HOCHSCHULE HEILBRONN

Technical Documentation & Setup Guide

People Counter Prototype with Visualization using Grafana

262183 Innovation Lab (WS21/22) - Hochschule Heilbronn

Team LoRa Lösung
Lauren Walton, Philipp Senfft, Dennis Sommer, Rania Adam

1. Introduction

This Guide is designed to help understand the Software and Hardware architecture of our prototype that we designed and implemented as part of the Innovation Lab Subject at the Hochschule Heilbronn. It also includes a Setup Guide so that it is possible to reproduce and build upon this project.

2. Software Documentation

The Software is based on a Ubuntu Server hosted in the BW Cloud Network. The Server can be accessed directly via SSH, for this to be possible you will need to send your public key to an admin of the server so that they can add it to the authorized_keys file. The Server was initialized using an Image of Ubuntu version 20.4.0 and an initialization script was used to install Docker version 20.10.7.

Two containers are running with Docker - one for the Prometheus Time Series Database instance (running on the default port 9090) and one for the Grafana instance (running on the default port 3000). We wrote a small Spring Boot Application (port 8080) that can receive web requests and dynamically create custom metrics that are later pulled by Grafana from Prometheus to be displayed in the Dashboard(s). We used an additional Micrometer plugin for Spring Boot, as the newer versions of Spring Boot do not support Prometheus directly. Nginx is used to expose these applications outside their network and to configure a reverse proxy for the Spring Boot Application.

For the Documentation about these Software Components have a look at the following links:

Docker: https://docs.docker.com/

Prometheus: https://prometheus.io/docs/prometheus/latest/getting_started/

Grafana: https://grafana.com/docs/grafana/latest/

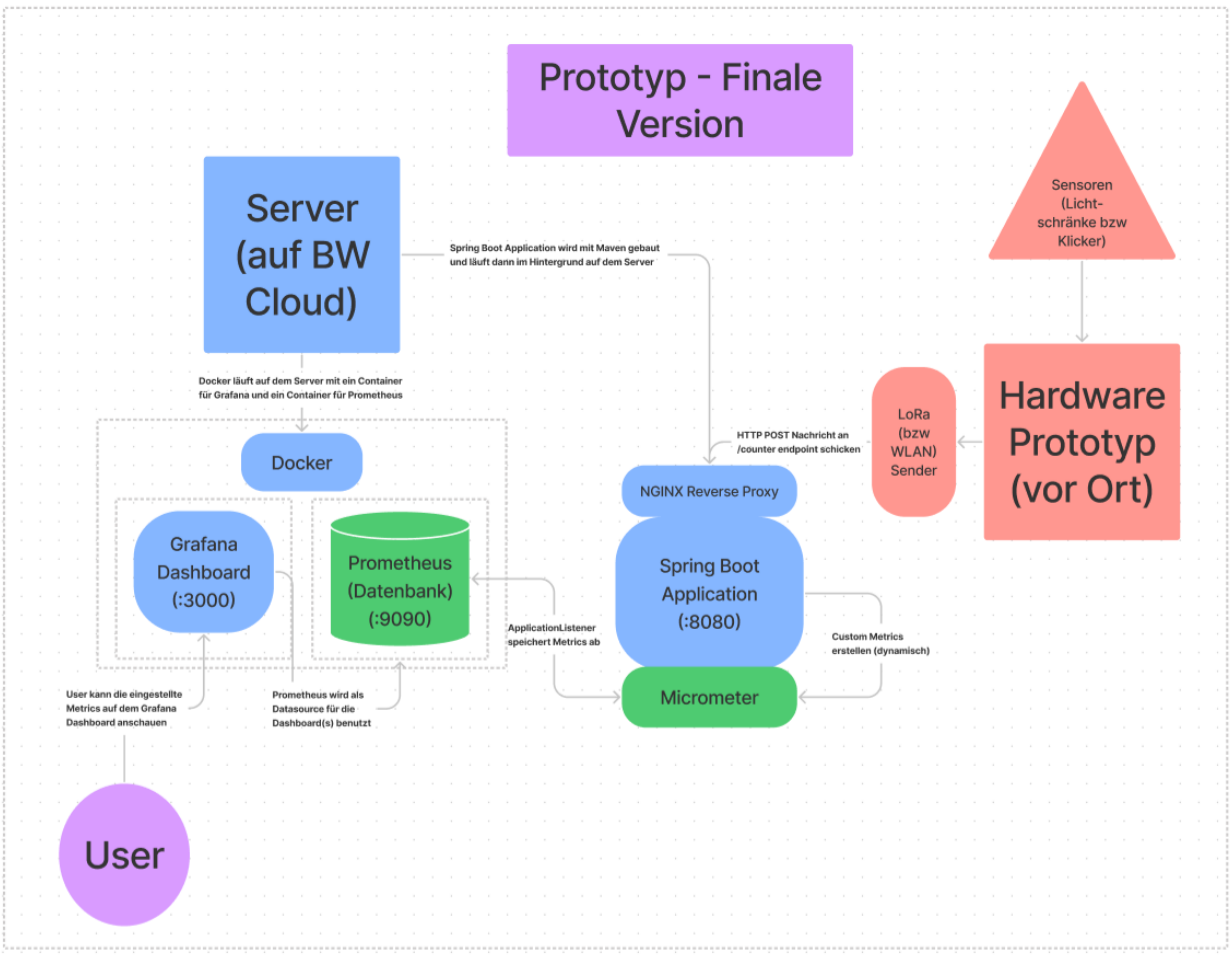
Spring Boot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

