



Project Documentatie IoT

Roaming Ochtend Alarm Clock from Hell
(ROACH)

Emre Arslan
Joseph Assayag

Inhoud

Projectidee	3
Geplande Features vs Actual Features	3
Schatting Requirements vs Actual Requirements	3
WebApp	5
Communicatie	5
Mqtt	7
Code	9
HTML	9
Javascript	10
CSS	14
Arduino	16
Electronica	16
Motor controller + dc motors + batteryPack	16
L298N H-bridge	16
H-bridge	16
Battery Pack	16
Motion sensor + buzzer	17
Buzzer - NPN Transistor	17
Werking transistor	17
MotionSensor	17
Wifi (Esp8266) module	18
Bootmode verbindingen (om code te flashen)	18
Weerstand	18
Alarm turn off button	19
Code	20
Arduino UNO	20
ESP8266	31
Conclusie	35



Projectidee

Dankzij de Roaming Ochtend Alarm Clock from Hell (ROACH) hebben we overslapen herleidt tot een begrip uit een ver verleden. De ROACH doet naast het afgaan van een alarmsignaal ook een wilde rondrit en probeert om niet gepakt te worden dankzij een bewegingsmelder. Het is dus een echt wek-duiveltje.

Ons project betrof het ombouwen van een speelgoedauto en het daarop voorzien van de noodzakelijke features. De bedoeling was om een speels project te hebben dat zinvol was, waar we veel uit zouden leren en uiteraard om altijd op tijd op te staan voor de lessen IoT die om 8u15 beginnen.

Geplande Features vs Actual Features

Geplande Feature	Uitgevoerd?
Uur van opstaan instelbaar vanaf smartphone via internet op webapp	JA
Voorzien van een verschrikkelijk alarmsignaal	JA
Een knop om het alarmsignaal af te zetten	JA
Een motion sensor die bij het benaderen de roaming clock verder laat rijden (om niet gepakt te worden) in een willekeurig pattern	JA

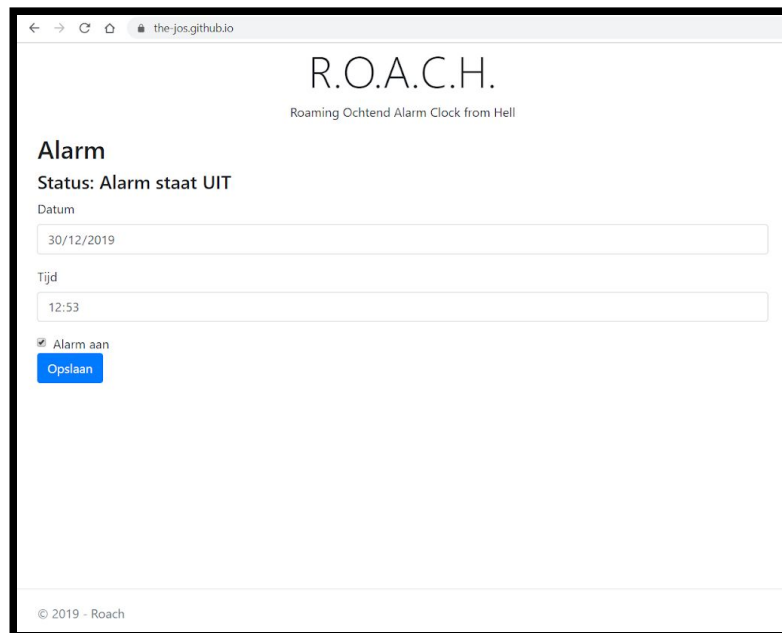
Schatting Requirements vs Actual Requirements

Geplande Requirement	Actual Requirement
De speelgoedauto is voorzien van een eigen voeding.	Dit bleek inderdaad nodig. De Arduino is voorzien van een typische 9V batterij, de H-bridge die de 2 motoren aanstuurt van 4 x 1.5V (AA) batterijen.
De manier waarop de auto wordt bestuurd zullen we moeten reverse engineeren om onze arduino de motors van de auto te laten aansturen.	We hebben de oorspronkelijke RF module verwijderd en onze H-bridge aangesloten op de bestaande bekabeling om zo de 2 motoren aan te sturen.
Om het uur bij te houden is een klok module nodig.	Dit bleek niet nodig. We berekenen de waarde van de countdown op de webapp en laten de Arduino simpelweg elke seconde aftellen vanaf deze countdown.
Een sensor moet voorzien worden om	Dit is bijzonder goed gelukt. De sensor

kwaadwillige ochtendmensen te detecteren	detecteert zeer goed in 360 graden wat voor ons doeleinde perfect is.
Logica is nodig om zo vervelend mogelijk rond te rijden en te stoppen en navenant te reageren als de sensor een ochtendmens detecteert.	We hebben een aantal random routines voorzien om de richting en de duur van het rijden te bepalen. Zo bekomen we succesvol een hoge graad van hulpeloosheid bij het slachtoffer van de Roach.
Er is een (mono-)schakelaar nodig om het alarmsignaal te doen stoppen.	Dit hebben we gedaan met een LED drukknop. De LED knippert als het alarm afgaat. Als je de knop indrukt, stopt het alarm. Je moet de Roach wel eerst vangen.
Een internetplatform moet voorzien worden om het uur van de wekker in te stellen.	In orde.
Er is een vorm van communicatie nodig naar het internet, hoogstwaarschijnlijk wifi, dat een extra module zal vereisen.	Een ESP8266-01 module is hiervoor via seriële verbinding aangesloten aan de Arduino. Er werd een eigen protocol voorzien om deze communicatie te verzekeren.

WebApp

Voor de WebApp wordt gebruikt gemaakt van HTML5 en Javascript. De communicatie met Mqtt gebeurt via Javascript dankzij de Paho Mqtt library (<https://www.eclipse.org/paho/>) en een gratis account bij CloudMqtt (<https://www.cloudmqtt.com/>).



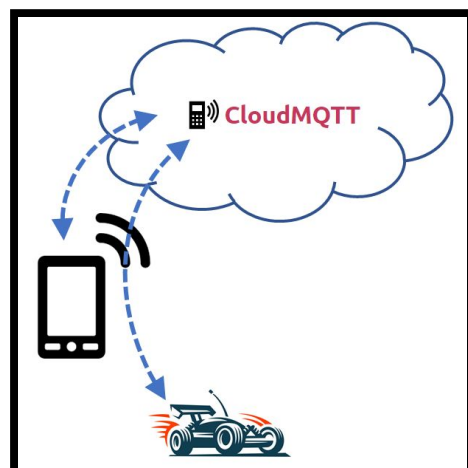
The screenshot shows a web browser window with the address bar displaying 'the-jos.github.io'. The page title is 'R.O.A.C.H.' with the subtitle 'Roaming Ochtend Alarm Clock from Hell'. The main heading is 'Alarm'. Below it, the status is 'Status: Alarm staat UIT'. There are two input fields: 'Datum' with the value '30/12/2019' and 'Tijd' with the value '12:53'. Below these fields is a checkbox labeled 'Alarm aan' which is checked, and a blue button labeled 'Opslaan'. At the bottom left, there is a copyright notice '© 2019 - Roach'.

