Tbl_Persoon p
       LEFT JOIN Tbl_Persoon_Rol pr ON p.ID_Persoon=pr.ID_Persoon AND pr.IsDele=0
       LEFT JOIN Tbl_Rol r ON pr.ID_Rol=r.ID_Rol AND r.IsDele=0
       LEFT JOIN Tbl_Persoon_Functie pf ON p.ID_Persoon=pf.ID_Persoon AND pf.IsDele=0
       LEFT JOIN Tbl_Functie f ON pf.ID_Functie=f.ID_Functie AND f.IsDele=0
       WHERE p.IsDele=0
       GROUP BY p.ID_Persoon, p.Voornaam, p.Achternaam
       ORDER BY Naam"

    Using cmd As New SqlCommand(sql, conn)
        Dim DA_Personen As New SqlDataAdapter(cmd)
        DA_Personen.Fill(DT_Personen)
    End Using
End Using

' Maak voor elke persoon een rij aan met juiste informatie
LV_Personen.Items.Clear()
For Each row As DataRow In DT_Personen.Rows
    Dim Str_Naam As String = row("Naam")
    Dim LV_Item = New ListViewItem(Str_Naam) With {
                      .Tag = row("ID_Persoon")
                  }
    LV_Item.SubItems.Add(row.GetStringSafe("Functie"))
    LV_Item.SubItems.Add(row.GetStringSafe("Rol"))
    LV_Item.SubItems.Add(row("Reeksen"))
    LV_Item.SubItems.Add(row("Fotos"))
    LV_Personen.Items.Add(LV_Item)
Next
```

Figuur 14 Code persoon data uit database halen

Het form voor het toevoegen van een persoon bevat een TableLayoutPanel met daarin een TextBox is voor het invoeren van de voor- en achternaam. Voor de selectie van de functie en rol van de persoon werd een eigen component ontwikkeld, aangezien WinForms standaard geen combobox aanbiedt die meerdere selecties ondersteunt.

Figuur 15 Form persoon toevoegen

In het form voor het bewerken van een persoon wordt onderaan een lijst weergegeven met alle reeksen die aan deze persoon zijn gekoppeld. De manier waarop deze wordt weergegeven, is hetzelfde als is in het form voor beheer van reeksen, wat later in dit document wordt toegelicht.

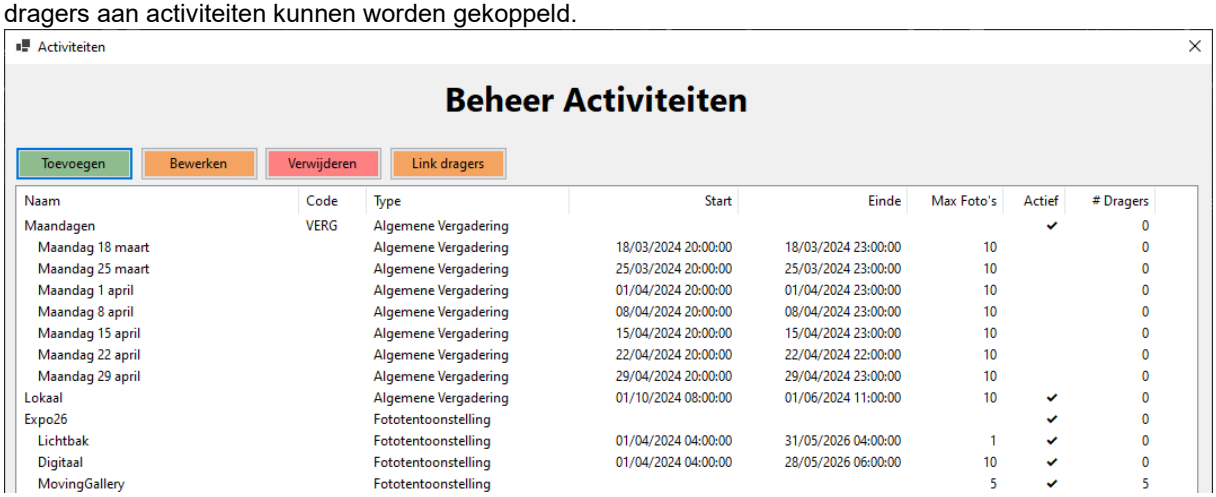


Figuur 16 Form persoon bewerken

De zelfgemaakte comboboxcomponent bestaat uit een label dat de geselecteerde waarden weergeeft, een knop om de dropdown te openen en een CheckedList-component die als dropdown wordt weergegeven. In deze checklist worden alle beschikbare opties getoond, elk als een checkbox.

4.2.2. Activiteiten

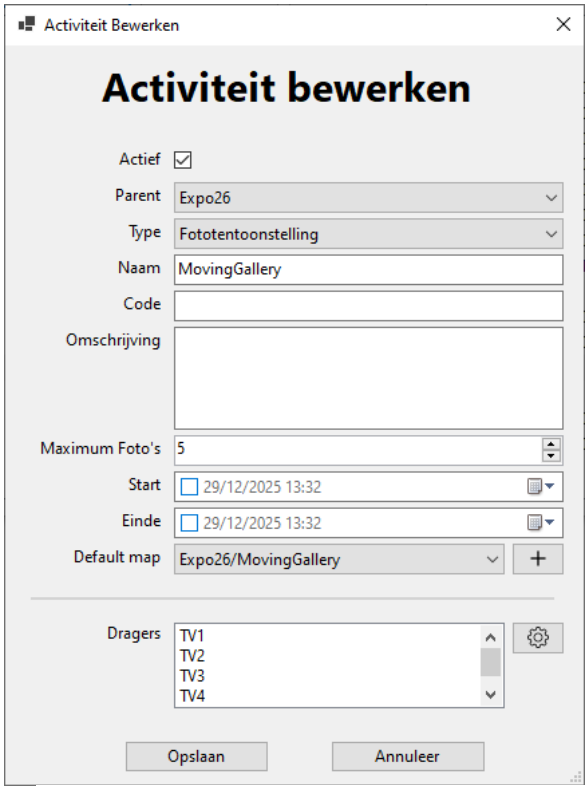
Het form “Beheer Activiteiten” volgt dezelfde structuur als het form “Beheer Persoon”. Het verschil is dat in de activiteitenlijst een visuele inspringing wordt toegepast om de hiërarchie tussen de activiteiten weer te geven. Deze inspringing wordt bepaald door de HID-property in de database, die het niveau van de parent-childrelatie aangeeft.
In de lijst toont ook de start- en einddatum van de activiteit alsook het aantal dragers er aan deze activiteit gekoppeld zijn.
Daarnaast is er ook een extra knop “Link dragers” aanwezig. Deze opent een afzonderlijk form waarin dragers aan activiteiten kunnen worden gekoppeld.



Figuur 17 Form beheer activiteiten

Het form voor het aanmaken en bewerken van een activiteit zijn vrijwel identiek. Het enige verschil is dat bij het bewerken onderaan een lijst wordt weergegeven met de dragers die aan de activiteit zijn gekoppeld.
Naast de dropdownlijst voor het selecteren van de standaardmap bevindt zich een knop om een nieuwe map toe te voegen wanneer de gewenste waarde nog niet beschikbaar is.

In deze forms wordt ook de parent-activiteit geselecteerd. Op basis van deze selectie wordt bij het opslaan in de database een correct hiërarchie-ID gegenereerd. Wanneer een activiteit van parent verandert of wanneer een parent wordt verwijderd, worden alle HID-waarden van de onderliggende children opnieuw berekend. Dit wordt gedaan via de onderstaande SQL-query, die wordt uitgevoerd voor elke activiteit die moet worden bijgewerkt.



Figuur 18 Form activiteit bewerken

```
DECLARE @hid_parent hierarchyid;
DECLARE @hid_item hierarchyid;

-- Step 1: Get parent HID or root
IF @id_parent IS NULL
    SET @hid_parent=hierarchyid::GetRoot();
ELSE
    SELECT @hid_parent=HID_Activiteit FROM activiteit.Tbl_Activiteit WHERE
ID_Activiteit=@id_parent;

-- Step 2: Try to generate new HID under parent
SELECT @hid_item = @hid_parent.GetDescendant(MAX(HID_Activiteit), NULL)
FROM activiteit.Tbl_Activiteit
WHERE HID_Activiteit.GetAncestor(1)=@hid_parent;

-- Step 3: Fallback if no siblings exist
IF @hid_item IS NULL SET @hid_item=@hid_parent.GetDescendant(NULL, NULL);

-- Step 4: Update item
UPDATE activiteit.Tbl_Activiteit
SET HID_Activiteit=@hid_item, ID_Activiteit_Parent=@id_parent
WHERE ID_Activiteit=@id;

-- Step 5: Update children
UPDATE activiteit.Tbl_Activiteit
SET HID_Activiteit=@hid_item.GetDescendant(NULL, NULL)
WHERE ID_Activiteit_Parent=@id;
```

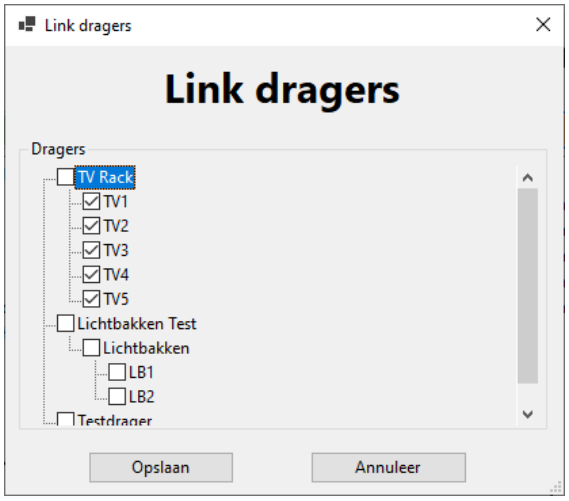
Figuur 19 SQL-query HID aanpassen

Om in de applicatie gegevens uit een DataTable in een combobox te tonen, wordt deze DataTable rechtstreeks gekoppeld aan de datasource van de combobox. Hierdoor is het niet nodig om een aparte lijst te maken of de items handmatig toe te voegen. Onderstaand zie je een voorbeeld van hoe dit wordt gedaan. Ook zie je de functie om een leeg item toe te voegen aan het begin van de DataTable zodat de optie om niets te selecteren mogelijk is.