Micrometer: https://micrometer.io/docs

Nginx: https://nginx.org/en/docs/

The Hardware sends an HTTP Request to the Server - a POST request to the endpoint ‘/counter’, which then creates the custom metrics. Prometheus is configured to watch this port for metrics and grafana is configured to read out of prometheus as its datasource. The current Test server is running at: http://193.196.52.243 - both prometheus and grafana are open to external access.

2.1 Context diagram (final Version of the prototype)



2.2 Setup Guide(s)

Here we provide external resources with additions of our own to help you set up an identical Software Prototype.

Ubuntu Server:

To use BW Cloud like we did, following the instructions on creating an account and setting up a server here: https://www.bw-cloud.org/en/bwcloud_scope/use

During creation of the Instance, you can also add an initialization script. We used this function to pre-install Docker, but this can also be done once the Instance has loaded either via script or directly using the command line.

Docker:

We used the following script to install Docker (also prepares the instance for future installs):

```
#!/bin/bash

###    This file should be run in SUDO mode

### The script file needs to be executable, i.e.
#    chmod +x script.sh

#    Update package index
apt-get update

#    Install tools
apt-get install -y \
      apt-transport-https \
      ca-certificates \
      curl \
      software-properties-common

#    Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

#    Setup stable repo
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

#    Update package index (again)
apt-get update

#    Install latest version of Docker CE
apt-get install docker-ce -y

sudo apt install docker.io

#    Run Hello World
docker run hello-world
```

Prometheus:

The Prometheus Instance will need a custom yml file so that it can read metrics from the Spring Boot Server on port 8080. Create or copy the following yml file into the etc/prometheus folder on the server (create this folder if it does not exist already).

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
      - static_configs:
      - targets:
            # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries
scraped from this config.
  - job_name: "prometheus"

      # metrics_path defaults to '/metrics'
      # scheme defaults to 'http'.

      static_configs:
      - targets: ["localhost:9090"]

  - job_name: "spring"
      metrics_path: "/actuator/prometheus"
      scrape_interval: 5s
```

```
    static_configs:
    - targets: ["localhost:8080"]
```

This yml tells prometheus to pull metrics from the application on port 8080 every 5 seconds using the /actuator/prometheus endpoint, which will be set up by Micrometer.

The container can then be started with Docker using:

```
sudo docker run -d -p 9090:9090 -v
/etc/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
prom/prometheus
```

This may take some time as it will need to pull the prometheus image first. The endpoint with graphing functions won't be visible yet as we have not set up Nginx, but you can check that the container is running with:

```
sudo docker ps
```

Grafana:

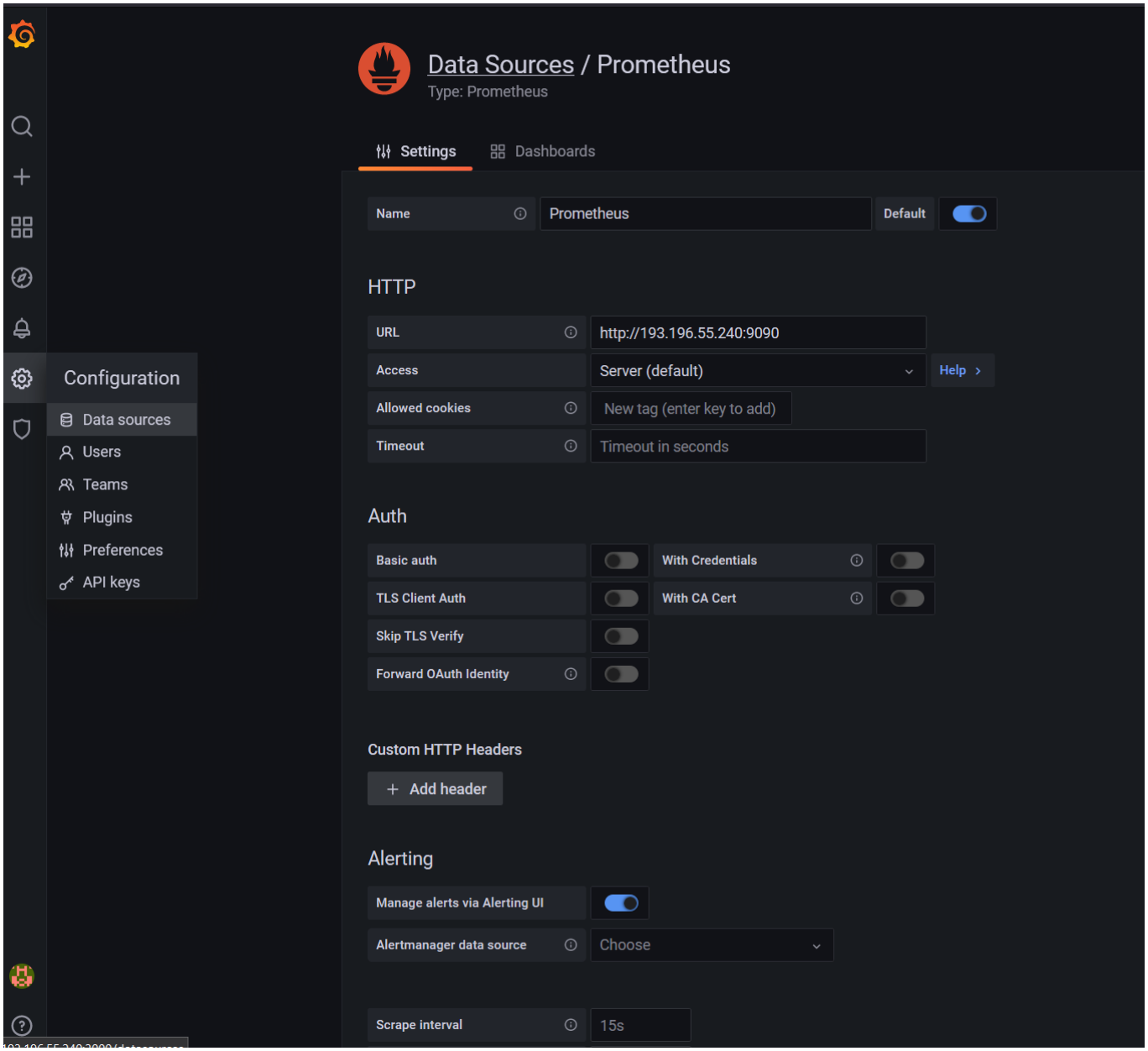
The installation of Grafana is similar, but since we configure the datasource after installation we do not need any special configuration files, we can just start the container with:

```
sudo docker run -d -p 3000:3000 --net=host grafana/grafana
```

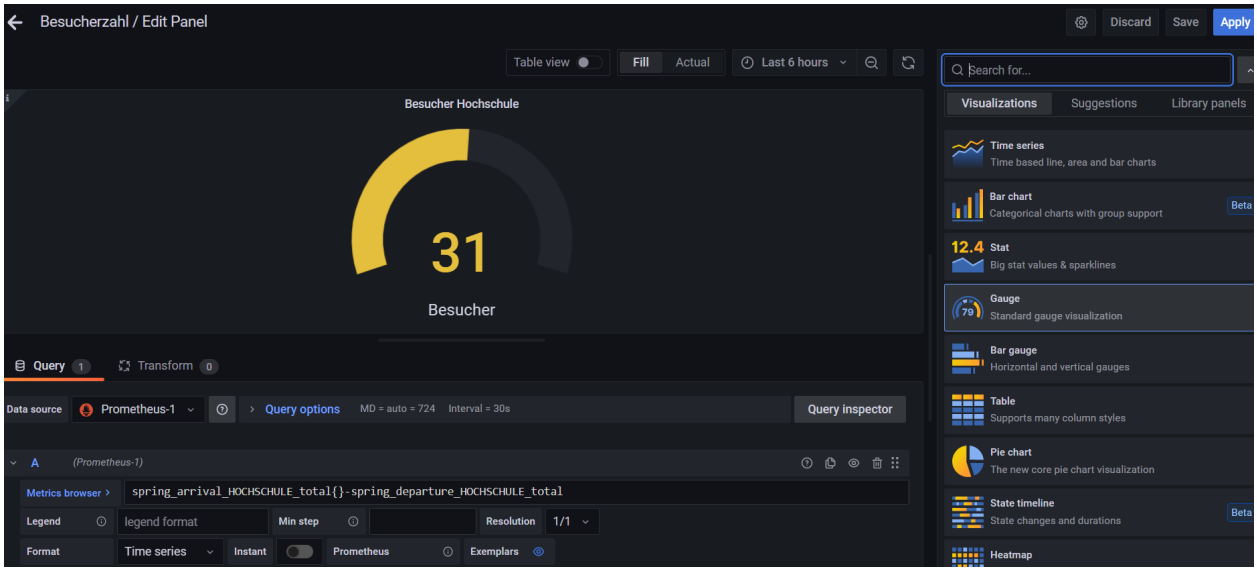
Again this may take some time, as the grafana image first needs to be pulled before it can be started. When you use docker ps now, you should see both containers running.

You need to configure Prometheus as the datasource for grafana before you can use the metrics for your dashboard. Once you can reach the endpoint at :3000 (you may need to finish installing nginx first), login with username admin, password Innows2122 - you can change this right after you login the first time if you'd like. On the left hand side, click on the cog and then on Data Sources (see image below).

You will be prompted to select the type of data source, click on Prometheus and then enter the correct IP and port of prometheus - this will be localhost by default, but this will not work in a Container environment. At the bottom of the page you can click on 'Save & Test' to check that the datasource is correctly configured.


Data Sources / Prometheus
Type: Prometheus
Settings
Dashboards
Name
Prometheus
Default
HTTP
URL
http://193.196.55.240:9090
Access
Server (default)
Help
Allowed cookies
New tag (enter key to add)
Timeout
Timeout in seconds
Auth
Basic auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
Custom HTTP Headers
Add header
Alerting
Manage alerts via Alerting UI
Alertmanager data source
Choose
Scrape interval
15s
Configuration
Data sources
Users
Teams
Plugins
Preferences
API keys

To create the individual diagrams, queries must be written and the individual diagrams selected.



To determine the number of visitors, the number of visitors who left the event is subtracted from the number of visitors who arrived. It is also possible to display only the number of the last 5 hours or 24 hours.