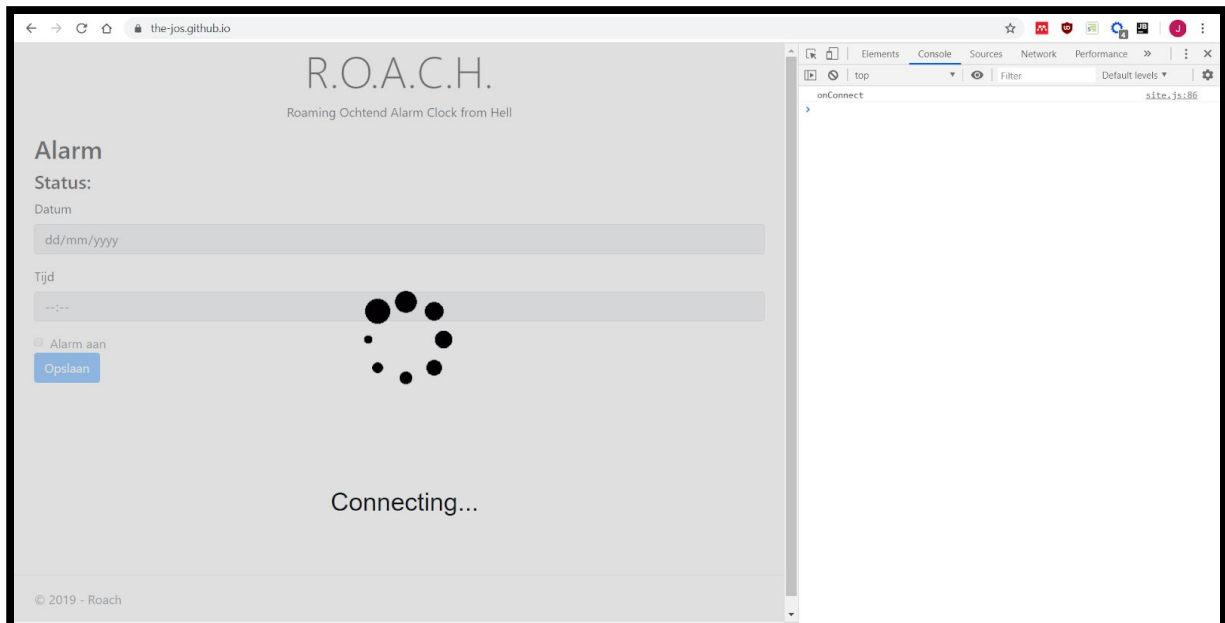
Communicatie

Het idee is dat bij opladen van de webapp de Roach meteen gevraagd wordt om zijn statusinformatie. Zolang er geen antwoord is wordt een Connecting-animatie getoond en is het formulier vergrendeld. Bij het ontvangen van een antwoord wordt deze animatie verborgen, het formulier ontgrendeld en gevuld met de ontvangen informatie.

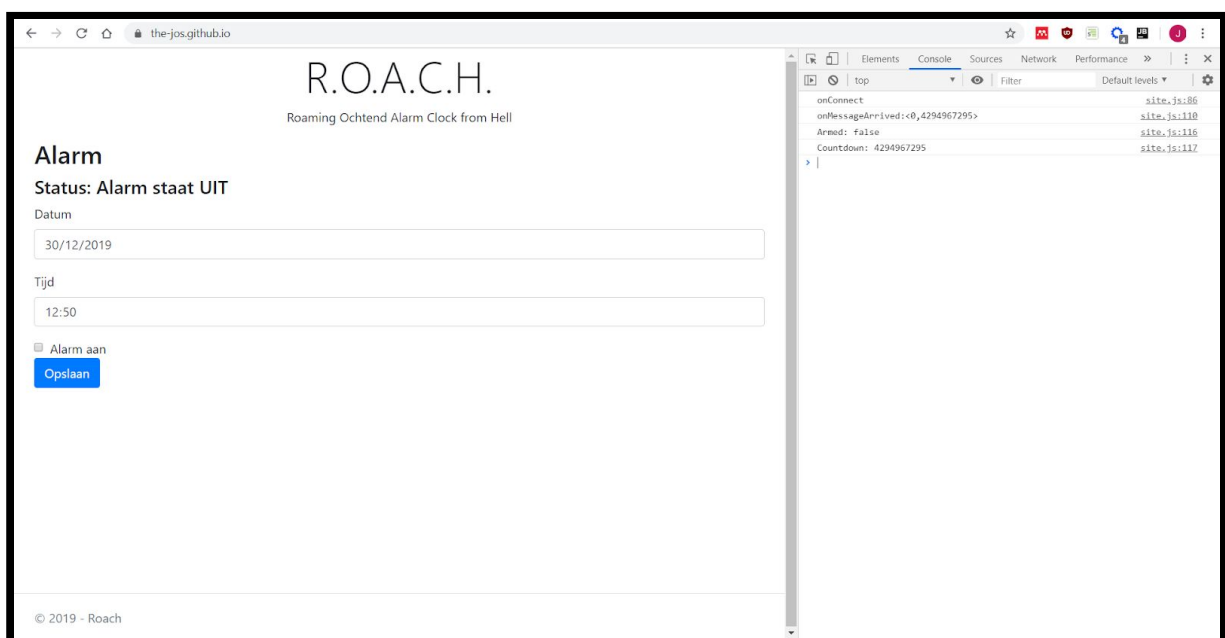
Het drukken van de Opslaan knop zorgt ervoor dat de ingevulde informatie omgezet wordt naar een voor de Roach verstaanbaar bericht. Dit bericht wordt dan via Cloudmqtt verstuurd naar de Roach. De Roach antwoordt met zijn statusinformatie, waardoor het proces opnieuw begint; de statusinformatie wordt dan getoond op de webapp. Deze logica maakt dat de webapp zeer robuust is.

De Roach maakt gebruik van de hotspot-functionaliteit van een telefoontoestel om zich te verbinden.

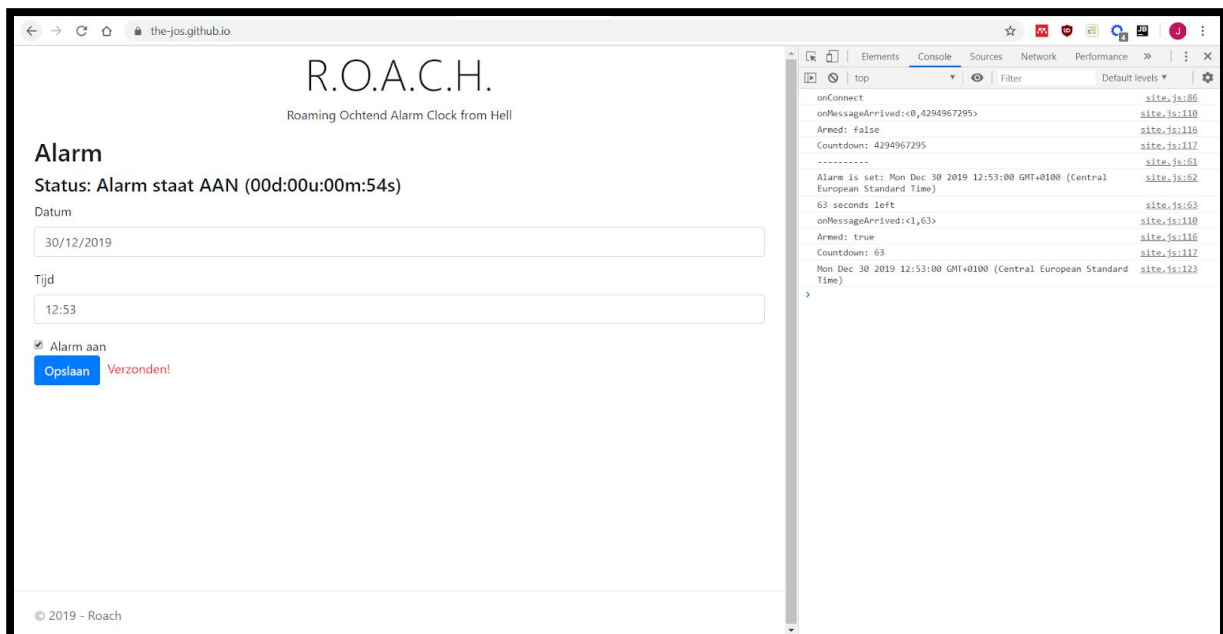




Dit is het scherm die je ziet, voorzien van een animatie, als de Roach uit is of niet verbonden is. De `onConnect()` methode werd aangeroepen en de webapp wacht op een antwoord d.m.v de `onConnect()` functie (console rechts ter illustratie, normaal wordt deze niet weergegeven). Indien de Roach verbonden is verdwijnt de animatie vrijwel meteen.



De webapp heeft in dit scherm een antwoord ontvangen. De `onMessageArrived(message)` functie heeft het bericht vertaald en het formulier ingevuld (console rechts ter illustratie met uitgewisselde data, normaal wordt deze niet weergegeven). In dit geval staat het alarm uit. De huidige tijd wordt door deze functie ingevuld in het formulier en de 'Alarm aan'-checkbox is uitgevinkt gezien het alarm af staat.