```
ComboAddEmpty(DT_Activiteiten, "ID_Activiteit", "Naam")
Combo_Parent.DisplayMember = "Naam"
Combo_Parent.ValueMember = "ID_Activiteit"
Combo_Parent.DataSource = DT_Activiteiten

Public Sub ComboAddEmpty(ByRef dt As DataTable, idKolom As String, naamKolom As String)
    Dim legeRij As DataRow = dt.NewRow()
    legeRij(idKolom) = DBNull.Value
    legeRij(naamKolom) = ""
    dt.Rows.InsertAt(legeRij, 0)
End Sub
```

Figuur 20 Code DataTable binden aan combobox

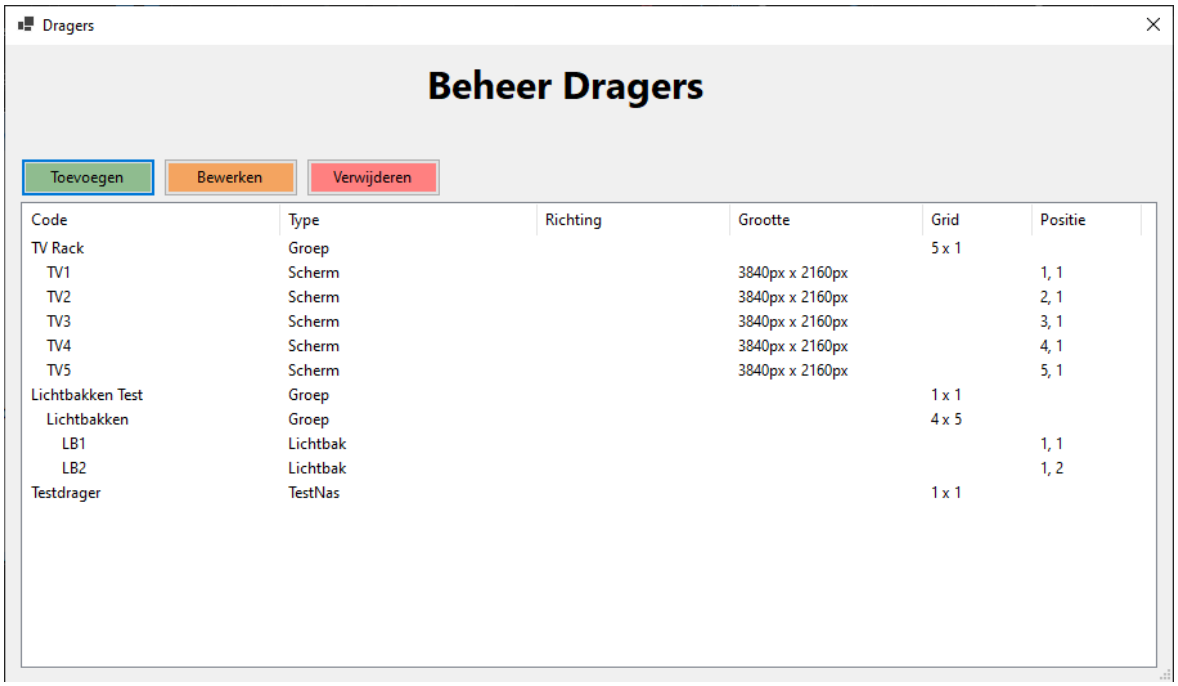
Om de koppeling tussen dragers en een activiteit te beheren, wordt in het form een treeview-lijst getoond met alle dragers. Via een checkbox kan de gebruiker selecteren welke dragers gekoppelde moeten worden. Wanneer een drager wordt geselecteerd die zelf children heeft, worden deze automatisch mee geselecteerd.



Figuur 21 Form link dragers aan activiteit

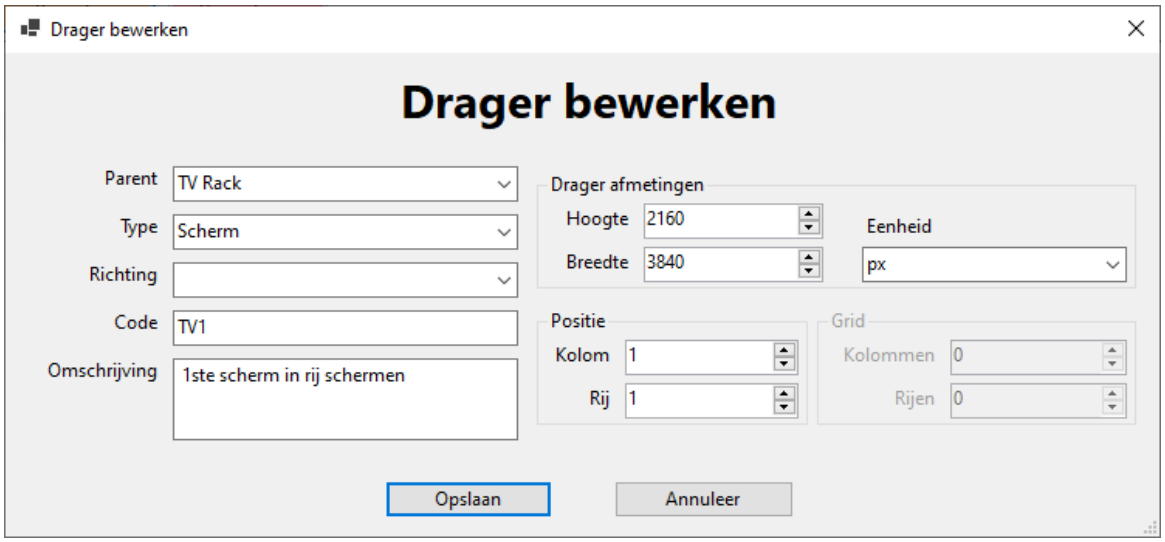
4.2.3. Dragers

Het form “Beheer Dragers” gebruikt in de lijst dezelfde visuele inspringing als bij de activiteiten, zodat de hiërarchie tussen dragers duidelijk zichtbaar is. Daarnaast wordt voor elke drager ook de grootte weergegeven, wat zowel de fysieke afmetingen als de resolutie kan zijn.
Wanneer een drager een parent is, bevat deze doorgaans een grid. In deze grid krijgt elke child-drager dan een specifieke positie toegewezen.



Figuur 22 Form beheer dragers

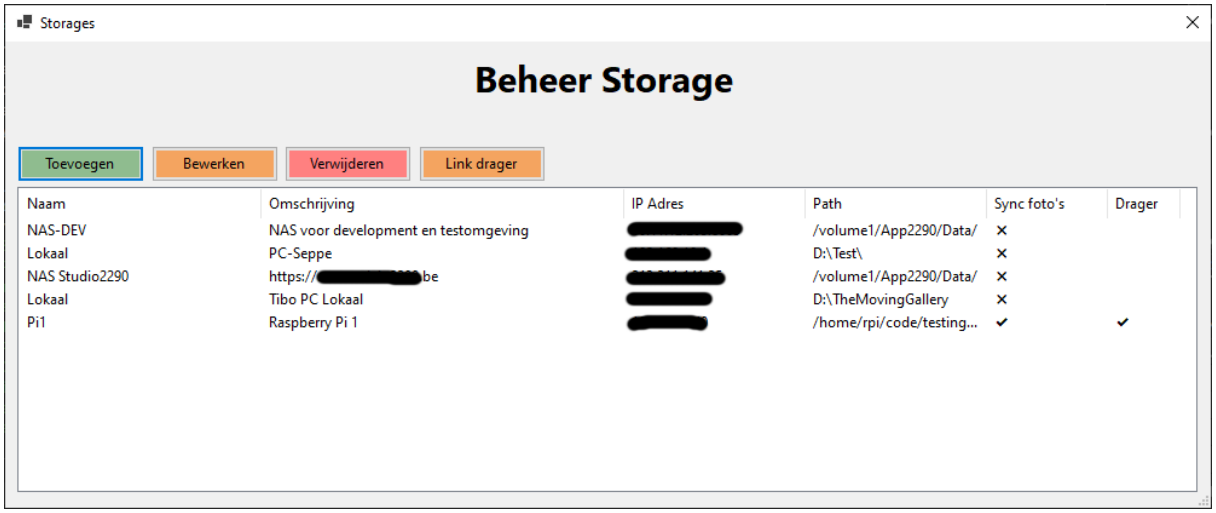
Wanneer de gebruiker een drager wil toevoegen of bewerken (beide forms zijn identiek), kunnen de afmetingen worden ingevoerd, op voorwaarde dat er een eenheid gekozen wordt. Als het dragertype een scherm of lichtbak is, dan kan deze geen grid bevatten. De grid-instellingen worden in dit geval automatisch uitgeschakeld en zijn niet beschikbaar om in te vullen, zoals zichtbaar is in het onderstaand voorbeeld.



Figuur 23 Form drager bewerken

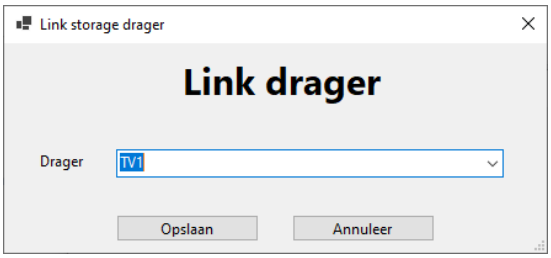
4.2.4. Storage

Het beheer van storage binnen de applicatie is voornamelijk bedoeld om de opslagplaats voor de foto's te configureren. Aan elke storage-configuratie zijn een IP-adres en een pad gekoppeld. Daarnaast is er een optie voorzien om ervoor te zorgen dat de foto's op de ingestelde storage automatisch worden gesynchroniseerd.
Verder kan een storage gekoppeld worden aan een drager. In deze opstelling is dat nodig om aan te geven welke Raspberry Pi aan welke drager verbonden is.



Figuur 24 Form beheer storage

Voor het form om de storage aan drager te koppelen is een simpele combobox voorzien aangezien er maar één actieve koppeling mogelijk is.



Figuur 25 Form link drager aan storage

4.2.5. Reeksen

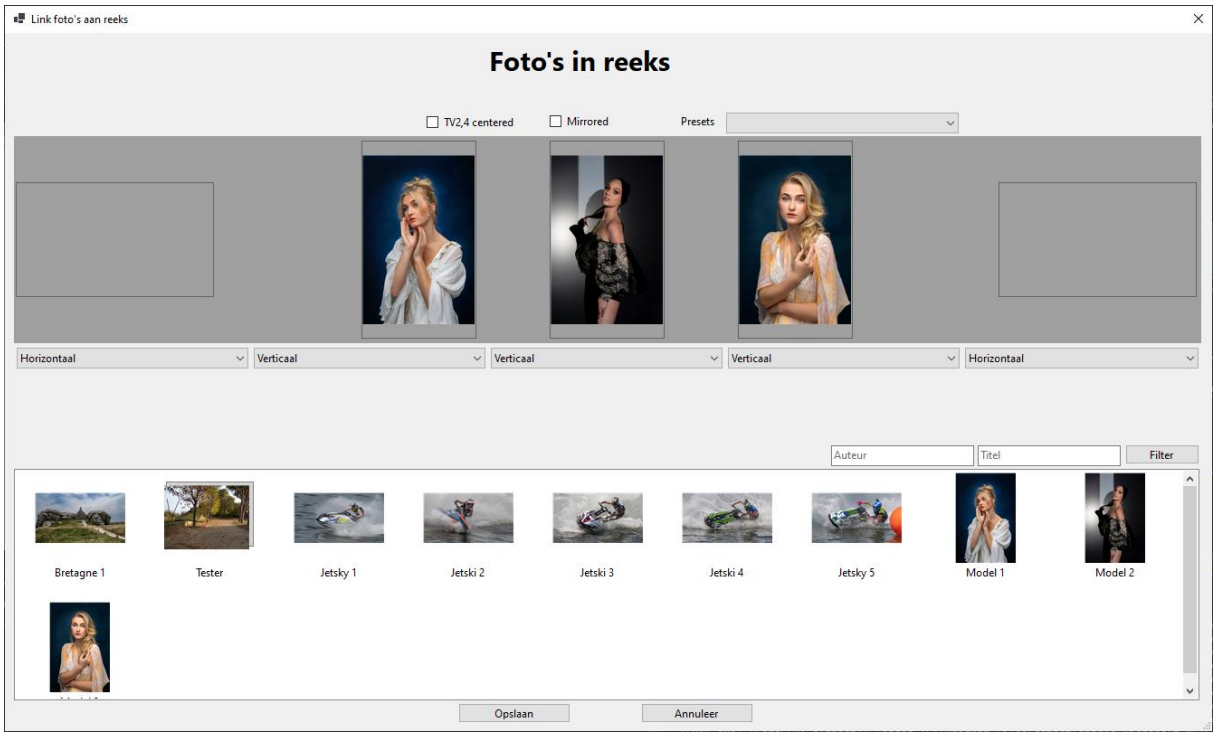
