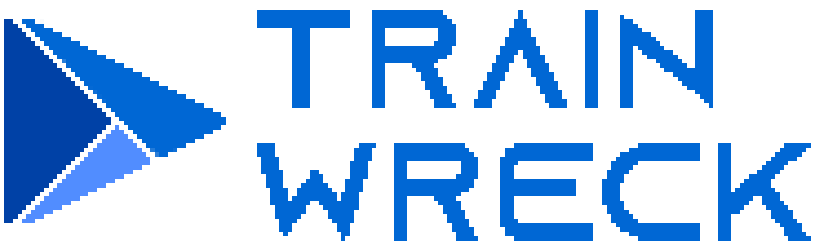


Auteurs:

- Job Lampe
- Mathijs Glazema
- Sven Loeve

Inhoud

Introductie

Product beschrijving:

Trainwreck is een groep die opgericht is om een betere presentatie ervaring te creëren. Dit wilde de groep bereiken per middel van automatisering. Deze automatisering kwam in de vorm van een door magneten aangestuurde trein (aan een rails) en een gerobotiseerde arm die samen een camera over de breedte van het lokaal konden verplaatsen.

Deze camera zou een video feed streamen die de gebruiker via een klein kastje (die ook als inlog systeem gebruikt wordt) naar de laptop gestuurd wordt. Deze video feed kan dan gebruikt worden tijdens presentaties en kan gedeeld worden via alle populaire chat platformen. Denk hierbij aan Teams, Zoom of Discord.

De combinatie van een door magneten aangestuurde trein en robotische arm zorgt voor een kortere setup tijd en haalt de irritaties, die er bij de huidige setup zijn, weg. Met de nieuwe setup is presenteren zo makkelijk als je laptop neerzetten, inloggen op de besturing site, de camera positie kiezen en je bent ‘good to go’.

Team:

De groep Trainwreck bestond oorspronkelijk uit 6 personen. Inmiddels is dit afgeslankt naar 3 gepassioneerde personen. Verdere toelichting is hieronder te vinden:

- Job:
    - rol: PO, developer
    - kennis: Front-end, Back-end, 3D modelleren en TI
    - talen: HTML, CSS, JS, C++

- Mathijs:
    - rol: developer
    - kennis: Front-end, Back-end en TI
    - talen: HTML, CSS, JS, C#, C++

-Sven:
    - rol: Scrum Master, developer
    - kennis: Front-end, Back-end, 3D modelleren en TI
    - talen: HTML, CSS, JS, C++

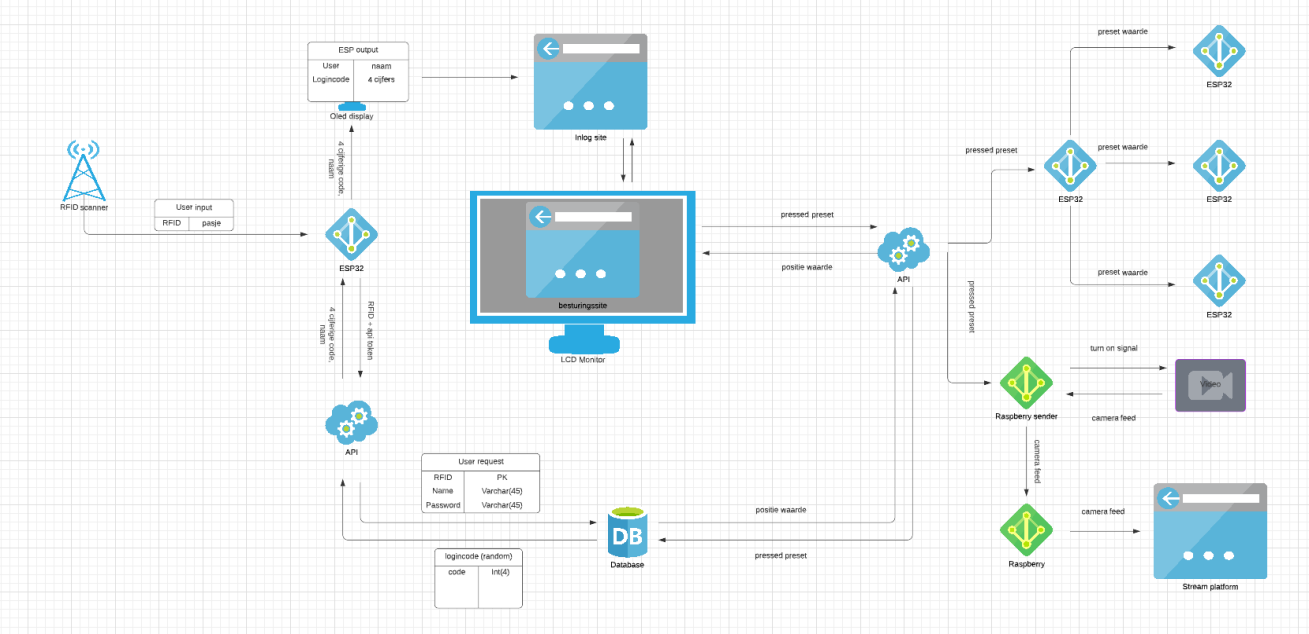
Informatie

Toelichting:

In het volgende hoofdstuk wordt een gedetailleerde uitleg gegeven over alle individuele componenten waar het product uit bestaat. Zo wordt er eerst ingegaan op het globale idee en hoe dit idee uitgevoerd is om vervolgens verder in te gaan op de code achter de site, database en hardware. Op het einde wordt er uitleg gegeven bij de ontwerpen van zowel de magnetische rails als de robot arm. Hierbij wordt er gekeken naar de huls als gedetailleerde doorsneden.

Werking:

Het systeem bestaat uit een frontend, backend en een netwerk van microcontrollers met sensoren. Hoe dit al deze onderdelen met elkaar communiceren is goed te zien op de onderstaande afbeelding.



In de bovenstaande afbeelding is te zien dat alles begint bij de RFID scanner en database. Zo stuurt de database, na het scannen van een geregistreerde pas, een random gegenereerde code op die uit 6 characters bestaat. Met deze code kan de gebruiker inloggen op de besturings site en door het invullen van een locatie voor de camera de microcontrollers aansturen die in de rails en arm zitten.

De gebruikte microcontroller is een ESP-32 en beschikt over een ingebouwd intern netwerk en wifi chip. Na het ontvangen van de gewenste camera positie sturen de ESP's de magneten, sensoren en motoren aan om zo op de gewenste plek te komen. Zodra de camera op de gewenste plek staat zal een raspberry beginnen met het uitzenden van video feed naar een raspberry die in de RFID scanner behuizing verwerkt zit. Deze video feed kan nu op Teams, Zoom of Discord geselecteerd worden als webcam. Na het selecteren van de video feed als webcam is het proces afgerond en kan er gepresenteerd worden.

De rails en arm worden in 3 stappen die op de achtergrond plaatsvinden aangestuurd, maar om überhaupt bij het aansturingssysteem te komen zijn een aantal stappen nodig. Deze stappen worden hieronder verder toegelicht:

1. Zoek de inlogpagina op op de laptop waarmee je wil streamen

2. Sluit het RFID scanner kastje aan op de laptop

3. Als je in het bezit bent van een geregistreerde RFID kaart ga dan verder naar 3.2. Zo niet, zie 3.1

   3.1: Scan de KeyFob en scan de kaart die je wil registreren. Ga hierna naar 3.2

   3.2: Plaats kaart op de RFID scanner en voer de code die op het scherm in op de inlogpagina

4. Ga naar de besturings pagina en selecteer de gewenste plaatsing, hoogte en rotatie voor de camera

5. Druk op bevestigen en wacht totdat de camera op zijn plaats staat

6. Selecteer, op de gewenste streamingdienst, de video feed van de camera.

7. De setup is nu klaar. De presentatie kan beginnen.

Frontend

Beschrijving

Het idee van de frontend van dit project was om een veilige geautomatiseerde omgeving te geven waar je gemakkelijk de robotica kon aansturen, informatie kan terugvinden over de robot en beeld van de camera terug kan kijken.
De frontend is geschreven met het VueJS framework met de volgende plugins en dependencies:

plugins	Versions:
- @vue/cliservice	4.5.17
- @vue/cli-plugin-babel	4.5.17
- @vue/cli-plugin eslint	4.5.17
- @vue/cli-plugin-router	4.5.17

Main-dependencies	
- Axios	0.27.2
- Core-JS	3.22.7
- Vue	3.2.36
- Vue-router	4.0.15

Dev-dependencies	
- @vue/compiler-sfc	3.2.36
- babel-eslint	10.1.0
- eslint	6.8.0
- eslint-plugin-vue	7.20.0

voor alle API calls wordt axios gebruikt en voor de routing en authorisation van de user word Vue-router gebruikt.