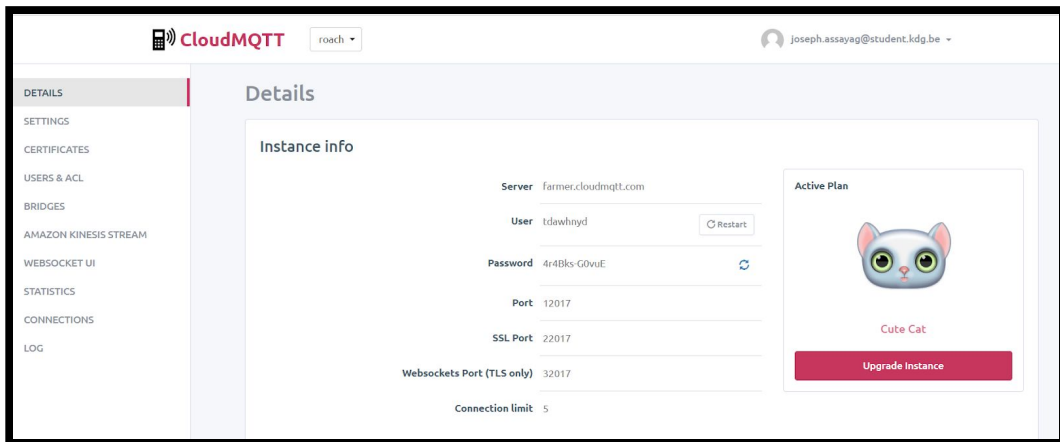
Er werd in bovenstaand scherm zojuist een alarm ingesteld. De countdown is begonnen. De webapp toont bij Status de resterende tijd. Er wordt ook melding gemaakt dat de instelling werd verzonden, naast de Opslaan-knop. Als ontwikkelaar kan je in de console zien dat er na het verzenden van het alarm een nieuw bericht is ontvangen van de Roach; dit is een bericht met de nieuwe statusinformatie. Dergelijke logica zorgt ervoor dat alles dubbel nagekeken wordt. Zo weet de gebruiker zeker dat het alarm weldegelijk is ingesteld. Indien er in de tussentijd een connectie probleem zou zijn ontstaan, zou de Connecting-animatie weer blijven staan. Indien de gebruiker per ongeluk een fout tijdstip ingegeven zou hebben, zou het formulier dit weergeven.

Het afzetten van een bestaand geprogrammeerd alarm kan eenvoudig door de checkbox uit te vinken en op Opslaan te klikken.

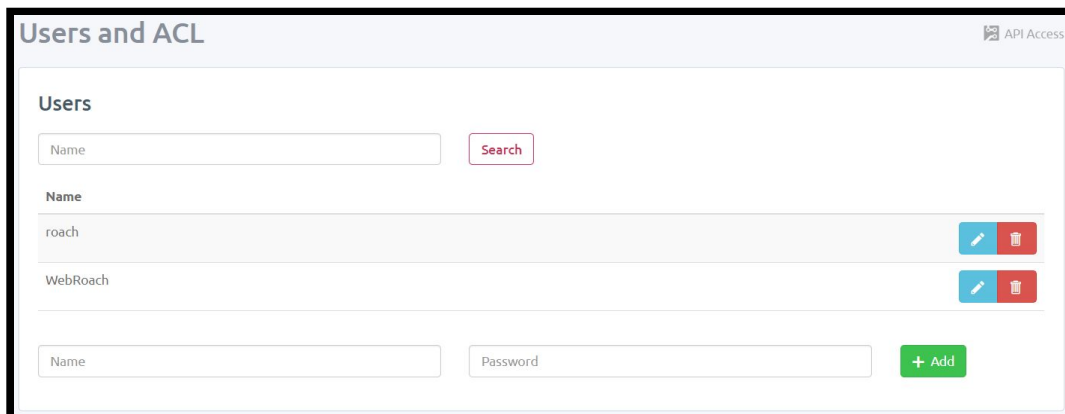
Mqtt

Als provider hebben we voor CloudMqtt gekozen. Deze provider laat ons toe een gratis instantie te creëren met beperkte doch voldoende functionaliteiten.

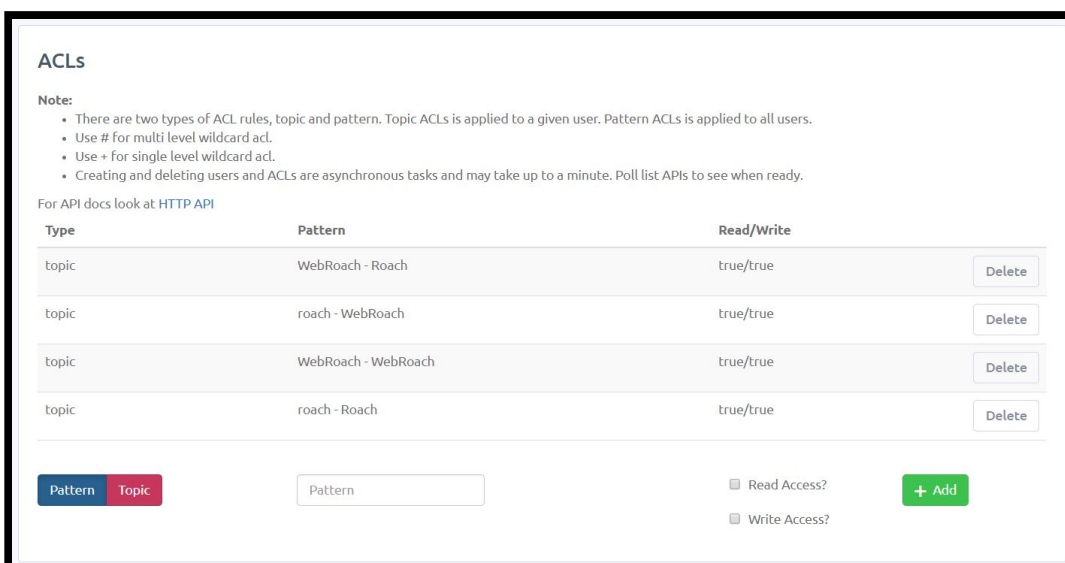
De R.O.A.C.H zal berichten publiceren met de gebruiker 'roach' op de topic 'Roach' en zal luisteren naar de topic 'WebRoach'. De webapp zal berichten publiceren met de gebruiker 'WebRoach' op de topic 'WebRoach' en zal luisteren naar de topic 'Roach'.



Hierboven staan de instellingen van onze CloudMQTT instantie beschreven. Sommige van deze waarden worden gebruikt door de ESP8266 module (Server en Port) en de webapp (Server en Websockets Port).



We hebben twee gebruikers aangemaakt die berichten kunnen lezen en schrijven.



Aan deze gebruikers hebben we rechten toegekend zodat ze kunnen lezen en schrijven op beide topics.

Code

HTML

De HTML code bevat enkel de layout van het formulier. De javascript en css code maakt voornamelijk gebruik van de 'class' en 'id' attributen om de functionaliteit van elk veld te bepalen. De javascript code zal naargelang de toestand de nodige attributen toevoegen of verwijderen, om de gewenste weergave te bekomen (zie sectie Javascript).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>R.O.A.C.H.</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.mi
n.css" />
    <link rel="stylesheet" href="/css/site.css" />
  </head>
  <body>

    <header>
      <div class="text-center">
        <h1 class="display-4">R.O.A.C.H.</h1>
        <p>Roaming Ochtend Alarm Clock from Hell</p>
      </div>
    </header>
    <div class="container">
      <main role="main" class="pb-3">

        <form>
          <fieldset id="fields" disabled>
            <h2>Alarm</h2>
            <h4>Status: <span
id="alarmstate"></span><span id="remainingtime"></span></h4>
            <div class="form-group">
              <label
for="datum">Datum</label><input class="form-control" id="datum"
name="datum" type="date" required/>
            </div>
            <div class="form-group">
              <label
for="tijd">Tijd</label><input class="form-control" id="tijd" name="tijd"
type="time" required/>
            </div>
          </fieldset>
        </form>
      </main>
    </div>
  </body>
</html>
```

```

        <div class="form-check">
            <input class="form-check-input"
id="armed" name="armed" type="checkbox"/><label class="form-check-label"
for="armed">Alarm aan</label>
        </div>
        <button id="submitButton"
type="button" class="btn btn-primary">Opslaan</button><span
id="result"></span>
    </fieldset>
</form>
<div id="loading"></div>
</main>
</div>
<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2019 - Roach
    </div>
</footer>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.
js" type="text/javascript"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"><
/script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.bund
le.min.js"></script>
    <script src="/js/site.js"></script>
</body>
</html>

```

Javascript

De Javascript code vormt het hart van de functionaliteit van de webapp.

Ten eerste zorgt de **onConnect()** functie voor wat er moet gebeuren als de webapp verbinding gemaakt heeft met de Cloudmqtt dienst. In onze setup stopt deze functie de onConnect animatie en vraagt het onmiddellijk de statusinformatie van de Roach op via de functie **requestInfoFromRoach()**. Zo zorgt elke pagina refresh ervoor dat de laatste informatie wordt opgevraagd uit de Roach.

Ten tweede zorgt de **submitAlarm()** functie ervoor dat de waarden ingevuld in het formulier omgezet en gestuurd worden naar de Cloudmqtt dienst. Indien de checkbox 'Alarm aan' aangevinkt is, wordt een nieuw alarm ingesteld. Indien deze checkbox uitgevinkt is, wordt het alarm afgezet. Ook hier wordt op het einde **requestInfoFromRoach()** opgeroepen. Op die manier zorgt elke handeling op de webapp voor het ophalen van de recentste statusinformatie.

Ten derde zorgt de functie **onMessageArrived(message)** voor het omzetten van het antwoord van de Roach in gegevens dat op de webapp getoond kunnen worden.

Tenslotte zorgt de functie timer() voor de teller die aftelt op de webapp indien een alarm is ingesteld.