Bij het form voor het beheer van de reeksen, zoals in onderstaand voorbeeld, wordt een lijst weergegeven met de naam van de reeks en de auteur(s). Daarnaast wordt ook de opstelling en gebruik van de vijf mogelijke dragers binnen een reeks getoond. Een V duidt een verticale oriëntatie aan, een H een horizontale oriëntatie, en – geeft aan dat de drager geen foto bevat. Dit overzicht biedt een duidelijk beeld om de juiste reeksvolgorde samen te stellen.
Het is ook mogelijk om de reeksen van volgorde van de reeksen aan te passen via drag-and-drop. Nadien moet de aangepaste volgorde worden opgeslagen, of kan deze worden gereset wanneer de oorspronkelijke volgorde gewenst is.

Naam	Auteur(s)	1	2	3	4	5
Bretagne	Dirk Broeckx	H	H	H	H	H
Testreeks	Tibo Vermunicht	V	-	-	-	H
Jetski's	Marc Pandelaers	H	H	H	H	H
Model	Marc Pandelaers	-	V	V	V	-
Koningsee	Dirk Broeckx	H	H	H	H	H
Abstrakt	Geert Van Aelst	V	V	H	V	V
Astrofotografie	Bram Goossens	H	H	H	H	H
Rotterdam	Geert Van Aelst	V	H	V	H	V

Figuur 26 Form beheer reeksen

Het koppelen van foto's aan een reeks gebeurt via een form waarin de vijf dragers die de opstelling vormen zichtbaar zijn. Onderaan bevindt zich een lijst met alle beschikbare foto's. De gebruiker kan de gewenste foto's vanuit deze lijst naar de juiste drager slepen. De lijst kan bovendien worden gefilterd op basis van de titel of de auteur van de foto.
Bovenaan zijn opties aanwezig voor de positiebepaling van de schermen. De gebruiker kan kiezen uit vooraf ingestelde presets, of de schermen handmatig op de gewenste positie plaatsen. Deze positie wordt opgeslagen en vervolgens gereflecteerd in de fysieke opstelling op de tentoonstelling.

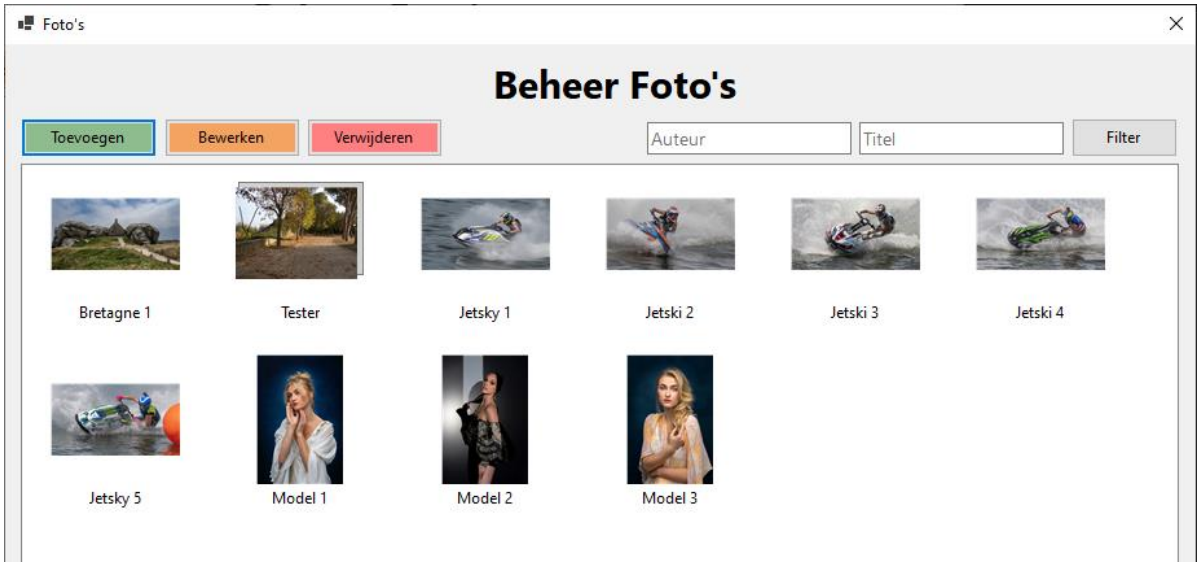
Voor het tonen van de schermen en de mogelijkheid van positionering werd de standaard picturebox uitgebreid zodat deze de schermen op schaal weergeeft ten opzichte van het paneel waarin ze zich bevinden. Ook is er een functie aanwezig die kan controleren op overlapping van twee pictureboxen en deze corrigeren indien nodig. Daarnaast wordt de positie ook bijgehouden op een schaal van -1000 tot 1000 en deze positiewaarde is dus onafhankelijk van de grootte van het scherm.



Figuur 27 Form foto's in reeks plaatsen

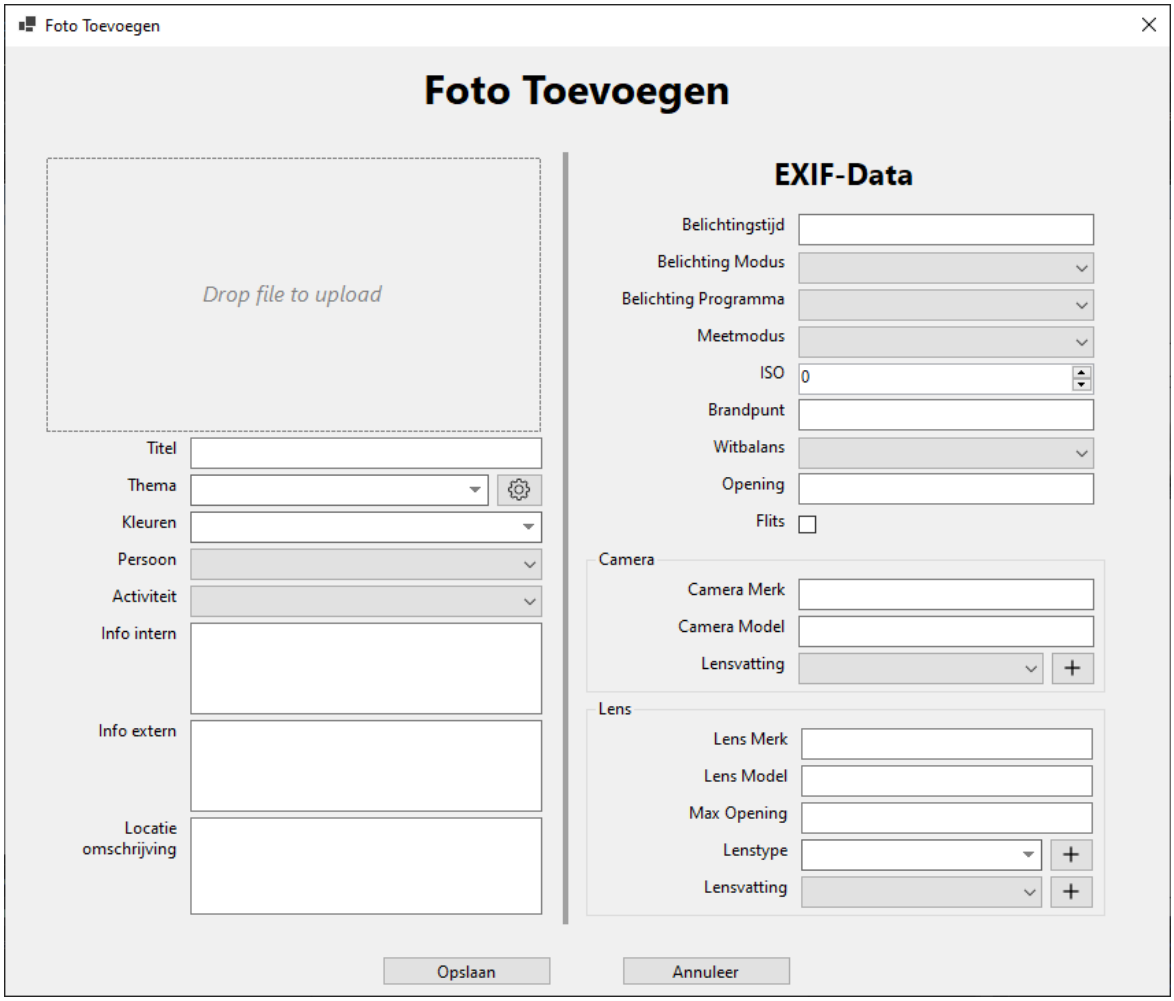
4.2.6. Foto's

Het form voor het beheer van de foto's ziet er iets anders uit dan de andere beheerschermen. In plaats van een lijst worden de foto's weergegeven als thumbnails, telkens met bijhorende titel onderaan. Wanneer er meerdere versies van een foto bestaan, worden deze getoond in een stacked-weergave, waarbij de recentste versie vooraan staat (zoals zichtbaar in het voorbeeld bij foto 'Tester').
Bovenaan bevinden zich tekstvakken waarmee de gebruiker kan filteren op basis van de auteur of de titel van de foto.



Figuur 28 Form beheer van foto's

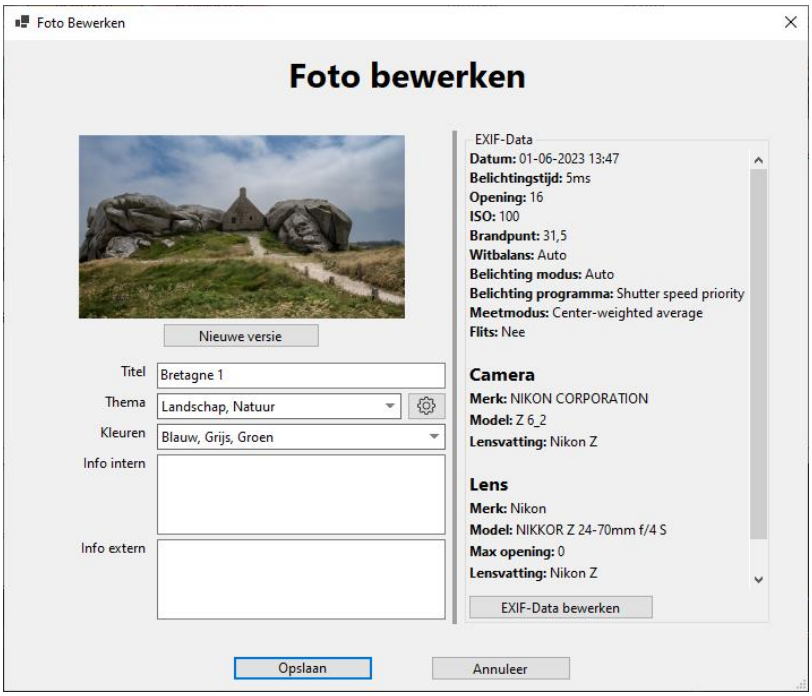