Login Pagina

De login pagina van de applicatie is beveiligd met een 6 karakter code die bestaat uit hoofdletters, kleine letters en cijfers. De development pagina heeft een extra knop om een code op te halen om in te loggen.

Standaard is er geen code om in te loggen op de site deze wordt aangemaakt in de backend en database om moment van aanvraag. dit gebeurd op de development frontend of op de RFID scanner waar een code wordt opgegeven op moment van het scannen van een valide pas. hier is in de sectie RFID scanner meer informatie over te vinden.

Authorisation

De site maakt gebruik van een JWT token. met elke Page switch en besturings actie wordt gecontroleerd in de backend of dat de JWT token nog klopt. als de token niet meer klopt volgens de standaard JWT regelgeving wordt de gebruiker uitgelogd en moet hij/zij opnieuw inloggen. als de token klopt wordt de actie die de user wilde ondernemen toegestaan.

Backend

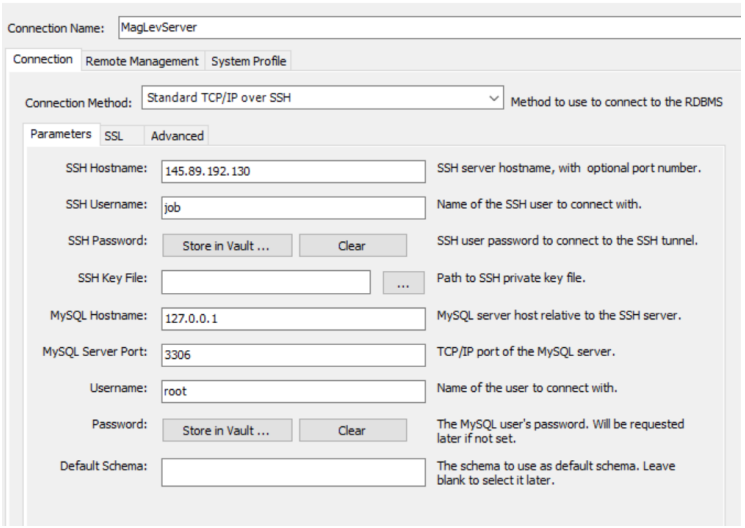
Generale informatie over de backend/server

De backend is geschreven in node.js met als framework express. De gehele applicatie staat op SS

Inloggegevens voor db in code

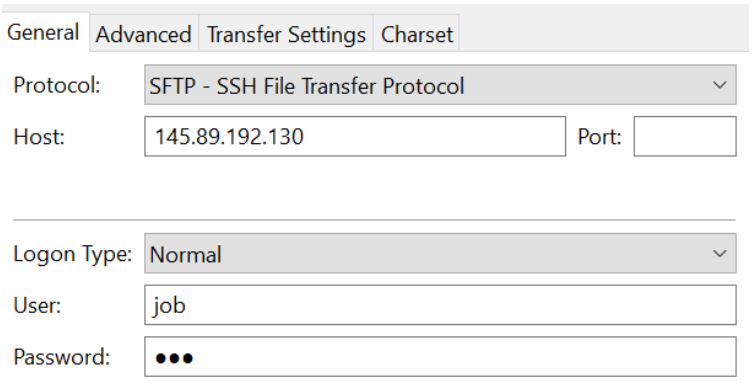
Database:
Host: localhost
User: root
Password: root
Databasenaam: maglevdb

Inlog gegevens voor de db in MySQL Workbench:



Het wachtwoord is root

Gegevens om in de server in the loggen



Het wachtwoord is job

Instructies om de applicatie op te starten/ veranderen

Als je de applicatie wilt starten moet je eerst verbinding maken met de server via ssh dit doe je
door eerst je cmd te open als je cmd hebt geopend dat tik je: ssh job@145.89.192.130. In daarna moet je een wachtwoord invullen het wachtwoord is job.

Als je dat in de server zit moet je het command invullen: sh startLocalhost.sh.

Dat start de applicatie op met de site.

Als je de backend wilt aanpassen moet je ingelogd zijn in filezilla als je dan ingelogd bent moet je het mapje MagLev-1 veranderen met de nieuwe backend als je de frontend wilt aanpassen moet je het mapje webroot veranderen het gaat om het mapje webroot die in 'job' staat niet die in MagLev-1 staat

Informatie over autherizatie

De autorizatie is gemaakt met JWT(JSON web token) dat is een token waarmee je endpoints mee kan autoriseren. In onze code maken we een JWT token aan waarmee je later verschillende acites kan uitvoeren zoals het versturen van data om de arm te updaten. Alle endpoints waarbij in de router verifyToken zoals hieronder (foto 1) staat gebruikt een JWT token. De JWT token moet worden meegeven in de header onder de naam authorization en het moet er staan zoals hieronder(foto 2) aangegeven dus eerst bearer en daarna de JWT token. De JWT token word hieronder(foto3) gemaakt er word ook een tijd megegeven zodat de token maar tot half 11 s avonds geldig is de tijd word bepaald door een functie die timeTilTen heet en heeft de tijd tot half 11 in minuten terug.

```
3   router = express()
4   const { verifyToken } = require("../handler/auth");
5   //routes voor location
6   //router.post("/addLocation", location.newLocation)
7   router.put("/updateLocationHeight", verifyToken, location.updateLocationHeight)
8   router.put("/updateLocationCam", verifyToken, location.updateLocationCam)
9   router.get("/getNewLocation/:name", location.getNewLocation)
10
11  module.exports = router;
```

Foto 1

Parameters

Name	Description
authorization *(header)*	bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV

Foto 2

Foto 3

Api documentatie

Voor de api documentatie kan je gaan naar 145.89.192.130/api-docs

Robotica

Introductie

In dit hoofdstuk wordt er besproken welke hardware Trainwreck heeft gemaakt en ontworpen voor het project. Dit kan variëren van plug-and-play bekabeling tussen 2 componenten tot een volledige PCB (Printed Circuit Board) die voor een bepaald doel is ontworpen.

De hardware producten waar we in dit hoofdstuk naar gaan kijken zijn als volgd:

- RFID scanner

- GPIO expansie

- Motoraansturing voor de arm

- Camera feed sender

- Led strip

In dit project zijn zoals hierboven te zien is redelijk wat producten gemaakt die een microcontroller nodig hebben om te functioneren. De gekozen microcontrollers zijn ESP 32's en beschrikken over een wifi chip + extra functies waardoor ESP 32's op een speciaal intern netwerk kunnen communiceren. Dit netwerk heet het **ESPNOW** netwerk en heeft een bereik tot wel 50 meter. Dit netwerk kan op de 3 manieren die hieronder te zien zijn ingezet worden.

De 'One master' naar 'multiple Slaves' communicatie laat een systeem zien waarin er 1 hoofdapparaat is die data verstuurd naar de sub-apparaten.

De 'Multiple master's' naar 'One Slave' communicatie laat een systeem zien waarin er 1 sub-apparaten is die data ontvangt van meerdere hoofdapparaten.