```
var hostname = "farmer.cloudmqtt.com";
var port = 32017;
var inputs = document.getElementsByTagName('input');
var fieldset = document.getElementById('fields');
var submitButton = document.getElementById('submitbutton');
var resultSpan = document.getElementById('result');
var alarmstateSpan = document.getElementById('alarmstate');
var remainingtimeSpan = document.getElementById('remainingtime');
var countdowntime;
var requestCounter = 0;
var isArmed = false;
var options = {
    useSSL: true,
    userName: "roach",
    password: "roach",
    cleanSession: true,
    onSuccess: onConnect,
    onFailure: doFail
};
var countdownTimer = setInterval(timer, 1000);
var seconds = 0;

// Create a client instance
client = new Paho.MQTT.Client(hostname, Number(port), "clientId");

// set callback handlers
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;

// connect the client
client.connect(options);

submitButton.addEventListener("click", submitAlarm);
for (var i = 0; i < inputs.length; i++) {
    inputs[i].addEventListener("change", function () {
        resultSpan.innerHTML = "";
    });
}

function submitAlarm() {
    var alarmAanIsChecked = inputs[2].checked;
    if (alarmAanIsChecked) {
        // set alarm
        var alarmdate = inputs[0].valueAsDate;
        var hours = inputs[1].valueAsDate.getHours();
        var minutes = inputs[1].valueAsDate.getMinutes();
        alarmdate.setHours(hours - 1);
        alarmdate.setMinutes(minutes);
        alarmdate.setSeconds(00);

        var dif = alarmdate.getTime() - new Date().getTime();
        var secondsFromT1ToT2 = dif / 1000;
        var secondsBetweenDates = Math.floor(secondsFromT1ToT2);
        if (secondsBetweenDates < 1) {
```

```

        resultSpan.innerText = "Ongeldige Tijd!";
    } else {
        console.log("-----");
        console.log("Alarm is set: " + alarmdate);
        console.log(secondsBetweenDates + " seconds left");
        var message = new Paho.MQTT.Message("<N," +
secondsBetweenDates + ">");
        resultSpan.innerText = "Verzonden!";
        message.destinationName = "WebRoach";
        client.send(message);
        fieldset.setAttribute("disabled", "disabled");
        document.getElementById('loading').style.display = 'block';
    }
} else {
    // cancel alarm
    var message = new Paho.MQTT.Message("<Z,0>");
    resultSpan.innerText = "Alarm geannuleerd!";
    message.destinationName = "WebRoach";
    client.send(message);
    fieldset.setAttribute("disabled", "disabled");
    document.getElementById('loading').style.display = 'block';
}
}

// called when the client connects
function onConnect() {
    // Once a connection has been made, make a subscription and send a
    message.
    console.log("onConnect");
    client.subscribe("Roach");
    requestInfoFromRoach();
}

function requestInfoFromRoach() {
    var message = new Paho.MQTT.Message("<A,0>");
    message.destinationName = "WebRoach";
    client.send(message);
}

function doFail() {
    console.log("doFail");
}

// called when the client loses its connection
function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0) {
        console.log("onConnectionLost:" + responseObject.errorMessage);
    }
}

// called when a message arrives
function onMessageArrived(message) {
    console.log("onMessageArrived:" + message.payloadString);
    var payload = message.payloadString.match(/<(.*)>/);
    if (payload != null && payload.length > 0 &&
payload.pop().split(",").length === 2) {
        var extract =

```

```

message.payloadString.match(/<(.*?)>/).pop().split(",");
isArmed = extract[0] > 0;
countdownTime = parseInt(extract[1]);
console.log("Armed: " + isArmed);
console.log("Countdown: " + countdownTime);
var alarmTime = new Date();
if (isArmed) {
    seconds = countdownTime;
    countdownTimer = setInterval(timer, 1000);
    alarmTime.setSeconds(new Date().getSeconds() +
countdownTime);
    console.log(alarmTime);
    alarmstateSpan.innerText = "Alarm staat AAN";
} else {
    alarmstateSpan.innerText = "Alarm staat UIT";
}
inputs[0].valueAsDate = alarmTime;
inputs[1].value = String(alarmTime.getHours()).padStart(2, '0') +
":" +
String(alarmTime.getMinutes()).padStart(2, '0');
inputs[2].checked = isArmed;
document.getElementById('loading').style.display = 'none';
fieldset.removeAttribute("disabled");
requestCounter = 0;
} else if (requestCounter < 3) {
    requestInfoFromRoach();
}
}

function timer() {
    if (isArmed) {
        var days = Math.floor(seconds / 24 / 60 / 60);
        var hoursLeft = Math.floor((seconds) - (days * 86400));
        var hours = Math.floor(hoursLeft / 3600);
        var minutesLeft = Math.floor((hoursLeft) - (hours * 3600));
        var minutes = Math.floor(minutesLeft / 60);
        var remainingSeconds = seconds % 60;

        function pad(n) {
            return (n < 10 ? "0" + n : n);
        }

        remainingtimeSpan.innerText = " (" + pad(days) + "d:" +
pad(hours) + "u:" + pad(minutes) + "m:" + pad(remainingSeconds) + "s)";
        if (seconds == 0) {
            clearInterval(countdownTimer);
            remainingtimeSpan.innerText = "";
        } else {
            seconds--;
        }
    } else {
        clearInterval(countdownTimer);
        remainingtimeSpan.innerText = "";
    }
}
}

```

CSS

De CSS code speelt een kleine maar niet onbelangrijke rol; het wordt aangewend om de Connecting-animatie te tonen op vraag van de Javascript wijzigingen. Concreet komt het erop neer dat de loading <div> niet getoond zal worden zodra een bericht van de Roach via Cloudmqtt wordt ontvangen.

```
a.navbar-brand {
  white-space: normal;
  text-align: center;
  word-break: break-all;
}

html {
  font-size: 14px;
}

@media (min-width: 768px) {
  html {
    font-size: 16px;
  }
}

.border-top {
  border-top: 1px solid #e5e5e5;
}

.border-bottom {
  border-bottom: 1px solid #e5e5e5;
}

.box-shadow {
  box-shadow: 0 .25rem .75rem rgba(0, 0, 0, .05);
}

button.accept-policy {
  font-size: 1rem;
  line-height: inherit;
}

html {
  position: relative;
  min-height: 100%;
}

body {
  /* Margin bottom by footer height */
  margin-bottom: 60px;
}

.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  white-space: nowrap;
```

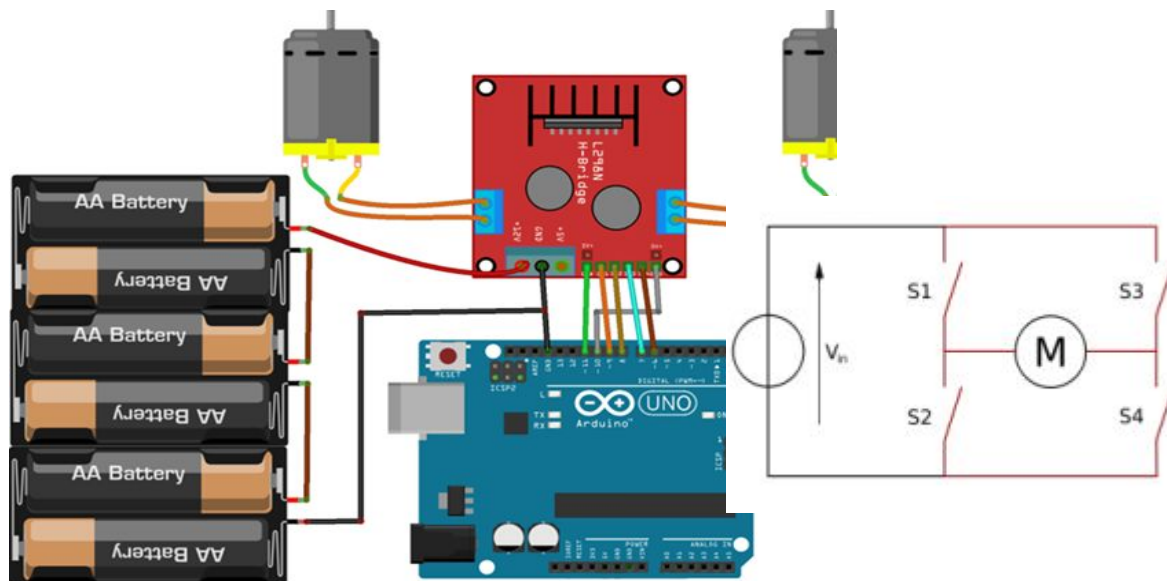
```
/* Set the fixed height of the footer here */
height: 60px;
line-height: 60px; /* Vertically center the text there */
}