```
spring_arrival_SPIELEMESSE_total{}-spring_departure_SPIELEMESSE_total{}
```

Query for visitors in a Event

```
increase(spring_arrival_MARKTPLATZ_total{}[5h])
```

Query for visitors in the last 5 hours in a Event

Spring Boot/Micrometer:

Install the Java Virtual Machine so the application can run with:

```
sudo apt install openjdk-8-jre-headless
```

You can check it has installed correctly with:

```
sudo java -version
```

You will need to copy over the application .jar to the server. Here is a link to the public repository where we have our spring boot application: https://github.com/Liruilos/lora

Since the project is using Maven, you can use Maven's 'package' function to create the .jar file. Check out the pom.xml to see which dependencies are used - we included the web starter kit and micrometer for the custom metrics. You can use scp or similar to move the .jar to the server and then when in the same folder as the .jar run:

```
sudo nohup java -jar lora-0.0.1-SNAPSHOT.jar &
```

This will run the application in the background without showing you the output in the console.

NGINX:

The final step to make everything accessible externally is to install Nginx and set up a reverse proxy for the spring boot application. Install nginx with

```
sudo apt-get install nginx
```

And make sure that the built in firewall allows web traffic with:

```
sudo ufw allow http
sudo ufw allow https
```

For the reverse proxy, follow the instructions as illustrated in this guide: https://www.linode.com/docs/guides/how-to-deploy-spring-boot-applications-nginx-ubuntu-16-04/

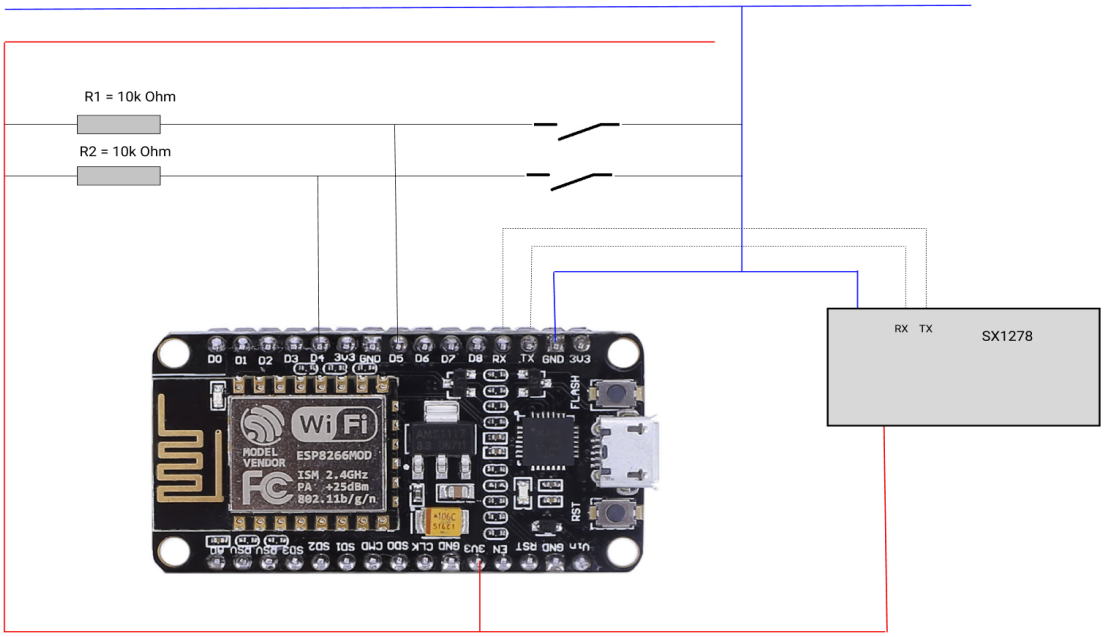