Als de gebruiker een foto wil toevoegen, kan dit door een lokaal opgeslagen fotobestand naar het invoervak te slepen. Vervolgens wordt de EXIF-data van de foto uitgelezen en automatisch ingevuld aan de rechterkant van het form. Voor zowel de camera als de lens wordt eerst gecontroleerd of er al een overeenkomstige entry in de database bestaat. De gebruiker kan deze gegevens dan nog aanvullen of corrigeren.
De info intern dient om de gebruiker toelichting of opmerkingen over de foto te laten toevoegen. De info extern is de info die wordt getoond op het infoscherm tijdens de tentoonstelling.



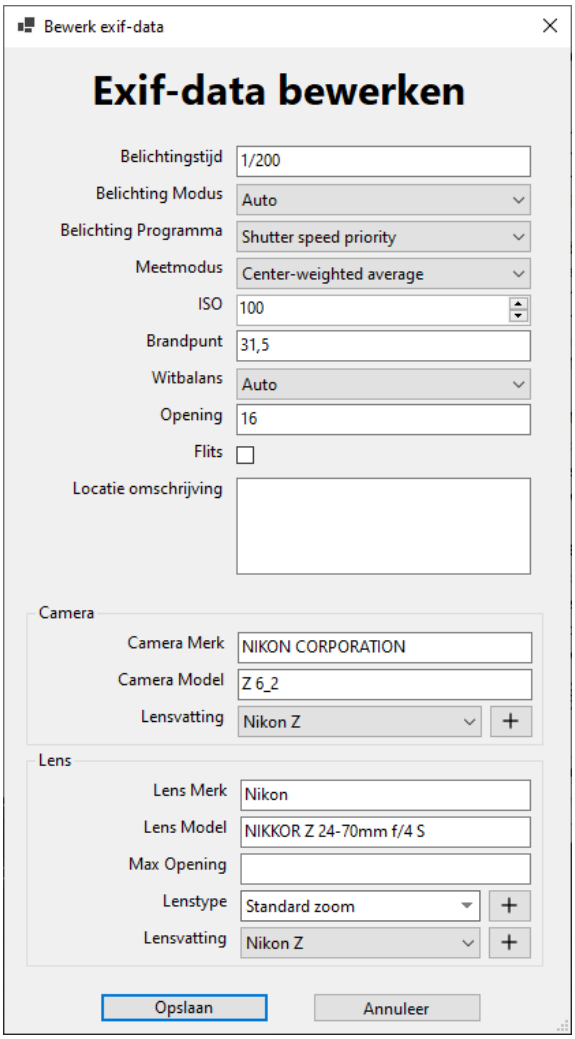
Figuur 29 Form foto toevoegen

Wanneer de gebruiker de informatie van een foto wil bewerken, kan dit via het form “Foto bewerken” form. In dit form kan enkel de basisinformatie van de foto worden aangepast. Aan de rechterkant wordt de EXIF-data van de foto weergegeven.

Als de gebruiker deze EXIF-data willen wijzigen, kan dit via een extra form dat verschijnt na het klikken op de daarop voorziene knop.

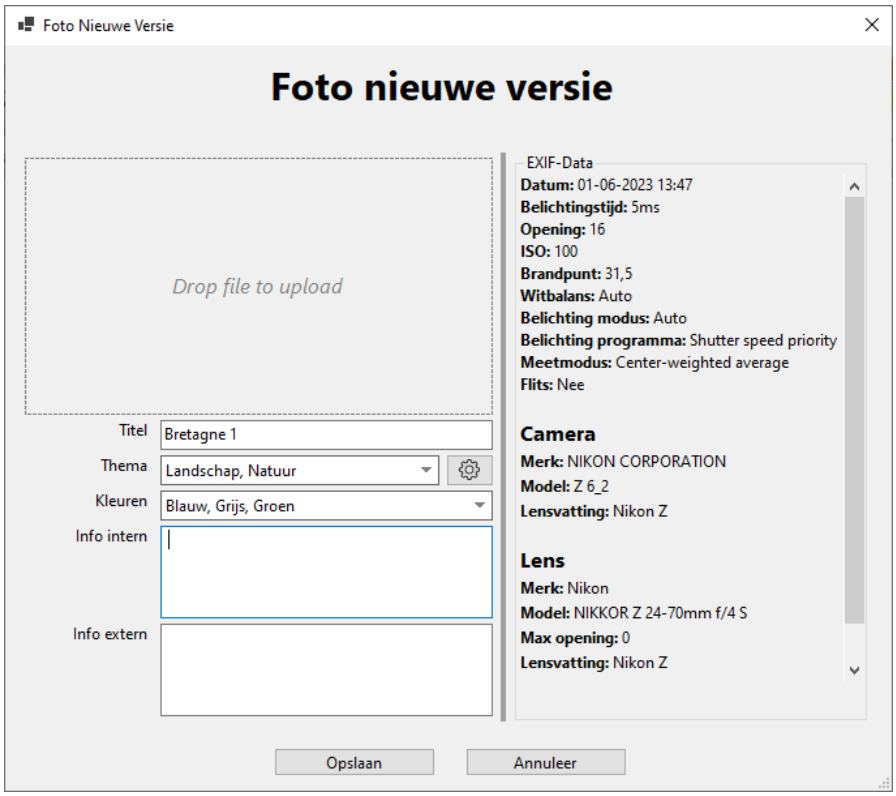


Figuur 30 Form foto bewerken



Figuur 31 Form EXIF-data bewerken

Wanneer de gebruiker een nieuwe versie van de foto wil uploaden kan men dit via het form dat hieronder zichtbaar is. Hier kan de gebruiker dan een nieuw bestand uploaden en de basisinfo van de foto aanpassen. De EXIF-data worden gekopieerd van de vorige versie. Bij het opslaan zal deze foto automatisch de vorige versie vervangen in de reeksen waarin deze voorkomt.



Figuur 32 Form nieuwe versie van foto

4.3. Synchronisatie foto's

Aangezien de foto's lokaal aanwezig moeten zijn op alle Raspberry Pi's die bij de schermen geplaatst worden, is het noodzakelijk om een manier te hebben waarop deze gesynchroniseerd kunnen worden en deze synchronisatiestatus kan bijgehouden worden.

Deze realisatie bestaat uit twee onderdelen. Het eerste onderdeel is een RESTful API die op elke Raspberry Pi draait. Deze API is ontwikkeld met behulp van de FastAPI-library, waardoor het opzetten van een efficiënte en uitbreidbare API aanzienlijk wordt vereenvoudigd.

De API die hieronder zichtbaar is bevat de volgende endpoints:

- GET /ping
  Stuurt het antwoord "pong" terug. Dit endpoint wordt gebruikt om de verbinding tussen de desktopapplicatie en de API te testen.

- GET /images
  Accepteert een path_prefix als parameter en doorzoekt dit pad recursief naar alle fotobestanden met de ondersteunde extensies. Het resultaat bestaat uit een lijst met de paden van alle gevonden fotobestanden, samen met een totaalcount.

- PUT /upload
  Deze functie wordt gebruikt om fotobestanden vanuit de desktopapplicatie naar de Raspberry Pi te uploaden. Deze foto wordt opgeslagen in het opgegeven pad. Als de map nog niet bestaat, wordt deze automatisch aangemaakt.

- DELETE /image
  Dit endpoint wordt gebruikt om een bestand op de Raspberry Pi te verwijderen. Deze wordt gebruikt bij de initiële synchronisatie om onnodige fotobestanden te verwijderen.