#loading {
    display: block;
    position: absolute;
    top: 0;
    left: 0;
    z-index: 100;
    width: 100vw;
    height: 100vh;
    background-color: rgba(192, 192, 192, 0.5);
    background-image: url("../images/wait.gif");
    background-repeat: no-repeat;
    background-position: center;
}
#loading::after{
    content: "Connecting...";
    font: 2em bold Helvetica, sans-serif;
    position: absolute;
    left: 50%;
    margin-left: -3em;
    bottom: 20%;
}
#result{
    color: red;
    margin-left: 10px;
}
```


Arduino

Electronica

Motor controller + dc motors + batteryPack



L298N H-bridge

De module, met de L298N als controller, is een dubbele H-bridge motor driver met weinig warmteproductie en interferentie. De module kan zelfs piekspanning van 46v verwerken en een piek stroom van 3A.

Deze module kan tot 2A aansturen met een vermogen van 25w

H-bridge

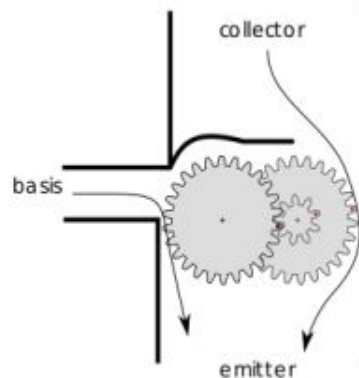
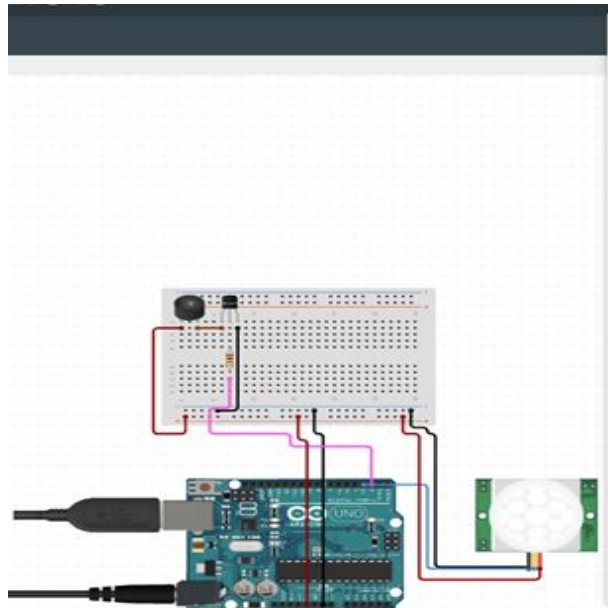
Als je bij een DC-motor de draden wisselt, dan zal deze ook in de andere richting beginnen draaien. Een H-bridge is een elektrische [schakeling](#) die het mogelijk maakt om de waarde van twee aansluitingen te verwisselen. Daarmee kan bijvoorbeeld de draairichting van een DC-motor omgekeerd worden. De waarde wordt bepaald door de stand van vier schakelaars.

Battery Pack

Onze dc motor werkt tussen 3-6v. Wij gebruiken 4xAA batterypack om onze dc motoren te ondersteunen.

Video: <https://drive.google.com/file/d/1NFGTmfocFjcGxAhELhTsJ1eOPQNFnwqK/view?usp=sharing>

Motion sensor + buzzer



Buzzer - NPN Transistor

De buzzer gebruiken we om het alarm af te spelen. De buzzer verbinden we met een transistor dat het geluid versterkt en voorkomt dat vreemde geluiden zouden afspelen. We gebruiken ook een weerstand van 1K om het geluid nog verder te verfijnen.

Werking transistor

Als er een kleine stroom vloeit van basis naar emitter, dan vloeit er een grotere stroom van collector naar emitter. De grotere stroom is altijd een bepaald veelvoud van de kleine stroom. Men noemt dit de versterkingsfactor. Een transistor versterkt dus een stroom. Als er een kleine stroom loopt over het linker-rad, dan zal dat een grote stroom creëren langs het rechter-rad. De verhouding is steeds dezelfde.

MotionSensor

De PIR motion sensor wordt gebruikt voor: als het beweging detecteert zou de wagen moeten weg rijden van de eigenaar zodat de eigenaar zeker wakker wordt tijdens het afzetten van de alarm.

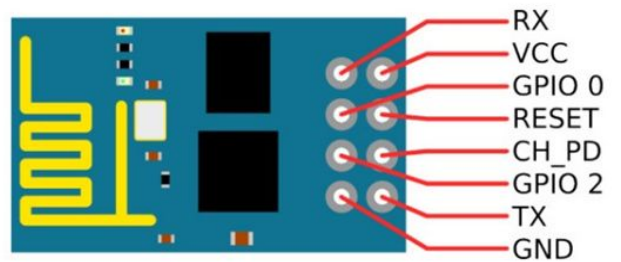
Pinout:

- GND – connect to ground
- OUT – connect to an Arduino digital pin
- 5V – connect to 5V

Video: https://drive.google.com/file/d/1hWBtXbCBohtZUeintFDRPVtSefC1i_Hy/view?usp=sharing

Wifi (Esp8266) module

ESP8266 pin	Arduino pin
RX	11
TX	10
GND	GND
VCC	3.3V
CH_PD(chip power-down)	3.3V



In het menu 'Preferences' moet je deze link invoeren::

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Bootmode verbindingen (om code te flashen)

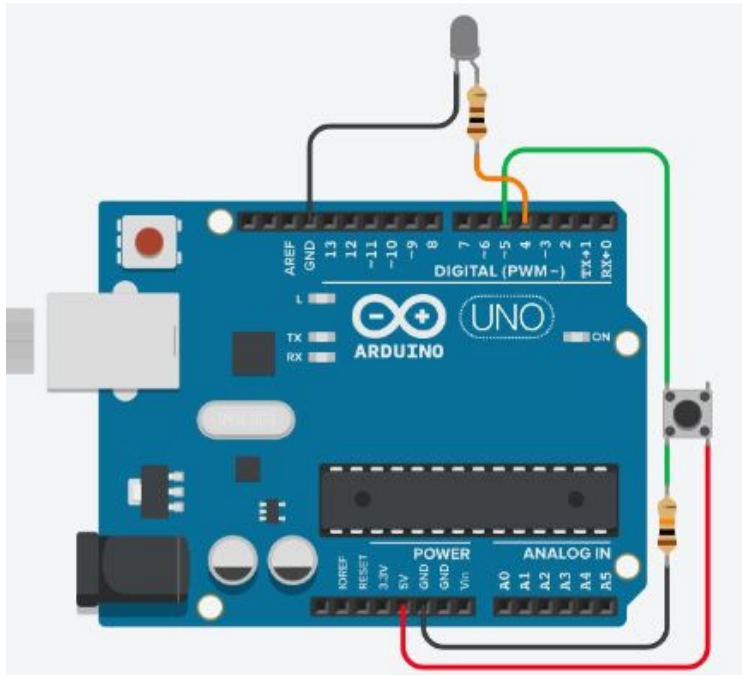
1. Arduino RST → GND Arduino
2. Esp8266 GPIO 0 → GND Arduino
3. Esp8266 RX → RX Arduino
4. Esp8266 TX → TX Arduino

Weerstanden

We gebruiken 2 weerstanden van 1k en 2k voor pin RX.

$$V_{out} = \frac{V_s \times R_2}{(R_1 + R_2)} = \frac{5 \times 2}{(1+2)} = 3.33v$$

Alarm turn off button



Voor de LED wordt een weerstand gebruikt, verder is de verbinding vrij eenvoudig. De schakelaar is verbonden met VCC 5V en zet enkel pin 5 onder spanning indien de knop ingedrukt is. De weerstand zorgt ervoor dat er geen kortsluiting ontstaat met GND.

Een led gebruikt tussen 1.8v en 3.3v, afhankelijk van de kleur. De Pushbutton bevat een weerstand omdat het aanbevolen is om uw led max 20mA te geven aangezien de pinouts van arduino 40mA geven.

De drukknop gebruikt een pull down weerstand van 1K ohm. Omdat er in de schakelaar en draden of andere componenten nog reststroom kan zitten, is het mogelijk dat de IC op dat moment toch nog een één ziet in plaats van een nul. Vooral bij snelle frequenties waarbij er snel geschakeld wordt tussen 5V en 0V kan storing ontstaan. De weerstand voorkomt dit.

Code

Arduino UNO

In deze sectie wordt de code besproken die gebruikt wordt om de Arduino UNO aan te sturen.

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <anyrtttl.h>
#include <binrtttl.h>
#include <pitches.h>
#include <avr/wdt.h>
```