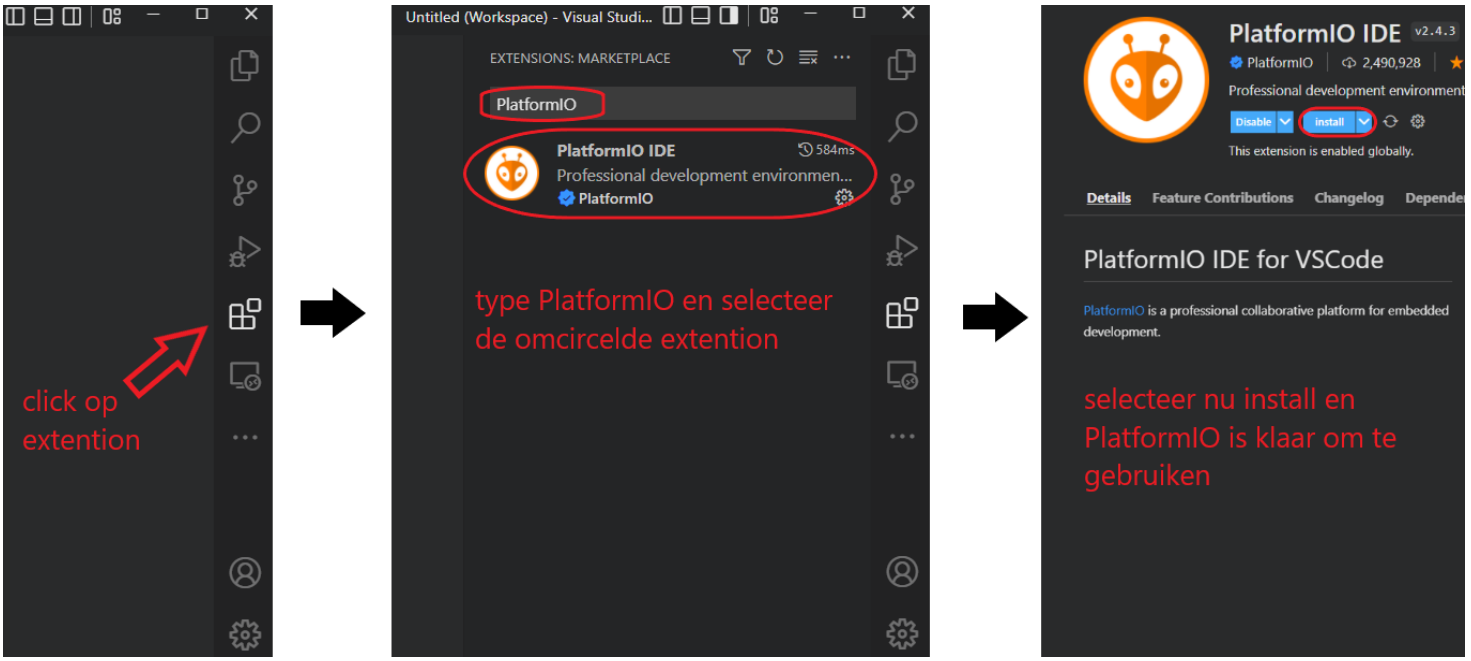
Dit systeem heeft geen 'Master' of 'Slave' apparaten, maar beschikt juist over een tweewegs communicatie tussen alle microcontrollers.

Verdere uitleg over de gebruikte taal en manier van communiceren tussen microcontrollers is te vinden op de eerst volgende pagina.

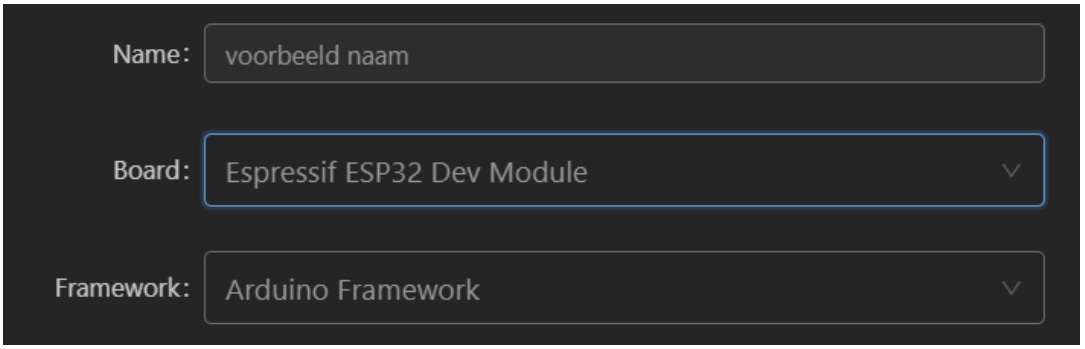
Bij het hoofdstuk ‘Werking’ is toegelicht hoe alle apparaten en microcontrollers met elkaar communiceren. Echter is er nog geen verder uitleg gegevens over de taal die er is gebruikt, welke VScode extension er is toegepast en hoe de microcontrollers op een intern netwerk met elk ander kunnen communiceren.

Laten we beginnen bij de gebruikte taal. Als er gewerkt wordt met niet al te krachtige microcontrollers is het een verstandige keuze om met de taal C of **C++** te werken. Aangezien de meeste programma’s **C++** beter ondersteunen dan C is er gekozen om dit als taal voor het project te gebruiken. De taal is namelijk compact en maakt gebruikt van weinig libraries, wat het dus ideaal maakt voor CPU’s die niet al te uitgebreide talen aankunnen.

Voor het schrijven van de code is gekozen voor een VScode extension genaamd **PlatformIO**. Hier is voor gekozen omdat deze extension het makkelijker en sneller maakt om code te editen of te schrijven. Dit heeft te maken met de hulp software van VScode die shortcuts aanbiedt en suggesties maakt tijdens het coderen over eventuele aanvullingen van teksten of statements. Om **PlatformIO** te kunnen downloaden moeten de volgende stappen genomen worden:



Na het installeren kan de gebruiker, door op de het PlatformIO icoontje te klikken, op het homescherm van de extensie komen. Vanuit hier kan de gebruiker op een bestaand of nieuw project klikken en de gewenste microcontroller met framework aanklikken. Voor het project zijn de onderstaande instellingen gebruikt.



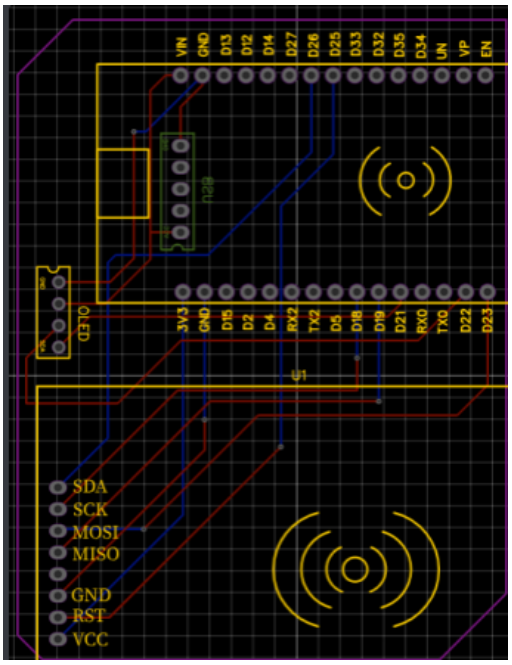
RFID scanner

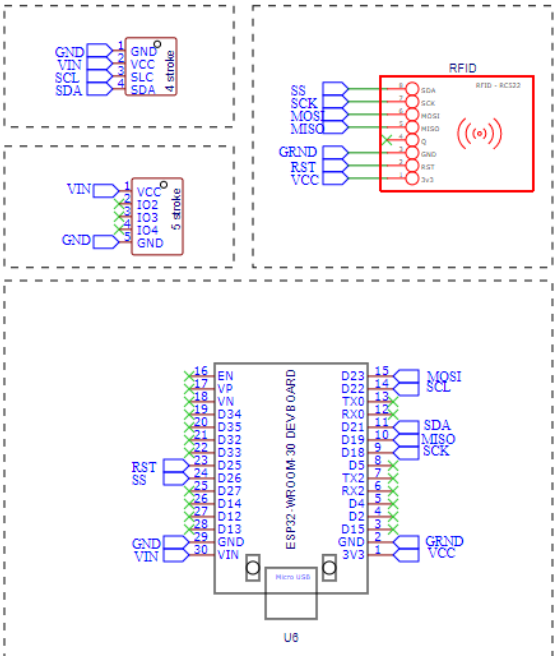
Generale informatie over RFID

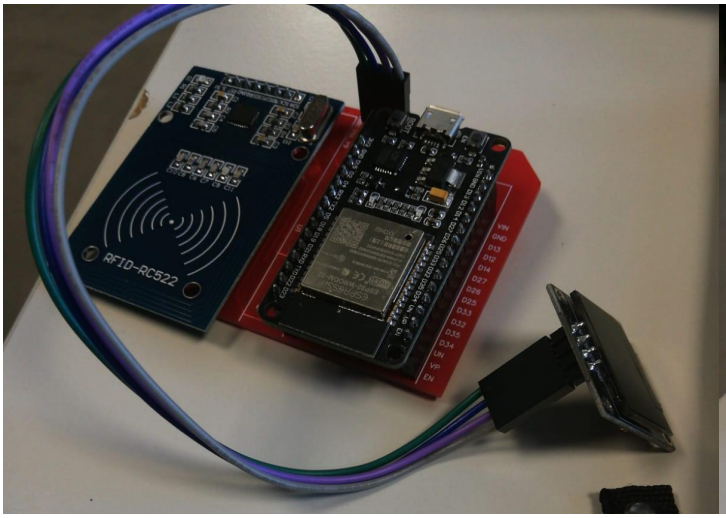
We hebben het inlog systeem gemaakt met een ESP-32 en een MFRC522. Ook hebben we een keyFob. Als je eerst de keyFob scanned en daarna een pasje dan voeg je dat pasje toe aan de database van pasjes waarmee je in het systeem kan. Als je een pasje scanned op die geldig is dan krijg je een code terug waarmee je kan inloggen op de site.