```python
@app.get("/ping")
def pong():
    logger.info(f"Ping pong")
    return {"ping": "pong!"}

@app.get("/images")
async def read_images(path_prefix: str) -> ListResponse:
    files = []
    for ext in ["png", "jpg", "jpeg"]:
        files += glob(f"{path_prefix}/**/*.{ext}", recursive=True)
    logger.info(f"Listed {len(files)} files from {path_prefix}")
    return {"images": files, "count": len(files)}

@app.put("/upload")
async def upload_image(data: Annotated[ImageUpload, Depends()]) -> UploadResponse:
    upload_path = Path(data.path)
    try:
        upload_path.mkdir(parents=True, exist_ok=True)
        file_path = upload_path / data.filename
        with open(file_path, "wb") as buf:
            copyfileobj(data.image.file, buf)

        logger.info(f"Uploaded file {data.filename} to {file_path}")
        return {"filename": data.filename, "saved_to": str(file_path)}
    except Exception as e:
        logger.error(f"Error saving file {data.filename} to {upload_path}: {e}")
        return {"error": str(e)}

@app.delete("/image")
async def delete_image(path: str, filename: str) -> DeleteRespone:
    file_path = Path(path) / filename

    if not file_path.exists():
        logger.warning(f"Delete failed, file not found: {file_path}")
        return {"deleted": False, "reason": "File not found"}
    try:
        file_path.unlink()
        logger.info(f"Deleted file {file_path}")
        return {"deleted": True, "filename": filename, "path": str(file_path)}
    except Exception as e:
        logger.error(f"Error deleting file {file_path}: {e}")
        return {"deleted": False, "reason": str(e)}
```

Figuur 33 FastAPI synchronisatie foto's

Het tweede onderdeel van de synchronisatie is de SyncManager-klasse in de desktopapplicatie, die verantwoordelijk is voor het volledige synchronisatieproces. Deze SyncManager haalt uit de database alle storages op waarvoor foto's gesynchroniseerd moeten worden. Voor elke storage wordt vervolgens geprobeerd een verbinding op te zetten. Wanneer deze verbinding niet tot stand komt, of de verbinding wegvalt, wordt op vaste intervallen automatisch een nieuwe poging ondernomen.
Zodra de verbinding succesvol is, wordt er een initiële synchronisatie uitgevoerd. Hierbij vraagt de SyncManager eerst een lijst op van alle fotobestanden die zich op de Raspberry Pi bevinden. Daarnaast wordt uit de database een lijst opgehaald met alle fotobestanden en hun eventuele synchronisatiestatus.

Deze twee lijsten worden met elkaar vergeleken. Hieruit worden dan de nodige bestanden geüpload of verwijderd van de Raspberry Pi. Tot slot worden alle statussen in de database bijgewerkt.

```
Private Async Function InitialSync(storage As Sync_Storage, token As CancellationToken) As Task
    Try
        Dim url As String = $"http://{storage.IP_Adres}:8000/images"
        Dim query As String = $"?path_prefix={Uri.EscapeDataString(storage.Path_Prefix)}"
        Dim response As HttpResponseMessage = Await http.GetAsync(url + query, Nothing, token)

        If response.IsSuccessStatusCode Then
            storage.IsOnline = True
            Dim json = Await response.Content.ReadAsStringAsync(token)
            Dim doc = JsonDocument.Parse(json)

            Dim count = doc.RootElement.GetProperty("count").GetInt32()
            Dim imageList As New List(Of String)
            For Each item In doc.RootElement.GetProperty("images").EnumerateArray()
                imageList.Add(item.GetString())
            Next

            Await CompareImages(storage, imageList)
            storage.HasInitialSynced = True
        Else
            storage.IsOnline = False
        End If
    Catch ex As Exception
        storage.IsOnline = False
    End Try
End Function

Private Async Function CompareImages(storage As Sync_Storage, fileList As List(Of String)) As Task
    Dim fsSet As New HashSet(Of String)(fileList.Select(AddressOf NormalizePath), StringComparer.OrdinalIgnoreCase)
    Dim DB_Files As List(Of DB_Filepath) = GetDbFiles(storage)