De includes zorgen voor de nodige externe library code. Gezien we in Visual Studio Code programmeren voegen we expliciet de Arduino library toe. De SoftwareSerial library zullen we gebruiken voor de communicatie tussen de Arduino UNO en de ESP8266. De anyrttl, binrttl en pitches libraries dienen voor het afspelen van de muziek als het alarm afgaat. De avr/wdt library is nodig voor de watchdog, die nagaat of de Arduino succesvol zijn loop blijft herhalen.

```
#define RIGHTFWD_PIN 4 // IN1 of L293N attached to pin 4 arduino = RIGHT FORWARD
#define RIGHTBCK_PIN 5 // IN2 of L293N attached to pin 5 arduino = RIGHT BACKWARD
#define LEFTBCK_PIN 6 // IN3 of L293N attached to pin 6 arduino = LEFT BACKWARD
#define LEFTFWD_PIN 7 // IN4 of L293N attached to pin 7 arduino = LEFT FORWARD
#define SENSOR_PIN 3 // Motion sensor attached to pin 3 arduino
#define BUZZER_PIN 2 // buzzer attached to pin 2 arduino
#define BUTTON_PIN 8 // button attached to pin 8 arduino
#define PIN_IN 10 // Software Serial to ESP
#define PIN_OUT 11 // Software Serial to ESP
#define LED_PIN 12 // LED button

#define STOP 0
#define FORWARD 1
#define BACKWARD 2
#define SPINLEFT 3
#define SPINRIGHT 4

#define OFF 0
#define ON 1
```

Dit zijn de #defines die als constanten zijn gedeclareerd om onze code te verduidelijken.

```

byte motionState = STOP;
int motionTime = 0;
byte alarmState = OFF;
SoftwareSerial EspSerial(PIN_IN, PIN_OUT);
byte buttonState = LOW;
const char *mario =
"mario:d=4,o=5,b=100:16e6,16e6,32p,8e6,16c6,8e6,8g6,8p,8g,8p,8c6,16p,8g,1
6p,8e,16p,8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,16d6,8b,16p,
8c6,16p,8g,16p,8e,16p,8a,8b,16a#,8a,16g.,16e6,16g6,8a6,16f6,8g6,8e6,16c6,
16d6,8b,8p,16g6,16f#6,16f6,16d#6,16p,16e6,16p,16g#,16a,16c6,16p,16a,16c6,
16d6,8p,16g6,16f#6,16f6,16d#6,16p,16e6,16p,16c7,16p,16c7,16c7,p,16g6,16f#
6,16f6,16d#6,16p,16e6,16p,16g#,16a,16c6,16p,16a,16c6,16d6,8p,16d#6,8p,16d
6,8p,16c6";
unsigned long countdownTime = 0UL - 1UL; // in seconds
unsigned long motionStartTime = 0;
bool armed = false;
unsigned long lastTick = millis();
unsigned long currentMillis = millis();
unsigned long previousLedMillis = 0;
byte ledState = LOW;

const byte numChars = 10;
char receivedChars[numChars];
char tempChars[numChars]; // temporary array for use when parsing
char webrequest[numChars] = {0};

boolean newData = false;

```

Vervolgens gebeurt de declaratie en (gedeeltelijk) de initialisatie van de globale variabelen. De meest opvallende is de mario constante; dit is het lied dat gebruikt wordt als het alarm afgaat. Het formaat heet Ring Tone Transfer Language (RTTL) en is uitgevonden door Nokia om ringtones te transfereren naar telefoontoestellen.

Verder worden er voornamelijk 3 types variabelen gebruikt: status variabelen (motionState, buttonState, alarmState, newData, armed, ledState), tijdgerelateerde variabelen (countdownTime, motionStartTime, lastTick, currentMillis, previousLedMillis) en variabelen die te maken hebben met het parsen van de seriële verbinding met de ESP8266 (EspSerial, numChars, receivedChars, tempChars, webrequest).

```

void startMotion();
void checkMotionState();
void alarmOn();
void alarmOff();
void checkButton();
void doAlarmstate();
void checkSerial();

```

```

void checkCountdownTime();
void recvWithStartEndMarkers();
void sendInfo();
void parseData();
void carSpinLeft();
void carSpinRight();
void carForward();
void carBackward();
void carStop();
void turnOffAlarm();
void blinkAlarmLed();

```

We gaan eerst alle functies declareren. De compiler is onderhevig aan de declaratie van functies in de juiste volgorde; je kan normaal gezien enkel een functie gebruiken die voordien reeds geschreven staat. Door alle functies in de code op deze manier bovenaan op te lijsten hoef je hier geen rekening mee te houden. Bovendien is het ook ineens een handig lijstje.

```

void setup()
{
  wdt_enable(WDTO_8S); //watchdog timer: reset after no loop for 8
seconds

  pinMode(BUZZER_PIN, OUTPUT); //BUZZER_PIN
  pinMode(13, OUTPUT);          //led indicator when singing a note
  pinMode(RIGHTFWD_PIN, OUTPUT);
  pinMode(RIGHTBCK_PIN, OUTPUT);
  pinMode(LEFTBCK_PIN, OUTPUT);
  pinMode(LEFTFWD_PIN, OUTPUT);
  pinMode(SENSOR_PIN, INPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(PIN_IN, INPUT);
  pinMode(PIN_OUT, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  EspSerial.begin(9600);
  Serial.println("Hello World!");
}

```

Deze functie wordt bij het opstarten van de Arduino automatisch opgeroepen. We bepalen hier de mode van de pins en starten de seriële communicatie met de ESP8266 en de reguliere seriële Rx/Tx. Deze laatste kan verbonden worden via USB met een computer om het toestel te monitoren. Ook belangrijk is de eerste lijn (`wdt_enable`) die de watchdogtimer initialiseert op 8 seconden. Indien een loop langer duurt dan 8 seconden zal de arduino zichzelf resetten. In principe zou een dergelijke reset niet mogen gebeuren.

```

void loop()
{
  checkSerial();
  if (armed)
  {
    checkCountdownTime();
  }
  doAlarmstate();
  wdt_reset();
}

```

De loop functie is de tweede functie, naast de setup() functie, die automatisch door de Arduino wordt uitgevoerd. Zoals de naam het aangeeft wordt deze functie voortdurend herhaald. Dit is dus in feite het hart van de logica van ons project. Eerst wordt inkomende data nagekeken. Vervolgens wordt indien nodig de countdown geverifieerd en aangepast. Hierna wordt de huidige alarmState logica uitgevoerd. Als laatste wordt de reset functie van de watchdogtimer aangesproken (hopelijk binnen de 8 seconden, anders zal de Arduino zichzelf resetten). In ons project hebben we ervoor gestreefd om deze loop zo snel mogelijk te laten doorlopen. Er wordt nergens in de code gebruik gemaakt van delay of sleep functionaliteiten. Dit is nodig om te voorkomen dat de watchdogtimer een reset uitvoert en zorgt er ook voor dat de seriële communicatie veel betrouwbaarder is.

```

void checkMotionState()
{
  if (alarmState == ON)
  {
    if (millis() > motionStartTime + motionTime)
    {
      motionStartTime = millis();
      if (motionState == SPINLEFT || motionState == SPINRIGHT)
      {
        motionState = random(1, 3);
        if (motionState == FORWARD)
        {
          carForward();
        }
        else if (motionState == BACKWARD)
        {
          carBackward();
        }
      } else if (motionState == FORWARD || motionState == BACKWARD)
      {
        motionState = STOP;
        carStop();
      }
    }
  }
}