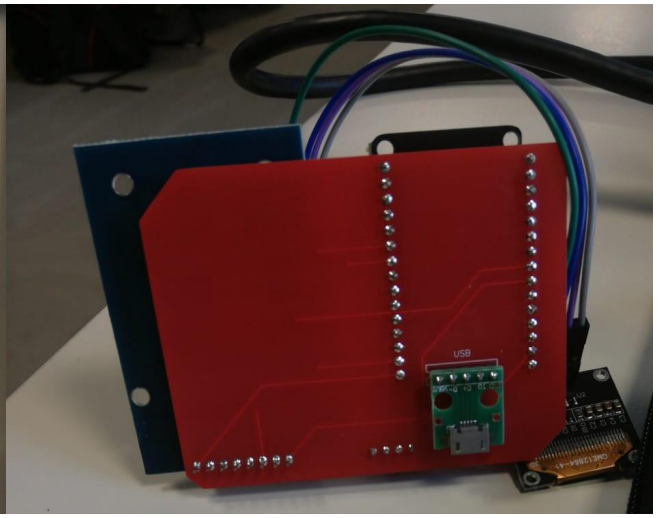
Informatie over de verbindingen

Hier onder staat informatie over hoe de verbindingen tussen de esp, mfrc, oled scherm, mini usb poort is.









Informatie over de code

De code is geschreven in c++ in PlatformIO.
Op line 65 wordt de functie rfid aangeroepen(die begint op line 86) die kijkt of er een pasje is die hij kan scannen, als hij een spasje scanned loopt hij door de code die het pasje geeft en replaced alle spaties met een underscore(foto 1). Daarna checkt hij of de keyfob gescanned is(foto 2). Als de keyfob gescanned is gaat hij in een modus waar je een pasje kan toevoegen(foto 3) als je de keyfob nog een keer scanned gaat hij uit de modus(foto 4). Als je normaal een pasje scanned roept hij een endpoint aan waarmee je een code terug krijgt als je een valide pasje scanned(foto 5).

Foto 1

```
86  void rfidStats(){
87    // put your main code here, to run repeatedly:
88    if ( ! mfrc522.PICC_IsNewCardPresent())
89    {
90      return;
91    }
92    if ( ! mfrc522.PICC_ReadCardSerial())
93    {
94      return;
95    }
96    content = "";
97    for (byte i = 0; i < mfrc522.uid.size; i++)
98    {
99       Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
100      Serial.print(mfrc522.uid.uidByte[i], HEX);
101      content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
102      content.concat(String(mfrc522.uid.uidByte[i], HEX));
103   }
104   for (int i = 0; i < content.length(); i++){
105     if(content[i] == ' '){
106         content[i] = '_';
107       }
108     }
```

Foto 2

```
112   if (content.substring(1) == "65_46_DC_D9" && keyFob == false) {
113     Serial.println("Authorized access");
114     keyFob = true;
115     scanning();
116     delay(2000);
117   }
```

Foto 3

```
118   else if(keyFob && content.substring(1)!= "65_46_DC_D9" ) {
119     Serial.println("add pass");
120     keyFob = false;
121     fobText = true;
122     scanning();
123     delay(500);
124     httpPostrequest(postNewAccount);
125     delay(2000);
126   }
```

Foto 4

```
127   else if (content.substring(1) == "65_46_DC_D9"){
128       keyFob = false;
129       scanning();
130       delay(2000);
131     }
```

Foto 5

```
132   else {
133     Serial.println(content.substring(1));
134     scanning();
135     delay(500);
136     httpPostrequest(postLogin);
137     delay(2000);
138   }
```

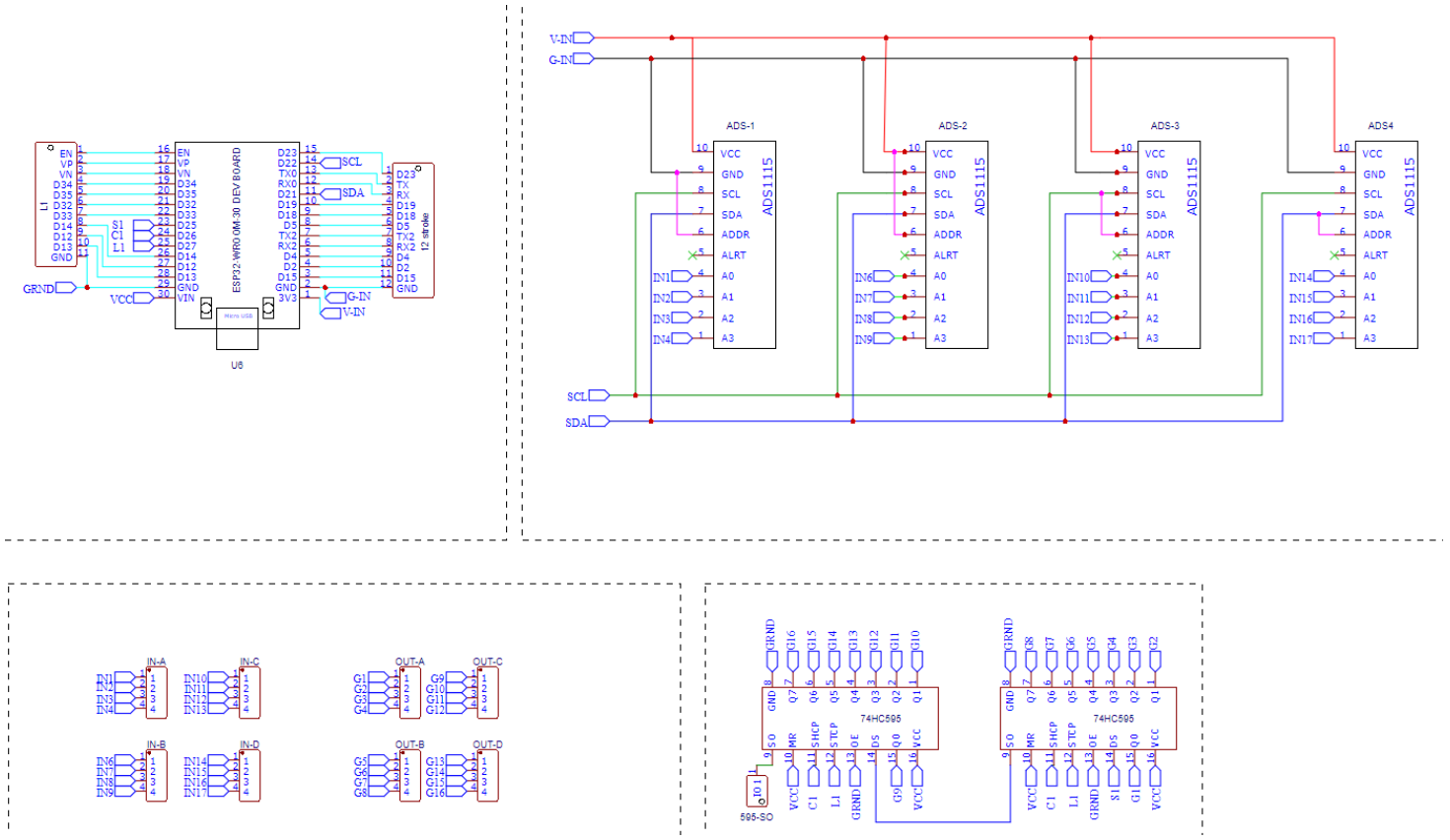
Hieronder staat de code waarmee je iets kan displayen op het schermpje.

```
163   void starterText(String inputText) {
164     display.clear();
165     display.setFont(ArialMT_Plain_10);
166     display.setTextAlignment(TEXT_ALIGN_LEFT);
167     display.drawString(10, 25, inputText);
168     display.display();
169   }
```

GPIO expansie

Microcontrollers hebben input en output poorten. De hoeveelheid poorten verschilt echter per microcontroller. Zo heeft een Arduino Uno 13 in/output poorten die gebruikt kunnen worden om data te ontvangen of juist data op te sturen, maar heeft een arduino nano er maar 10.