    Using conn As New SqlConnection(Cnst_DB_Connection)
        conn.Open()
        For Each file As DB_Filepath In DB_Files
            Dim norm_fp As String = NormalizePath(file.File_Path)
            If file.ID_Bestand Is Nothing OrElse file.ID_Bestand = 0 Then
                If file.ID_Sync IsNot Nothing AndAlso file.ID_Sync <> 0 Then
                    UpdateSyncedStatus(file.ID_Sync, True, True) ' delete sync status
                End If
                If fsSet.Contains(norm_fp) Then
                    Await DeleteImage(storage, file) ' REMOVE api call
                End If
            Else
                If Not fsSet.Contains(norm_fp) Then
                    Dim response = Await UploadImage(storage, file) ' UPLOAD api call
                    If file.ID_Sync Is Nothing OrElse file.ID_Sync = 0 Then
                        InsertSyncedStatus(file.ID_Bestand, storage.ID_Storage, response.StatusCode = 200)
                    Else
                        UpdateSyncedStatus(file.ID_Sync, response.StatusCode = 200)
                    End If
                ElseIf Not file.IsSynced Then
                    UpdateSyncedStatus(file.ID_Sync, True) ' update sync complete status
                End If
            End If
        Next
    End Using
End Function
```

Figuur 34 Code initiële synchronisatie

Na de initiële synchronisatie wordt op vaste intervallen gecontroleerd of er in de database fotobestanden aanwezig zijn waar de synchronisatie nog niet voltooid is, of waarvoor een nieuwere versie beschikbaar is. Voor deze bestanden wordt opnieuw een synchronisatiepoging uitgevoerd. Hierna wordt de synchronisatiestatus in de database bijgewerkt.

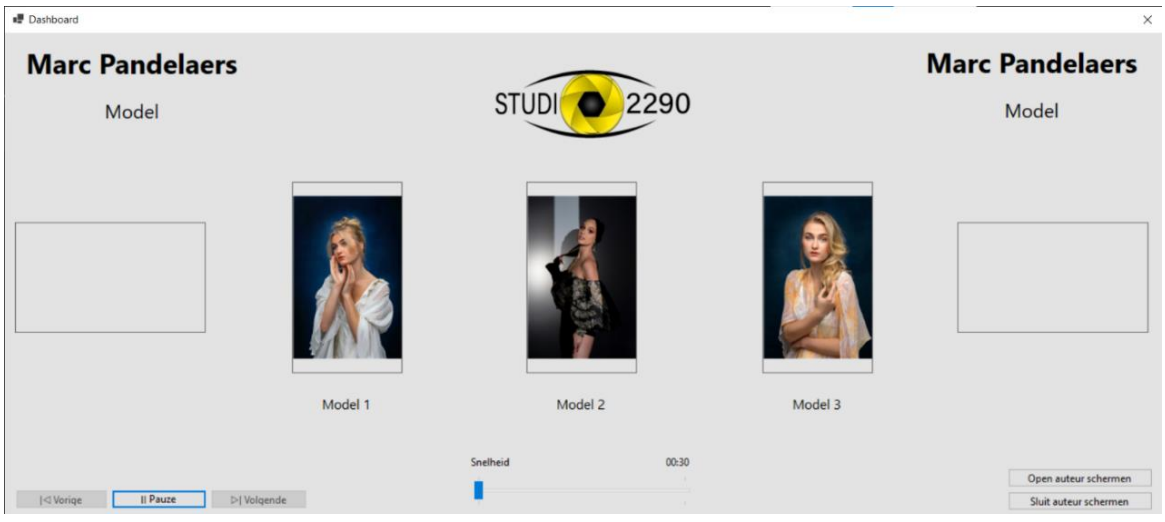
```
Private Async Function SyncUpdates(storage As Sync_Storage, token As CancellationToken) As Task
    Dim DB_Files As List(Of DB_Filepath) = GetDbFiles(storage, syncupdate:=True)

    For Each file As DB_Filepath In DB_Files
        ' Bepaal sync status
        Dim isNew As Boolean = file.ID_Bestand IsNot Nothing AndAlso file.ID_Sync Is Nothing
        Dim isOutdated As Boolean = file.DT_Changed IsNot Nothing AndAlso
                                    (file.DT_Synced Is Nothing OrElse file.DT_Synced < file.DT_Changed)
        Dim needsUpload As Boolean = isNew OrElse isOutdated OrElse
                                    (file.ID_Sync IsNot Nothing AndAlso Not file.IsSynced)

        ' Upload indien nodig
        If needsUpload Then
            Dim response = Await UploadImage(storage, file) ' UPLOAD api call
            If isNew Then
                InsertSyncedStatus(file.ID_Bestand, storage.ID_Storage, response.StatusCode = 200)
            Else
                UpdateSyncedStatus(file.ID_Sync, response.StatusCode = 200)
            End If
        End If
    Next
End Function
```

Figuur 35 Code synchronisatie update

4.4. Dashboard

Wanneer de gebruiker op de hoofdpagina op de knop “Open dashboard” klikt, wordt het onderstaande form geopend. In dit scherm wordt de eerste reeks weergegeven, zoals het systeem is opgesteld. Dit geeft de gebruiker een duidelijk beeld welke informatie op welke schermen wordt getoond. Bovenaan in de hoeken staan de naam van de auteur en de titel van de reeks. In het midden worden de foto’s getoond zoals ze op de schermen verschijnen, inclusief hun correcte positie en oriëntatie.

Links onderaan bevinden zich knoppen om de lus van de reeksen te starten of te pauzeren. Daarnaast kan met deze knoppen naar de volgende of vorige reeks worden genavigeerd, maar dit is enkel mogelijk wanneer de lus gepauzeerd is.
Centraal onderaan staat een trackbar waarmee de gebruiker de tijdsduur kan instellen voor de weergave van een reeks.
Rechts onderaan zijn twee knoppen aanwezig om de auteur-infoschermen te openen of te sluiten. Deze infoschermen worden weergegeven op de niet-primaire schermen die op de pc zijn aangesloten.



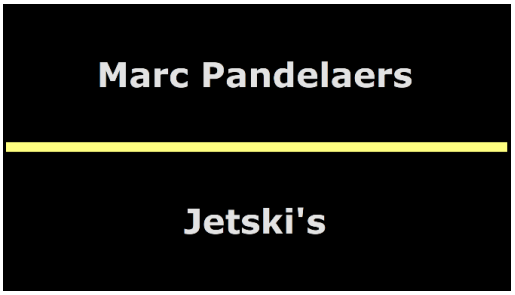
Figuur 36 Form dashboard

Wanneer de gebruiker op ‘Start’ drukt, wordt een timer gestart met de ingestelde tijdsduur. Zodra deze timer is afgelopen, wordt automatisch naar de volgende reeks in de lijst overgeschakeld.
Eerst worden alle schermen verborgen. Vervolgens wordt de nieuwe data naar alle schermen verstuurd. Daarna wordt ervoor gezorgd dat de schermen zich in de juiste positie en oriëntatie bevinden. Tot slot worden de schermen opnieuw getoond, nu met de nieuwe foto en bijhorende informatie.

De auteur-infoschermen tonen bovenaan de naam van de auteur en onderaan de titel van de reeks die op dat moment wordt weergegeven. Dit form wordt in fullscreen geopend, waarbij de lettergrootte automatisch wordt geschaald op basis van de resolutie van het scherm.

```
Public Sub ShowOnScreen(target As Screen)
    Dim fs As Single = (target.WorkingArea.Width / Me.Width) * 48
    Dim ff As FontFamily = Lbl_Auteur.Font.FontFamily
    Lbl_Auteur.Font = New Font(ff, fs, FontStyle.Bold)
    Lbl_Reeks.Font = New Font(ff, fs, FontStyle.Bold)
    Me.Bounds = target.Bounds
    Me.Show()
End Sub
```

Figuur 37 Code tekstgrootte schaling



Figuur 38 Form auteurinfo

4.5. WebSocket-communicatie

Voor de realtime communicatie tussen de desktopapplicatie en de Raspberry Pi werd er gekozen om een WebSocket-verbinding op te zetten met behulp van de FastAPI-library. Deze WebSocket maakt het mogelijk om onmiddellijk commando’s en statusupdates uit te wisselen zonder telkens nieuwe HTTP-verzoeken te moeten uitvoeren.
Naast de realtime communicatie serveert dezelfde FastAPI-applicatie ook twee HTML-pagina’s: één voor de tv-weergave en één voor het infoscherm.

```
app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/display_main")
async def get_main():
    return HTMLResponse(open("static/hoofdscherm/index.html").read())

@app.get("/display_info")
async def get_info():
    return HTMLResponse(open("static/infoscherm/index.html").read())
```

Figuur 39 Python code serveren webpagina's voor schermen

Wanneer een client verbinding maakt met de WebSocket, wordt deze toegevoegd aan een lijst met actieve clients. Elke client stuurt daarbij een parameter mee met de gebruikersnaam. Enkel de desktopapplicatie geeft daarnaast ook het basispad door waarin de foto’s zijn opgeslagen. Dit pad wordt gebruikt om de beelden correct te laden op de schermen. De clientlijst wordt later gebruikt om gerichte communicatie te sturen naar de juiste verbonden schermen.

Na het verbinden wordt een while-lus gestart die continu wacht op inkomende berichten van deze client. Op basis van het ‘command’ in dit bericht wordt de overeenkomstige actie uitgevoerd.

Wanneer de verbinding met een client wegvalt, wordt deze automatisch uit de lijst met actieve clients verwijderd.
Onderstaand zie je de code van deze realisatie.