```


De variabele `motionTime` is een random getal tussen 1000 en 2000 (zie methode `startMotion()`). Deze logica zorgt ervoor dat indien de auto reeds gedraaid heeft deze nu rechtdoor gaat (willekeurig vooruit of achteruit). Indien de auto reeds rechtdoor geweest is komt deze nu tot stilstand. Concreet betekent het dat als deze methode wordt aangeroepen (bij motionsensor detectie) de auto al eerst 360 graden gedraaid heeft gedurende 1 tot 2 seconden, en nu 1 tot 2 seconden een rechte lijn zal rijden, om daarna uiteindelijk tot stilstand te komen (tot de volgende bewegingsdetectie plaatsvindt).

```
void checkSerial()
{
    recvWithStartEndMarkers();
    if (newData == true)
    {
        strcpy(tempChars, receivedChars);
        // this temporary copy is necessary to protect the original data
        // because strtok() used in parseData() replaces the commas with \0
        parseData();
        newData = false;
    }
}
```

Dit is het hoogste niveau van de code om de seriële verbinding op binnenkomende data na te gaan. Indien er nieuwe data gedetecteerd wordt, wordt deze data overgezet naar een tijdelijke char array en geparst.

```
void checkCountdownTime()
{
    currentMillis = millis();
    if (currentMillis - lastTick >= 1000)
    {
        lastTick = currentMillis;
        countdownTime--;
    }

    if (countdownTime == 0)
    {
        armed = false;
        alarmState = ON;
        countdownTime = 0UL - 1UL;
    }
}
```

Deze functie zorgt voor het aftellen en voor het afgaan van het alarm als de countdown 0 bereikt.

```

void alarmOn()
{
  if (!anyrtttl::nonblocking::isPlaying())
  {
    anyrtttl::nonblocking::begin(BUZZER_PIN, mario); // start to play
music
  }
  else
  {
    anyrtttl::nonblocking::play();
  }
  checkButton();
  if (alarmState == ON)
  {
    blinkAlarmLed();
    byte motionDetected = 0;
    motionDetected = digitalRead(SENSOR_PIN); // read motion sensor value
    if (motionDetected == HIGH && motionState == STOP)
    {
      Serial.println("motion detected");
      startMotion();
    }
    checkMotionState();
  }
}
}

```

Dit is de logica voor als het alarm aan het afgaan is. De eerste keer wordt de begin() methode aangesproken om de muziek te starten. De opvolgende loops wordt de play() methode gebruikt om de noten af te spelen. De achterliggende library speelt enkel een noot af indien de timing juist is en doet dit op niet-blokkerende wijze. Op die manier wordt de main loop nooit tegengehouden. Verder wordt er nagekeken of de knop wordt ingedrukt, wordt de functie aangesproken die de LED van de drukknop doet flinkeren en wordt er nagegaan of er beweging gedetecteerd werd. Door de huidige motionState ook te controleren zal de auto enkel startMotion() aanspreken indien het op dat ogenblik stil staat. De functie checkmotionState() wordt tenslotte nog uitgevoerd om te zorgen dat de auto van richting veranderd op het juiste moment.

```

void blinkAlarmLed(){
  unsigned long ledMillis = millis();
  if (ledMillis - previousLedMillis >= 250) {
    // save the last time you blinked the LED
    previousLedMillis = ledMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {

```

```

    ledState = LOW;
}

// set the LED with the ledState of the variable:
digitalWrite(LED_PIN, ledState);
}
}

```

Deze logica zorgt ervoor dat de LED van de drukknop knippert.

```

void startMotion()
{
    motionState = random(3, 5);
    motionTime = random(1000, 2000);
    motionStartTime = millis();
    if (motionState == SPINLEFT)
    {
        carSpinLeft();
    }
    else if (motionState == SPINRIGHT)
    {
        carSpinRight();
    }
}

```

Deze code initialiseert de eerste beweging van de auto. De auto begint met een 360 graden spin gedurende 1 tot 2 seconden. De juiste duur wordt willekeurig bepaald. Ook de richting is willekeurig. In latere loops zal checkMotionState() zorgen voor de verdere bewegingen van de Roach.

```

void alarmOff()
{
    if (anyrtttl::nonblocking::isPlaying())
    {
        Serial.println("ALARM TURNED OFF !!");
        turnOffAlarm();
    }
}

```

Deze code wordt uitgevoerd indien het alarm uit staat en er geen alarm ingesteld is. Eigenlijk gebeurt er hier niets. Er wordt enkel voor gezorgd dat een eventueel alarm dat nog voordien aan het afgaan was, wordt afgezet indien nodig.

```

void checkButton()
{
    buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == HIGH)
    {
        Serial.println("-Button Pressed-");
        turnOffAlarm();
    }
}

```

Hier wordt nagegaan of de knop wordt ingedrukt. Zoja, dan wordt het alarm afgezet.

```

void turnOffAlarm()
{
    digitalWrite(LED_PIN, LOW);
    anyrtttl::nonblocking::stop();
    alarmState = OFF;
    armed = OFF;
    countdownTime = 0UL - 1UL;
    lastTick = 0;
    carStop();
}

```

Deze code zorgt ervoor dat het alarm wordt afgezet en alle relevante variabelen terug naar hun startwaarden keren. Deze code wordt opgeroepen bij het indrukken van de knop, bij het annuleren van een alarm via de webapp en bij het instellen van een nieuw alarm via de webapp.

```

void doAlarmstate()
{
    switch (alarmState)
    {
        case ON:
            // Serial.println("do Alarm_ON");
            alarmOn();
            break;
        case OFF:
            // Serial.println("do Alarm_OFF");
            alarmOff();
            break;
    }
}

```

Hier wordt bepaald welke code aangeroepen dient te worden. Dit gebeurt aan de hand van de huidige alarmState.

```

void recvWithStartEndMarkers()
{
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (EspSerial.available() > 0 && newData == false)
    {
        rc = EspSerial.read();
        Serial.print(rc);

        if (recvInProgress == true)
        {
            if (rc != endMarker)
            {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars)
                {
                    ndx = numChars - 1;
                }
            }
            else
            {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker)
        {
            recvInProgress = true;
        }
    }
}

```

Deze code ontvangt data over de software seriële verbinding met de ESP8266. Het kijkt de binnenkomende data na op de start en eind markers '<' en '>' na en zorgt voor de volledigheid en integriteit van een binnenkomend bericht. Als de eindmarker '>' gedetecteerd wordt, wordt newData op true gezet en zal parseData() bij de volgende loop van checkSerial() de data doornemen. Indien een nieuwe startmarker '<' wordt gedetecteerd dan wordt al het voorgaande weggegooid. Dit is een systeem dat niet 100% waterdicht is, maar dat meer dan voldoende robuust is voor ons doeleinde.

```

void sendInfo()
{
    String armedState = armed ? "1" : "0";
    String data = "<" + armedState + "," + countdownTime + ">";
    EspSerial.println(data);
}

```

Dit is de methode die ervoor zorgt dat de webapp de huidige statusinformatie zal ontvangen. De huidige relevante data wordt hier eenvoudig weggeschreven naar de ESP8266, waar het bericht ingepakt zal worden in een message en verstuurd zal worden naar CloudMqtt.

```

void parseData()
{
    char *strtokIndx; // this is used by strtok() as an index

    strtokIndx = strtok(tempChars, ","); // get the first part - the string
    strcpy(webrequest, strtokIndx);      // copy it to webrequest

    switch (webrequest[0])
    {
        case 'A':
            // website requests current Alarm configuration
            {
                sendInfo();
            }
            break;
        case 'N':
            // website sets New alarm
            {
                turnOffAlarm();
                strtokIndx = strtok(NULL, ","); // this continues where the
previous call left off
                countdownTime = atol(strtokIndx); // convert this part to unsigned
long
                Serial.print("New Alarm Countdown: ");
                Serial.println(countdownTime);
                currentMillis = millis();
                lastTick = millis();
                armed = true;
                sendInfo();
            }
            break;
        case 'Z':
            // website Zeroes alarm (cancel/disarm)
            {

```

```

        turnOffAlarm();
        sendInfo();
    }
    break;
default:
    // invalid command in webrequest, do nothing
    Serial.print("Invalid request received: ");
    Serial.println(webrequest[0]);
    break;
}
}