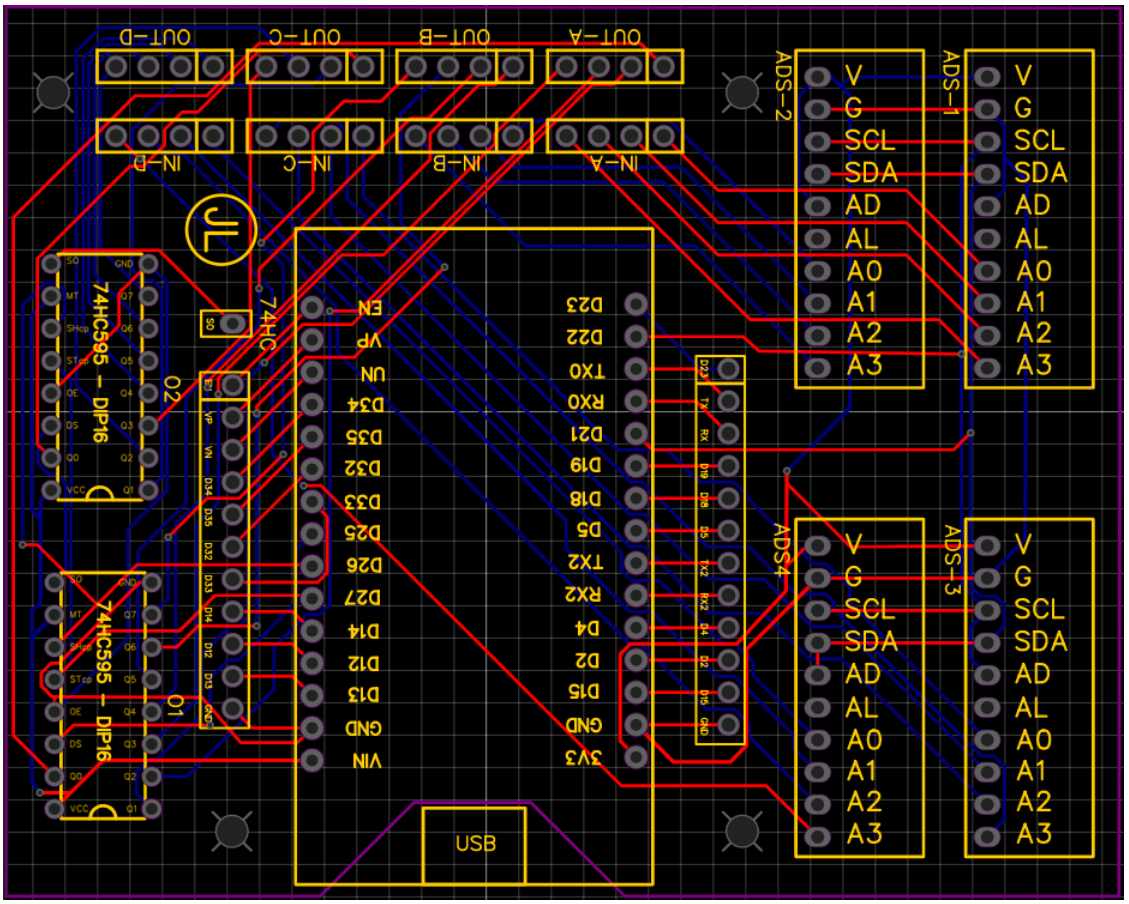
Het verschil in de hoeveelheid poorten op een microcontroller is ook goed merkbaar op de ESP 32. Zo heeft de ESP 32 maar 8 poorten die zowel voor in als output gebruikt kunnen worden en nog 10 andere poorten die specifiek voor input of communicatie bedoelt zijn en dus niet voor als output zender gebruikt kunnen worden. Aangezien dit project veel in/output poorten nodig heeft voor het aansturen van de elektromagneten is er een expansie board gemaakt. Dit expansie board beschikt over 16 extra input en 16 extra output poorten. Op de schematische tekening (van de gemaakte PCB) hieronder is te zien hoe deze 32 extra poorten tot stand zijn gekomen.



Op de bovenstaande afbeelding is te zien dat de extra input poorten uit 4 ads1115 modules bestaan en dat de ‘ADDR’ poort bij iedere module aan een andere leiding gekoppeld zijn. Deze mismatch koppeling van de ‘ADDR’ zorgt ervoor dat alle ADS1115 modules een ander MAC-adres krijgen en hiermee allemaal individueel aangesproken kunnen worden door de microcontroller. Meer hierover bij de code uitleg.

Op de schematische tekening zijn ook 2 74HC595 shift resistors te zien. Deze resistors zorgen voor de extra 16 output poorten, wat voor een totaal van 32 extra poorten zorgt. In tegenstelling tot de ADS1115 modules werken de 74HC595’s niet parallel met elkaar maar in een schakel. Dit wil zeggen dat de resistors aangestuurd kunnen worden met een 16 bit code. Het volgende voorbeeld ‘**1100000000000011**’ zet de eerste 2 en laatste 2 poorten aan.

Om een beter beeld te geven van hoe de componenten aan de microcontroller gekoppeld zijn kan er naar het onderstaande afbeelding gekeken worden. Hierop is te zien waar de daadwerkelijke extra poorten zitten en waar de ADS1115 en 74HC595 gepositioneerd zijn staan.



De overige poorten aan de zijkant van de microcontroller geven de gebruiker de mogelijkheid om de niet gebruikte poorten van de ESP 32 toch te gebruiken om bijvoorbeeld iets te testen.

Bij de code is er geprobeerd overzicht te behouden van alle gebruikte functies en deze functies onder de juiste categorieën te plaatsen. Deze categorieën zijn per middel van comment aangeduid en zorgen ervoor dat 2 verschillende code goed van elkaar gescheiden worden. Zie de voorbeelden hieronder voor meer duidelijkheid.

1.
```
}
// ----------------------------------------------
// pinout setup + startup + write state
void clearRegisters(){
```

2.

3.
```
}
// ----------------------------------------------
// wifi + get request pre-sets
void wifi () {
```

```
}
// ----------------------------------------------
// input PIN setup
void adsReadCycle(){
```

De output poorten kunnen met de onderstaande function aangestuurd worden.

```
135   void writeGrpRelay(int port, int delayTime){
136     registers[port] = HIGH;
137     writeRegisters();
138     delay(delayTime);
139   }
```

Hierbij hoeft de gebruiker simpelweg alleen de gewenste poort in te vullen als parameter en de delay die hij/zij op de poort wil hebben staan. De delay geeft aan hoelang een poort aan staat. Hieronder staat een voorbeeld van hoe het eruit zou zien wanneer een gebruiker poort 6 voor 5 seconden aan zou zetten.

```
80   void loop(){
81     writeGrpRelay(6, 5000);
82   }
```

Het lezen van de data die binnenkomt op de ADS1115 modules kan in 1x gebeuren, maar dit zorgt voor efficiënte code en onnodig veel lijnen. Om het compact te houden wordt er gebruik gemaakt van een ‘for loop’ deze loop stopt pas wanneer de gegevens van alle 16 poorten binnen zijn. Zie de afbeelding hieronder.

```
89   void adsReadCycle(){
90     for(int j=0; j<4; j++){
91       adsCycle[j] = ads[0].readADC_SingleEnded(j);
92       delay(5);
93       adsCycle[j + 4] = ads[1].readADC_SingleEnded(j);
94       delay(5);
95       adsCycle[j + 8] = ads[2].readADC_SingleEnded(j);
96       delay(5);
97       adsCycle[j + 12] = ads[3].readADC_SingleEnded(j);
98       delay(5);
99     }
100  }
```

De bovenstaande ‘for loops’ halen alleen de gegevens op, maar tonen deze nog nergens. Aangezien de ADS1115 modules analoge signalen opvangen en geen digitale moet de ruis eruit gefilterd worden. De onderstaande loop filtert de ruis en zorgt ervoor dat de gebruiker alleen een 0 of een 1 te zien krijgt in de terminal.

```
102  void printValue(){
103    for(int i=0; i<16; i++){
104      if(adsCycle[i] < 17500){
105        adsCycle[i] = 0;
106      }
107      if(i<9){
108        Serial.println("DP " + String(i + 1) + "   : " + String(adsCycle[i]));
109        delay(5);
110      } else {
111        Serial.println("DP " + String(i + 1) + "  : " + String(adsCycle[i]));
112        delay(5);
113      }
114    }
115    Serial.println("--------------------- end input --------------------");
116  }
```