```python
@app.websocket("/ws")
async def websocket_endpoint(ws: WebSocket):
    await ws.accept()
    user = ws.query_params.get("user")
    clients[ws] = user
    if user == "desktop_app":
        dr = ws.query_params.get("image_path")
        app.mount("/images", StaticFiles(directory=dr), name="images")
    try:
        while True:
            res = await ws.receive_json()
            print("Received:", res)
            msg = WebSocketMessage.model_validate(res)

            match msg.command:
                case "ping":   # --- PING / PONG ---
                    await ws.send_json({"command": "pong"})
                case "new_foto":  # {command: new_foto, data: Obj}
                    message = {"action": "new", "data": msg.data.model_dump()}
                    await send_screens(message)
                    await ws.send_json({"response": "loaded"})
                ...
                case "move":  # {command: move, target: 500}
                    res = send_move_command(direction=msg.direction)
    except WebSocketDisconnect:
        print(f"User {user} disconnected")
    finally:
        clients.pop(ws, None)
```

Figuur 40 Python code WebSocket

Voor de client in de desktopapplicatie werd een klasse ontwikkeld die de verbinding met elke WebSocket beheert. Zodra een instantie van deze klasse wordt aangemaakt, probeert deze automatisch verbinding te maken met de WebSocket. Wanneer de verbinding niet tot stand komt of deze wegvalt, wordt er om de tien seconden een nieuwe poging ondernomen.

Via deze klasse kunnen vervolgens de juiste commando's naar elke WebSocket worden gestuurd om de schermen aan te sturen. Hieronder is een voorbeeld te zien van hoe een nieuwe foto naar de schermen wordt doorgestuurd.

```vb
Private Async Function UpdatePiImages() As Task(Of JsonDocument())
    Dim tasks As New List(Of Task(Of JsonDocument))
    For Each ws In Sockets
        Dim rf As Cls_DragerFoto = ReeksFotos.Find(Function(x) x.Kolom = ws.Kolom)
        If rf Is Nothing Then Continue For
        Dim json As String = rf.GetJsonString(ws.ID_Storage)
        tasks.Add(ws.SendAndReceiveAsync(json))
    Next
    Return Await Task.WhenAll(tasks)
End Function
```

Figuur 41 VB.Net code WebSocket message versturen

Na het versturen van een bericht wacht de client op een antwoord van de WebSocket. Pas wanneer dit antwoord ontvangen is, wordt verdergegaan met het uitvoeren van het volgende commando.

Voor het aansturen van de schermen maken deze eveneens verbinding met de WebSocket en wachten ze op inkomende commando's. Deze commando's kunnen onder andere aangeven dat er een nieuwe foto met bijhorende gegevens moet worden weergegeven, of dat het scherm moet getoond of verborgen worden.

Om de overgang tussen tonen en verbergen vloeiend te laten lopen, wordt er gebruik gemaakt van een eenvoudige CSS-transition die de opacity aanstuurt. Hierdoor ontstaat er een zachte fade-in en fade-out, wat de visuele ervaring aanzienlijk verbetert.

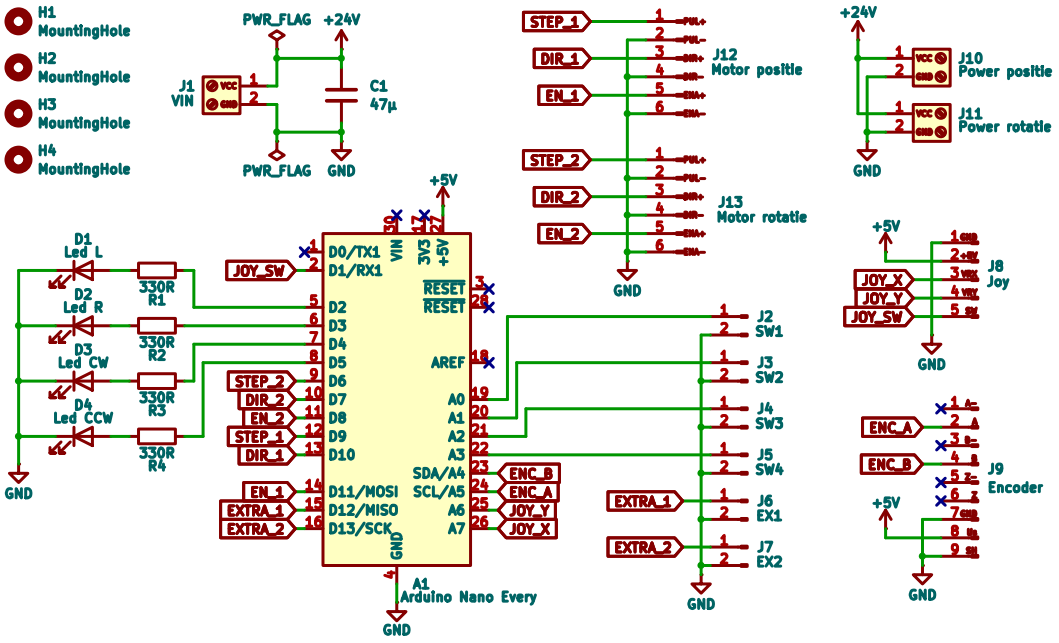
```
ws.onmessage = (event) => {
    const msg = JSON.parse(event.data);

    if (msg.action === "new") {
        if (id_foto != msg.data.ID_Foto) {
            id_foto = msg.data.ID_Foto;
            hasFoto = msg.data.HasFoto;
            if (hasFoto) {
                screen.style.display = ""
                img.src = `/images/${msg.data.Path_4K}`;
                titel.innerText = msg.data.Titel.toUpperCase();
                auteur.innerText = msg.data.Auteur.toUpperCase();
                info.innerText = msg.data.Info;
            } else {
                screen.style.display = "none"
                img.src = '';
                titel.innerText = '';
                auteur.innerText = '';
                info.innerText = '';
            }
        }
    }

    if (msg.action === "show" && hasFoto) {
        requestAnimationFrame(() => (screen.style.opacity = "1"));
    }