```

Deze methode ontcijfert ontvangen berichten en zet ze om naar de juiste actie. Als de methode loop() het hart is van de code, dan is dit het brein. In eerste instantie wordt het bericht verder uitgesplitst (ontdaan van startmarker, de komma en de eindmarker). Uit webrequest[0] wordt het commando uitgelezen. Enkel indien het gaat om een nieuw alarm wordt de long value uitgelezen en als nieuwe countdown ingesteld en worden een aantal variabelen gereset. In alle gevallen wordt nieuwe statusinformatie naar de webapp gestuurd, behalve indien het om een ongeldig commando gaat.

```

void carSpinLeft()
{
    digitalWrite(RIGHTFWD_PIN, 0);
    digitalWrite(RIGHTBCK_PIN, 180);
    digitalWrite(LEFTBCK_PIN, 0);
    digitalWrite(LEFTFWD_PIN, 180);
}

void carSpinRight()
{
    digitalWrite(RIGHTFWD_PIN, 180);
    digitalWrite(RIGHTBCK_PIN, 0);
    digitalWrite(LEFTBCK_PIN, 180);
    digitalWrite(LEFTFWD_PIN, 0);
}

void carForward()
{
    digitalWrite(RIGHTFWD_PIN, 180);
    digitalWrite(RIGHTBCK_PIN, 0);
    digitalWrite(LEFTBCK_PIN, 0);
    digitalWrite(LEFTFWD_PIN, 180);
}

void carBackward()

```

```

{
  digitalWrite(RIGHTFWD_PIN, 0);
  digitalWrite(RIGHTBCK_PIN, 180);
  digitalWrite(LEFTBCK_PIN, 180);
  digitalWrite(LEFTFWD_PIN, 0);
}

void carStop()
{
  digitalWrite(RIGHTFWD_PIN, 0);
  digitalWrite(RIGHTBCK_PIN, 0);
  digitalWrite(LEFTBCK_PIN, 0);
  digitalWrite(LEFTFWD_PIN, 0);
}

```

Deze code zorgt ervoor dat de Roach in de juiste richting rijdt. Hierbij wordt de H-bridge aangesproken, die op zijn beurt de 2 motoren zal bedienen.

ESP8266

Wij hebben ervoor gekozen om een ESP8266-01 te gebruiken en deze serieel te laten communiceren met een software seriële verbinding van de Arduino. Hieronder de code met uitleg.

```

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

```

De ESP8266 maakt gebruik van twee libraries. De eerste library zorgt voor de WiFi verbinding. De tweede library zorgt voor de Mqtt functionaliteiten.

```

const char* ssid = "xx";
const char* password = "cqnvqvrq";
const char* mqtt_server = "farmer.cloudmqtt.com";
const int mqttPort = 12017;
const char* mqttUser = "roach";
const char* mqttPassword = "roach";

const byte numChars = 100;
char receivedChars[numChars]; // an array to store the received data
boolean newData = false;
unsigned long lastReconnectAttempt = 0;
const int reconnectDelay = 5000;

```



```
WiFiClient espClient;  
PubSubClient client(espClient);
```

Dit zijn de globale variabelen die gebruikt worden doorheen onze code. De eerste twee variabelen (ssid en password) zijn nodig voor de Wifi configuratie. Daarna volgen er vier variabelen (mqtt_server, mqttPort, mqttUser en mqttPassword) ter configuratie van de Mqtt client. Vervolgens zijn er drie variabelen (numChars, receivedChars en newData) die gebruikt worden voor de software seriële communicatie met de arduino. Daarna zijn er twee variabelen (lastReconnectAttempt en reconnectDelay) die dienen voor het herstellen van de verbinding met de Mqtt provider. Als voorlaatste wordt de WiFi client gedeclareerd (espClient). De laatste declaratie (PubSubClient client) is strikt genomen het aanspreken van de constructor van PubSubClient om de instantie 'client' te initialiseren.

```
void callback(char* topic, byte* payload, unsigned int length);  
boolean reconnect();  
void recvWithEndMarker();  
void publishNewData();
```

We hebben ook hier gebruik gemaakt van forward declaration. Dit zijn de methodes die geïmplementeerd zijn in de ESP.

```
void setup() {  
    delay(1500);  
    WiFi.begin(ssid, password);  
    Serial.begin(9600);  
    client.setServer(mqtt_server, mqttPort);  
    client.setCallback(callback);  
}
```

Onze setup() methode zorgt voor de initialisatie van de seriële verbinding, de WiFi en de Mqtt client. De setCallback methode bepaalt welke functie opgeroepen zal worden bij het ontvangen van een Mqtt bericht. Er is gebruik gemaakt van delay() om de Arduino zelf de kans te geven om volledig opgestart te zijn alvorens er berichten ontvangen kunnen worden door de ESP. Dit is de enige keer dat we delay() gebruiken en hier wordt het enkel en alleen bij het opstarten aangeroepen. Het zorgt dus niet voor enige blokkering tijdens het uitvoeren van de main loop(). We gaan ervanuit dat dit aanvaardbaar is. In principe zou een loop met een verificatie op millis() ook mogelijk zijn, maar het resultaat zou hetzelfde zijn, namelijk een blokkering van de enige cpu thread. Dat het gebruik van delay() veel leesbaarder is heeft dan ook de doorslag gegeven om het in dit specifiek geval toch te gebruiken.

```
void loop() {  
    if(WiFi.status() == WL_CONNECTED){
```

```

        if (!client.connected()) {
            unsigned long now = millis();
            if (now - lastReconnectAttempt > reconnectDelay) {
                lastReconnectAttempt = now;
                // Attempt to reconnect
                if (reconnect()) {
                    lastReconnectAttempt = 0;
                }
            }
        }
        else {
            client.loop();
            recvWithEndMarker();
            publishNewData();
        }
    }
}

```

Als de ESP niet verbonden is dan doet deze in feite helemaal niets. Pas bij het bestaan van een draadloze verbinding zal er nagaan worden dat de Mqtt verbinding weldegelijk geconnecteerd is. Zoniet zal er om de 5 seconden (waarde van reconnectDelay) opnieuw geprobeerd worden. Indien de verbinding bestaat wordt de standaard client.loop() methode van de Mqtt library aangeroepen en wordt binnenkomende data op de seriële verbinding (afkomstig van de Arduino) uitgelezen door recvWithEndMarker() en indien nodig doorgestuurd naar de Mqtt provider door publishNewData(). Ook wel interessant is dat er voor de ESP geen watchdog moet ingesteld worden. De ESP8266 heeft een hardware watchdog die de module automatisch reset indien de loop() functie niet snel genoeg doorlopen wordt¹.

```

void callback(char* topic, byte* payload, unsigned int length) {
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();
}

```

Deze methode wordt aangeroepen zodra er een bericht wordt ontvangen op de Mqtt queue. Het stuurt alle data eenvoudigweg door naar de Arduino, waar de daadwerkelijke parsing en erachter horende logica zal plaatsvinden.

¹ <https://techtutorialsx.com/2017/01/21/esp8266-watchdog-functions/>
<https://arduino-esp8266.readthedocs.io/en/latest/faq/a02-my-esp-crashes.html#what-is-the-cause-of-restart>

```

boolean reconnect() {
    if (client.connect("RoachClient", mqttUser, mqttPassword)) {
        client.subscribe("WebRoach"); // callback: mqtt bus ->
arduino
        client.publish("Roach", "Hello from Roach!");
    }
    return client.connected();
}

```

Dit is de reconnect procedure voor in het geval de verbinding verbroken werd. Er wordt opnieuw gesubscribed op de juiste topic en er wordt een Hello bericht gestuurd. Het resultaat van client.connected(), een boolean, wordt gereturned.

```

void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
}

```

Deze functie parst berichten komende van de Arduino. Eenmaal het bericht volledig ontvangen is (het einde van een bericht wordt aangegeven door '\0'), zal deze in publishNewData() in zijn volledigheid verstuurd worden naar de Mqtt provider. Dit wordt aangegeven door newData op true te zetten. Deze parse logica is eenvoudiger dan die van de Arduino, omdat het veel minder variatie is op wat voor data er zal binnenkomen.

```
void publishNewData() {  
    if (newData == true) {  
        client.publish("Roach", receivedChars); // publish: arduino  
-> mqtt bus  
        newData = false;  
    }  
}
```

Indien `recvWithEndMarker()` nieuwe data ontvangen heeft en hierbij uiteindelijk `newData` op `true` gezet heeft, zal deze methode het volledige bericht in een keer doorsturen naar de Mqtt provider.

Conclusie

Met onze R.O.A.C.H. hebben we een leuk toestel dat goed werkt. Er waren meerdere uitdagingen, zoals het voorzien van de H-bridge, de ESP8266 WiFi module en het stateless programmeren. We zijn zeer tevreden over het resultaat.