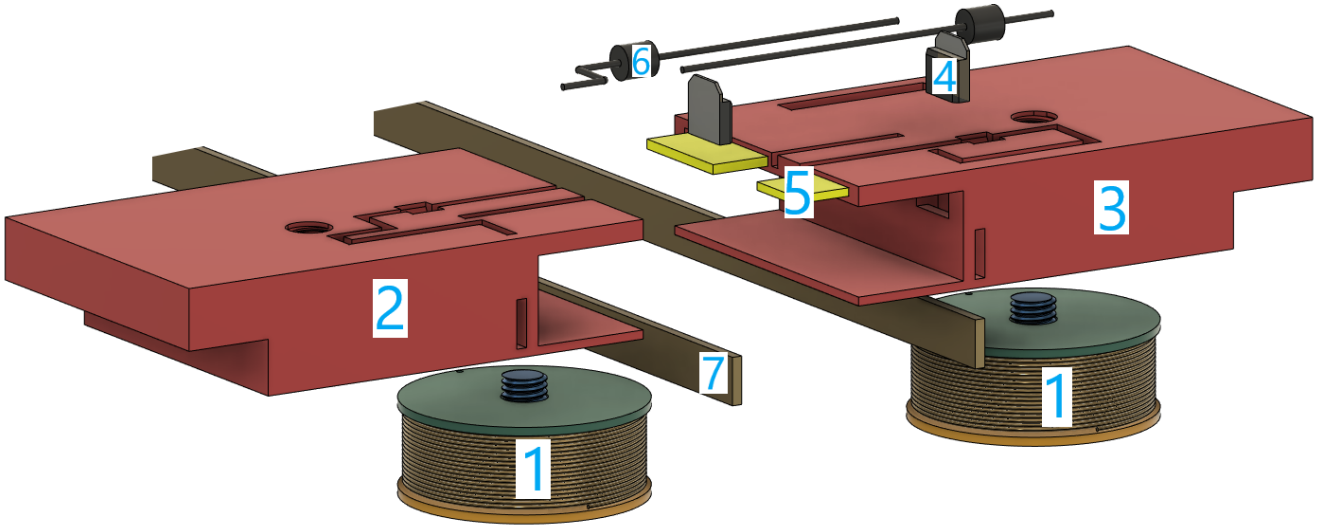
De data van de input pinnen kunnen in combinatie met de output pinnen gebruikt worden om zo bij een bepaald signaal van sensoren een specifieke output pin aan te sturen.

*Voor de hele code ga naar de bijlage en kijk in ‘**INnOutPut control**’.*

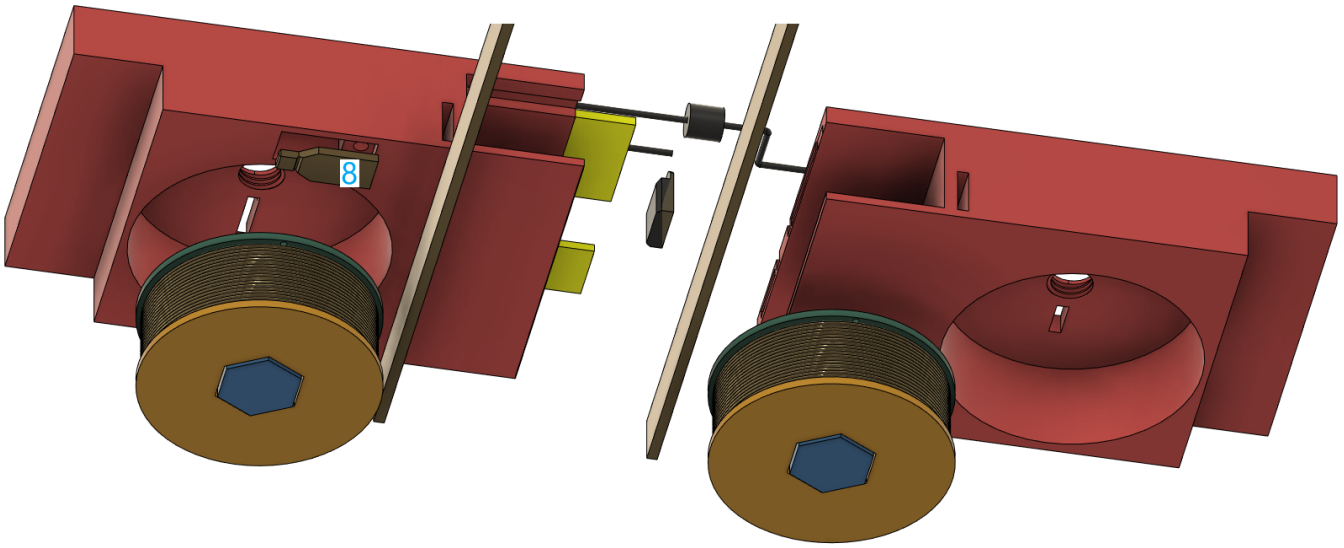
Rails aansturing module

De wagon op de rails wordt niet aangestuurd door een elektromotor, maar juist door een array van elektromagneten (*meer hierover in het volgende subhoofdstuk*). Deze array van elektromagneten bestaat uit modules waarvan iedere module beschikt over 2 elektromagneten. De module array wordt aangestuurd door een ESP 32 die aan de hand een **hall effect** sensoren weet waar de wagon zich bevindt en de magneten met deze informatie aan of uit zet.

Dit aan en uitzetten gebeurt in een split seconde en vraagt dan ook om een stevig switch. In de rails is er een **N-kanaal mosfet** gebruikt als switch. Het voordeel van een N-kanaal mosfet is niet alleen dat de stroomcircuit snel onderbroken kan worden, maar dat mosfets gebruikt kunnen worden voor applicaties waar veel stroom bij komt kijken. Zo zijn er in de rails modules mosfets gebruikt die tot wel 10 ampère aan stroom aankunnen. Hieronder is een uit elkaar gehaalde variant te zien van de module waarin goed te zien is hoe alle componenten met elkaar verbonden zijn.



Bovenaanzicht



Onderaanzicht

Op de vorige pagina waren nummers in de afbeeldingen te zien. Deze nummers geven de componenten in de modules aan. Hieronder staat omschreven wat het component is en welke functie het component heeft.

1. Elektromagneet: trekt de wagon met magneten aan de bovenkant vooruit
2. Linkers module: behuizing van componenten in module
3. Rechter module: behuizing van componenten in module
4. Mosfet: Switch voor stroomtoevoer van magneten
5. Positie pin: zorgt voor goede koppeling van linker en rechter module
6. Diode: zorgt ervoor dat de stroom maar 1 richting op kan
7. Kopere lat: Zorgt voor stroomtoevoer voor de magneten
8. Hall effect sensor: detecteert de magneten op de wagon en stuurt dit door naar ESP32

De esp 32 is in het midden van de module gepositioneerd, maar aangezien de array uit meerdere modules bestaat zal de esp 32 zich in het midden van deze array bevinden en hoort daarom niet bij een individuele module.

Elektromagneten

De elektromagneten in de rails zijn met speciale eisen gemaakt. Zo kan de koperen bedrading in de elektromagneet 10 tot 12 ampère aan en heeft de magneet een gemiddelde trekkracht van 65 newton wat omgerekend 6.5 kilogram aan trekkracht is. Om dit uit te rekenen is de onderstaande formule gebruikt.