    if (msg.action === "hide") {
        screen.style.opacity = "0";
    }
};
```

Figuur 42 Javascript code WebSocket inkomende commando's

4.6. Aansturing motoren

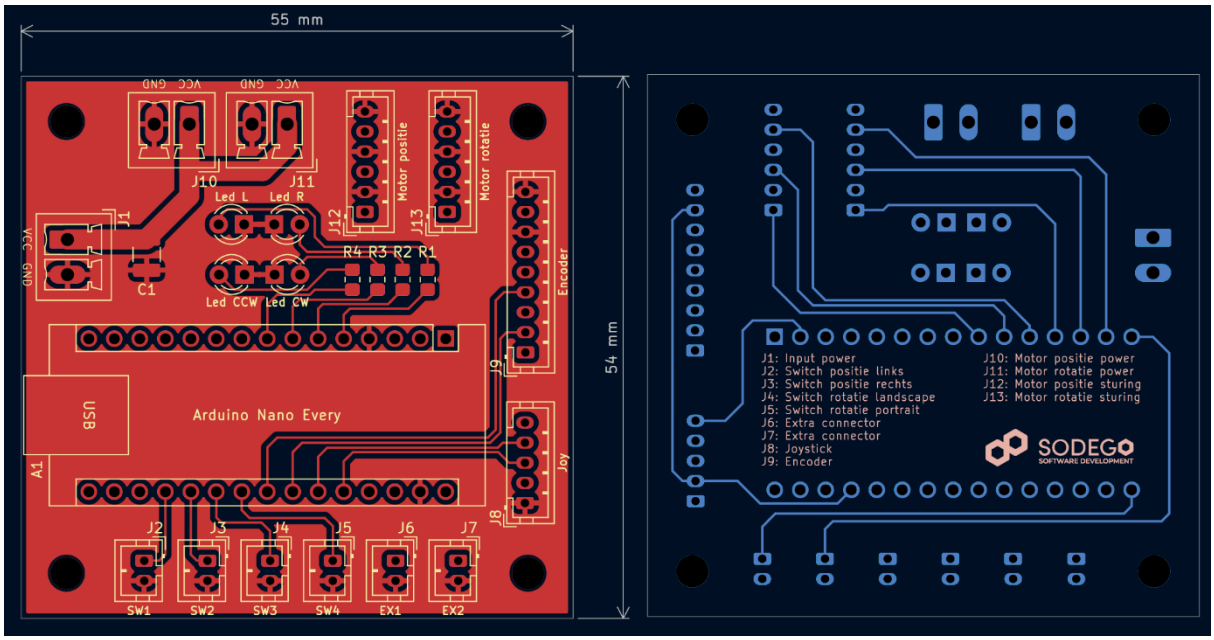
4.6.1. Printplaat

Na het testen van de motorsturing met een breadboardopstelling werd er een schema opgesteld voor het ontwerpen van een PCB. Dit is zowel professioneler als veiliger dan het gebruik van een breadboardopstelling.
Aangezien de beschikbare ruimte in de opstelling beperkt was, was het belangrijk om de PCB zo compact mogelijk te houden. Voor de meeste aansluitingen werd er gekozen voor JST-PH-connectoren, terwijl voor de voedingsaansluiting van de motoren Phoenix-connectoren worden gebruikt.
Daarnaast zijn er enkele LED-indicatoren voorzien die de aansturing van de motoren visueel weergeven. Dit is bijzonder handig om te controleren of de motorsturing correct werkt, bijvoorbeeld wanneer een motor onverwacht niet draait.

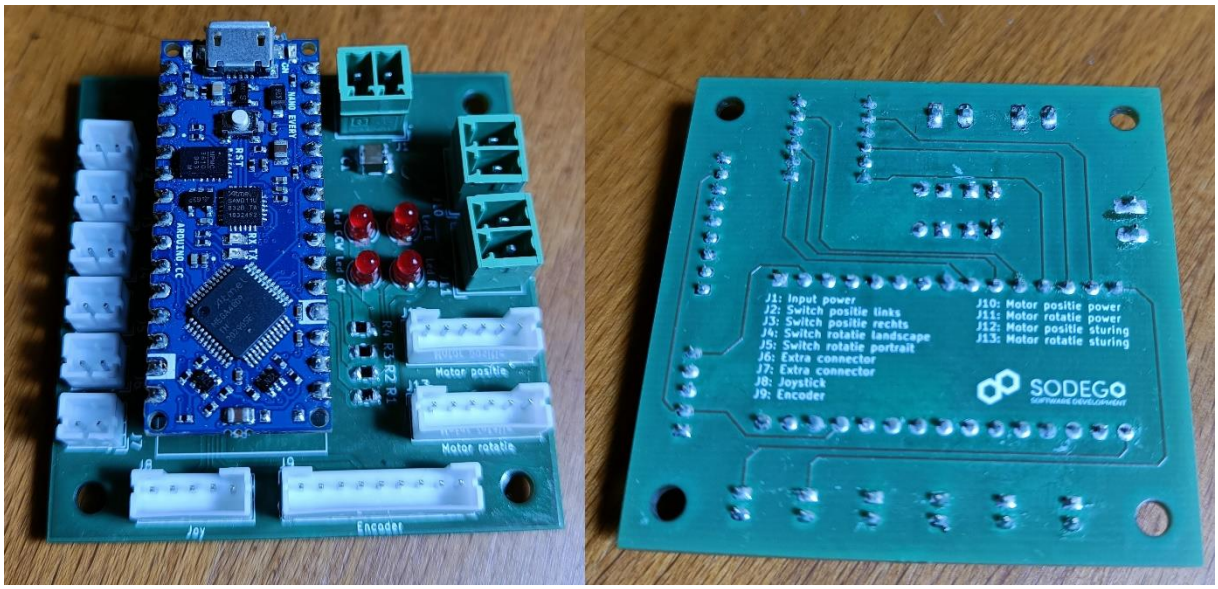


Figuur 43 Schema motoraansturing

Naast de connectoren voor de motoraansturing is er ook een aparte aansluiting voorzien voor de encoder, die gebruikt wordt voor de positiebepaling van de motor. Daarnaast is er een connector aanwezig voor het aansluiten van een joystick, waarmee de motoren indien nodig handmatig kunnen worden aangestuurd.
Tot slot zijn er zes tweepins-aansluitingen voorzien. Vier daarvan worden gebruikt voor de switchen die in de opstelling aanwezig zijn om de limieten van zowel de positie als de rotatie te detecteren. De twee extra aansluitingen zijn aanwezig op de PCB, maar hebben nog geen functionaliteit toegewezen.



Figuur 44 PCB trace lay-out



Figuur 45 Gesoldeerde PCB

4.6.2. Aansturing motoren

Wanneer de motor moet worden aangestuurd, wordt het commando vanuit de WebSocket via seriële communicatie naar de Arduino gestuurd. Dit commando bevat zowel de eindpositie die moet worden bereikt als de draairichting van de motor.
Na het versturen van dit commando wacht deze functie op een antwoord van de Arduino, waarin wordt aangegeven of de gewenste positie al dan niet is bereikt. Deze status wordt vervolgens teruggestuurd naar de desktopapplicatie.

```python
def send_move_command(target, direction="LEFT", expected_output="REACHED"):
    try:
        # Open serial connection
        with Serial(PORT, BAUDRATE, timeout=1) as ser:
            time.sleep(2)  # give Arduino time to reset

            # Flush any existing input/output
            ser.flushInput()
            ser.flushOutput()

            # Send command (encode as bytes)
            ser.write(f'MOVE {direction} {target}\r\n'.encode('utf-8'))
            ser.flush()

            while True:
                if ser.in_waiting > 0:
                    line = ser.readline().decode('utf-8').strip()
                    if line:
                        if expected_output in line:
                            return {"status": "SUCCESS", "message": line}
                        elif "ERROR" in line:
                            return {"status": "ERROR", "message": line}
                time.sleep(0.1)  # Avoid busy waiting

    except SerialException as e:
        print(f"Serial error: {e}")
        return None
```

Figuur 46 Python code seriële communicatie

Wanneer de Arduino een commando ontvangt, wordt de doelpositie voor de motor ingesteld. Voor de zijwaartse beweging wordt de eindpositie bepaald aan de hand van de encoder, terwijl voor de rotatie de eindschakelaars worden gebruikt om de doelpositie te definiëren.

Daarnaast is voorzien dat de motoren vloeiend versnellen en vertragen wanneer ze de eindpositie naderen. Dit voorkomt abrupte bewegingen en zorgt voor een nauwkeurige en stabiele positionering.

```arduino
SerialCommands serial_commands_(&Serial, serial_command_buffer,
                                sizeof(serial_command_buffer), "\r\n", " ");
void cmd_move(SerialCommands* sender);
SerialCommand cmd_move_("MOVE", cmd_move);

void cmd_move(SerialCommands* sender) {
    char* str_target = sender->Next();
    if (str_target == NULL) {
        sender->GetSerial()->println("ERROR no target");
        return;
    }

    long targetCount = atol(str_target);
    int result = moveToPosition(targetCount, 30);
    if (result) {
        sender->GetSerial()->print("REACHED");
        sender->GetSerial()->println(targetCount);
    } else {
        sender->GetSerial()->println("ERROR position not reached");
    }
}
```

Figuur 47 Arduino code command ontvangen

5. Besluit

Het project The Moving Gallery, waar ik tijdens mijn stage aan heb gewerkt, had als doel een geïntegreerde oplossing te ontwikkelen voor het beheren en aansturen van een tentoonstellingsopstelling met meerdere bewegende schermen. Tijdens de stage groeide de opdracht uit tot een volledig systeem waarin software, hardware en databeheer nauw samenwerken.

De applicatie bevat verschillende beheerschermen die op een intuïtieve manier toelaten om alle gegevens te beheren. Daarnaast is er een synchronisatie-architectuur ontwikkeld die ervoor zorgt dat alle fotobestanden op elk apparaat beschikbaar zijn. Via een WebSocket-verbinding wordt de realtime-communicatie mogelijk gemaakt, zodat alle schermen veilig en nauwkeurig kunnen bewegen naar hun ingestelde positie en rotatie en de juiste informatie bevatten om weer te geven.

Ik vond het een bijzonder boeiende opdracht om aan te werken, vooral door de variatie in gebruikte programmeertalen en de combinatie van software- en hardwarecomponenten. Hoewel het systeem nog niet volledig afgewerkt is – omdat de fysieke opstelling nog in opbouw is en daardoor niet volledig getest kon worden – zijn alle noodzakelijke onderdelen aanwezig om een werkend geheel te vormen.

Tegen het einde van mijn stage was er veel schakelen tussen verschillende onderdelen om alles correct met elkaar te laten samenwerken, wat het project uitdagend en leerrijk maakte.

LITERATUURLIJST

Microsoft. (z.d.). *Visual Basic .NET documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/visual-basic/
Microsoft. (z.d.). *Windows Forms documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/desktop/winforms/
FastAPI Project. (z.d.). *FastAPI documentation*. https://fastapi.tiangolo.com/
Microsoft. (z.d.). *SQL Server documentation*. Microsoft Learn. https://learn.microsoft.com/en-us/sql/sql-server/
SICK AG. (z.d.). *Incremental encoder DLS40E-S3AV00360* [Datasheet]. https://www.sick.com/hk/en/catalog/products/motion-control-sensors/incremental-encoders/dls40/dls40…
Wantai Motor. (z.d.). *NEMA 23 stepper motor 23HS22-1504S* [Datasheet].
Wantai Motor. (z.d.). *NEMA 23 stepper motor 23HS16-0884S* [Datasheet].
Arduino. (z.d.). *Arduino Nano Every: Product specifications*. https://docs.arduino.cc/hardware/nano-every/
Toshiba Semiconductor. (z.d.). *TB6600 stepper motor driver* [Datasheet].

GEBRUIK VAN AI

Voor het nalezen op spelling, grammatica en zinsbouw, en voor het verbeteren van de structuur en leesbaarheid van dit document, is gebruikgemaakt van ChatGPT en Microsoft Copilot. Deze hulpmiddelen zijn ingezet als ondersteuning en niet als vervanging van eigen inhoud of onderzoek.

FIGUURLIJST