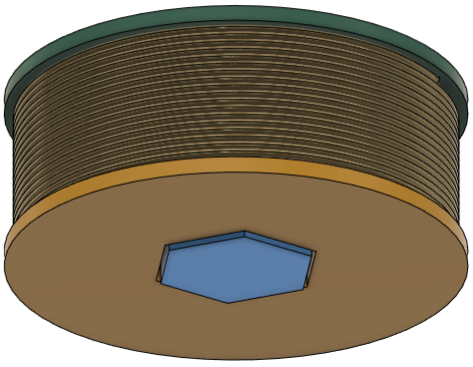
$$F = ((n \cdot i) \cdot \text{magnetische constante}) \,/\, g$$

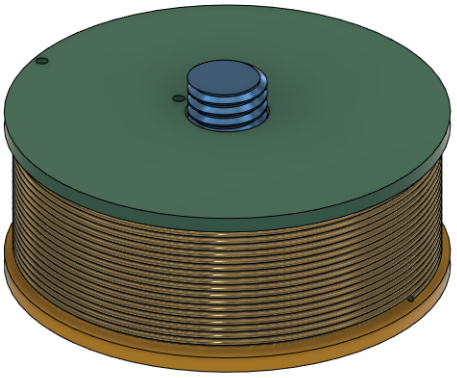
Hierin staat **F** voor **Tesla**, **i** voor stroom, **g** voor de lengte van de opening tussen de magneet en een stuk metaal, **n** voor de aantal omwentelingen in de magneet, en de magnetische constante is 4 x PI x 10-7.

Bij het invoeren van de waarde voor de magneten die gebruikt worden voor de rails ontstaat de onderstaande formule. Met als uitkomst: 0.251327 die omgerekend **2.5 kg** trekkracht is.

$$F = ((200 \cdot 10) \cdot 4 \cdot PI \cdot 10\text{-}7) \,/\, 0.01$$

Helemaal omgewenteld zal de elektromagneet uit komen te zien zoals hieronder gepresenteerd.





Motor Aansturing

De motoraansturing is een vrij simpel stuk code. tijdens het lopen van de code checked de microcontroller constant op nieuwe data waarin wordt gezegd waar hij heen moet bewegen. wanneer er nieuwe data wordt doorgegeven draait de motor naar het nieuwe punt toe vanaf zijn eigen 0 punt. op moment wordt deze nog fake snel aangegeven. om het in een makkelijke volgorde te zetten hoe het systeem checked waar het heen moet:

Huidige systeem:

- De motor krijgt een fake 0 punt waar hij vandaan moet bewegen.
- De ESP checked waar de motor heen moet bewegen in graden.
- De ESP kijkt naar de waarde of hij hoger of lager is dan waar hij heen wilt bewegen:
- Als de waarde hoger is beweegt de motor naar beneden.
- Als de waarde lager is beweegt de motor omhoog.

```
56
57    int labelStart = response.indexOf("height\":");
58    // find the first { after "content":
59    int contentStart = response.indexOf(":", labelStart);
60    // find the following } and get what's between the braces:
61    int contentEnd = response.indexOf(",", labelStart);
62    String content = response.substring(contentStart + 1, contentEnd);
63
64    int contentInt = content.toInt();
65  //    Serial.println(contentInt);
66
67    if(contentInt == oldValue){
68        berekenGradenHoogte(contentInt);
69        delay(1000);
70        return;
71    } else {
72        float aantalGradenDraaien = berekenGradenHoogte(contentInt);
73        Serial.println("aantal stappen:");
74        int goedDraaien = aantalGradenDraaien - oldValue;
75        Serial.print(goedDraaien);
76        stepper.rotate(goedDraaien * 10);
77        //   stepper.rotate(1.8);
78        oldValue = contentInt;
79      //   Serial.println(oldValue);
80    }
81
82  >   // Tell motor to rotate 360 degrees. That's it.···
91    delay(1000);
```

/\ op moment staat er in lijn 76 een x10 bij dit was voor demonstratie redenen en past de motor aan om 10x meer te draaien.

Volgende systeem:

Vanwege de restricties die we hadden en hebben en materiaal gezien en natuurkundig gezien moeten de motoren met reductie versnellingen draaien waardoor ze meer kracht krijgen maar minder snel draaien. dit betekend dat de motoren meerdere rotaties moeten doen voor een paar graden draaien. hiervoor zijn de stepper motoren niet de beste optie maar dit is wat we hadden. om het toch deels te laten werken met deze motoren hebben we ook de volgende code. in dit systeem controleren de motoren constant waar ze zitten op basis van een gradenmeter in de arm dit kan simpel gedaan worden met een potmeter of een meer gespecialiseerde optie. hierbij krijgt hij te weten welke graden hij moet hebben en op basis van die informatie draait hij in de plus of min om dit bij te stellen. met de eventuele toevoeging van een PID systeem zou dit perfect zijn om de arm op een gecontroleerde en soepele manier te laten bewegen maar hier zou nog verder onderzoek naar gedaan moeten worden.

```
#define DIR 27
#define STEP 26
#define MS0 32
#define MS1 33
#define MS2 25

// Instructions for the controller settings can be changed above.
DRV8825 stepper(MOTOR_STEPS, DIR, STEP, MS0, MS1, MS2);

// Setup instructions.
void setup()
{
    // ??
    Serial.begin(115200);

    // Target RPM and Step size (RPM, Step).
    stepper.begin(100, 32);
}

// Main code run on launch.
void loop()
{
    if (CurrentRotation != TargetRotation)
    {
        Serial.print(" Rotation= ");
        Serial.print(CurrentRotation);
        if (CurrentRotation > TargetRotation)
        {
            stepper.rotate(-1);
        }
        else if (CurrentRotation < TargetRotation)
        {
            stepper.rotate(1);
        }
    };
}
```

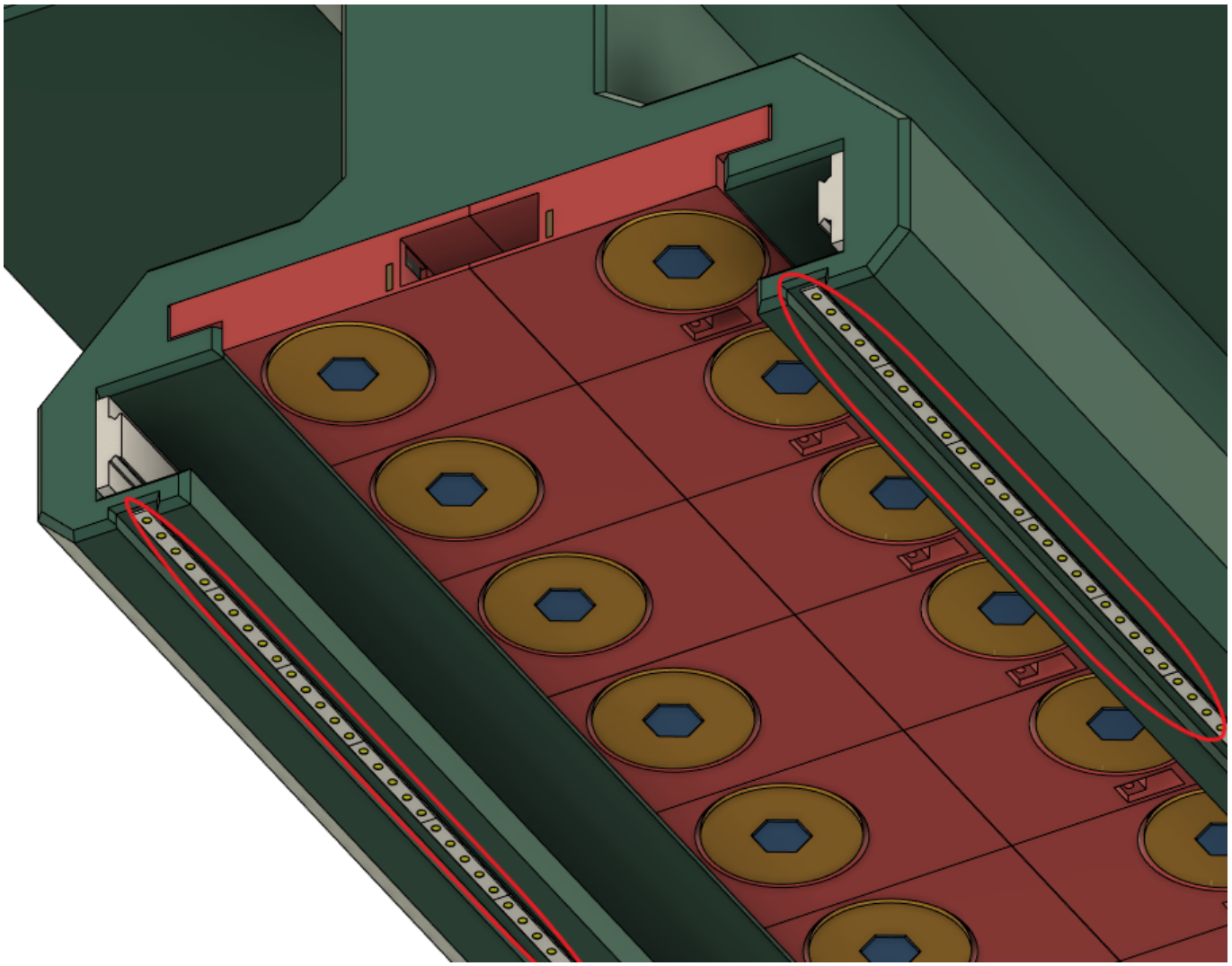
/\ bovenstaande code is nog in de prototype fase maar is wel werkend met de toevoeging van de potmeter en API call voor de gevraagde rotatie in graden.

Camera feed sender

De camera feed was een groot obstakel in dit project. voor deze arm wilden wij een wireless video systeem gebruiken vanwege het draaien van de arm. We hebben wel een concept dat zou kunnen werken maar hadden er de hardware nog niet echt voor. Een klein systeem dat we nu hadden gebouwd is dat we op een raspberry pi een signaal streamen naar een rtsp server die bereikbaar is in vlc media player. Een andere oplossing zodat het signaal meer accessible was in de site was om het rtsp signaal om te zetten in een rtmp signaal aangezien dat signaal nieuwer is en en meer dingen geïmplementeerd.

Led strip

Na het selecteren van een pre-set geven led-strips onder de rails aan waar de wagon gaat komen te staan. Dit zorgt ervoor dat mensen niet in de weg van de wagon gaan lopen zodra die begint te bewegen aangezien dit vroegtijdig door de led-strips aangegeven wordt. Op de afbeelding hieronder is te zien waar op de rails de led-strips bevestigd zitten.

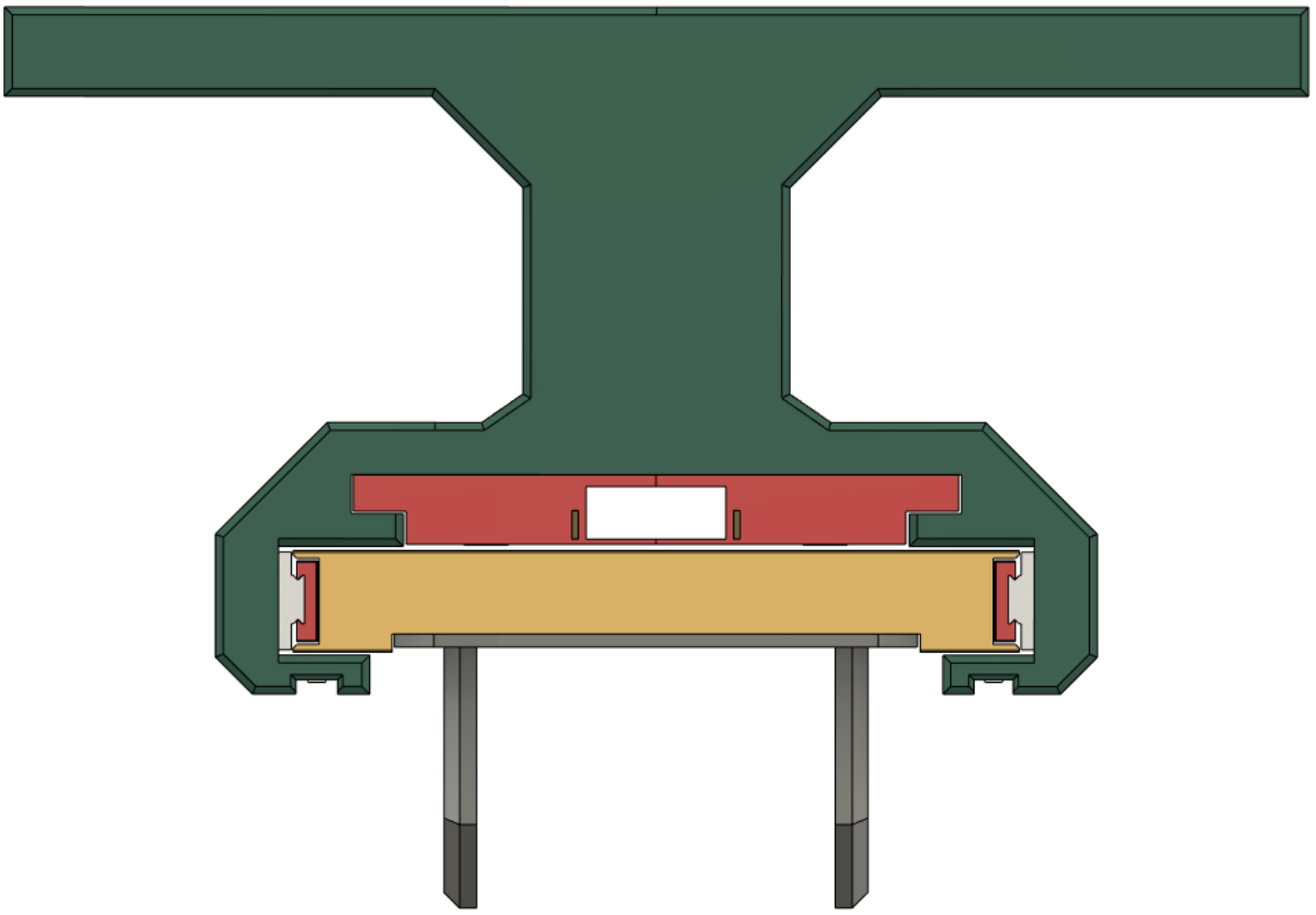


De preset voor de wagon krijgt een esp 32, die aan de led-strips gekoppeld is, binnen vanuit de site en voert 1 van de pre-set led settings uit. Iedere led-strip heeft een maximale hoeveelheid van 144 leds. Deze leds kunnen individueel aangesproken worden of in paren om in een later stadium bepaalde patronen of presets te testen.

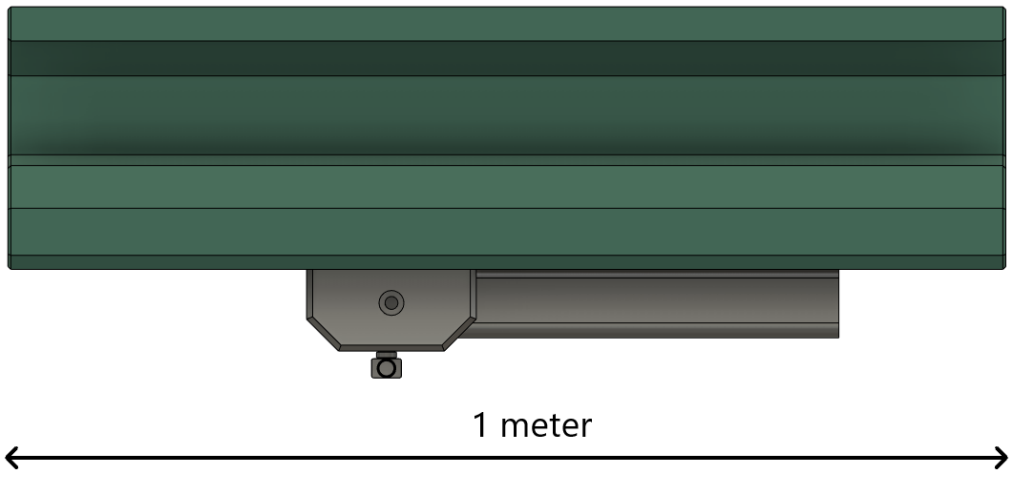
Design

Global design

De rails heeft een omgedraaid ontwerp. Zo hangt de arm/wagon aan de rails in plaats van dat de arm/wagon over de rails rijdt. Dit is gedaan omdat de rails aan het plafond zit en het dus makkelijker was de arm/wagon in de arm te laten rijden dan eroverheen. De arm/wagon zit aan de linker en rechter kan zit met een lineaire rail guide vast aan de rails. Op de onderstaande afbeelding is het vooraanzicht met de bevestigingspunten te zien.



De midden kolom is hol wat extra ruimte over laat voor de stroomtoevoer, de bekabeling en wat extra gewicht bespaard. De rails is zo ontworpen dat de arm net onder gipsplaten van het plafond uitkomt. Dit verklaart ook meteen waarom de rails zover naar beneden hangt. De rails bestaat uit 1 meter modules zoals hieronder te zien is.Dit is gedaan zodat er niet direct 1 specifieke afmeting voor de rails hoeft te zijn, maar dat de klant hier later nog op terug kan komen en kan besluiten de rails te verkorten of te verkleinen.



Motor design