Afterwards the prometheus instance will be accessible externally via: yourIP:9090, grafana at yourIP:3000 and the endpoint to send counter updates at yourIP/counter (the port 8080 is not required due to the reverse proxy).

3. Hardware Documentation

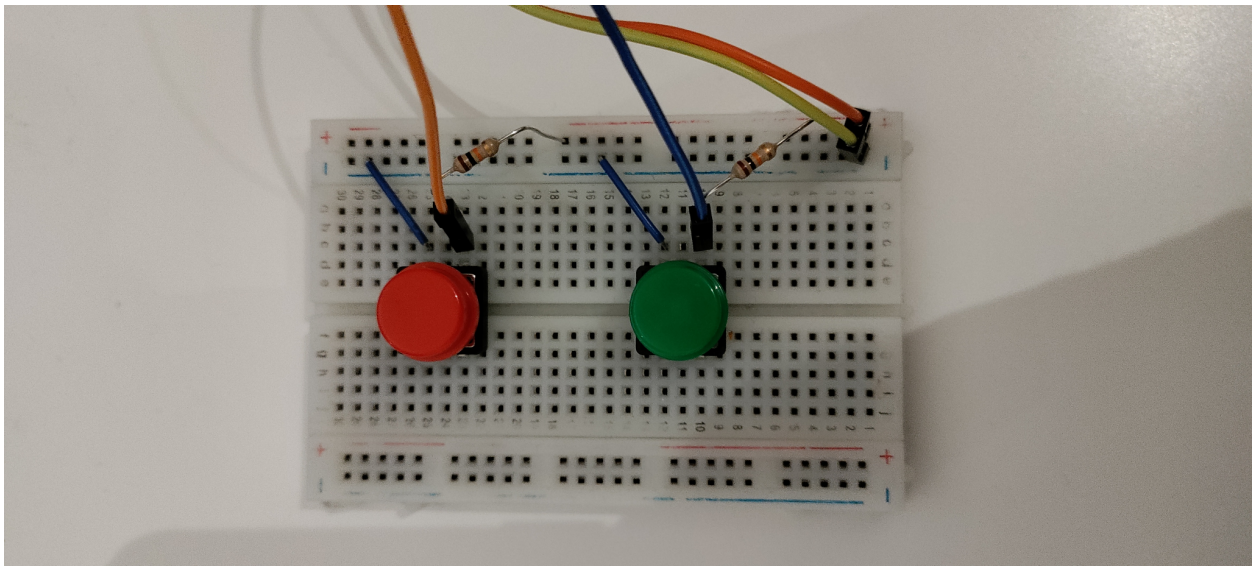
The actual hardware for the prototype is based on an Arduino ESP8266 Chip. Documentation for this chip can be found here: https://arduino-esp8266.readthedocs.io/en/latest/
There are two Buttons. The green Button is for arriving people, the red one for leaving people.

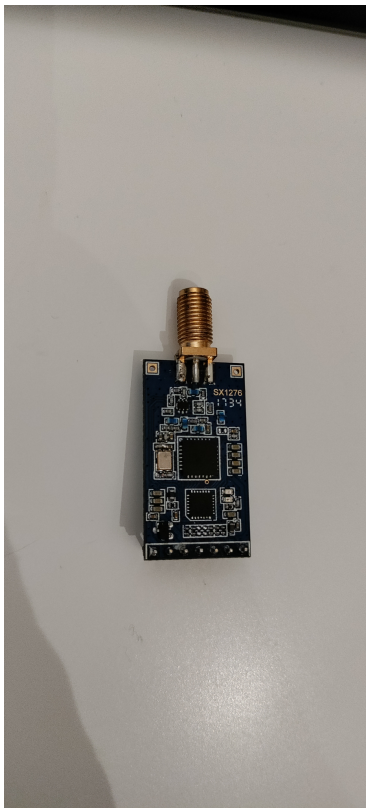
To use the prototype at an event, three variables in the code must first be adjusted and the program must be reloaded onto the µController. The variable "deviceId" must be chosen uniquely and assigned to an entrance or exit of an event. The variable ssid and password must be selected for the WLAN access in range.

3.1 Hardware Diagram



3.2 Images of the built Prototype





3.3 Arduino Code

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

//device ID
String deviceId = "HOCHSCHULE";

//Pins for the red and green button (red for leaving person, green for arrived person)
const byte interruptPin_G = 4;
const byte interruptPin_R = 5;

//login data for the Wifi network
const char* ssid     = "PS-NOTEBOOK 6647";
const char* password = "e015P]20";

//Your Domain name with URL path or IP address with path
const char* serverName = "http://193.196.52.243/counter";

//Variables for set a delay without stopping the program
unsigned int current_millis = 0;
unsigned int timeout = 5000;

//is there still data to send per WiFi
boolean data_to_send = false;

//counter to in and decrease the people of the event
int counter = 0;

void setup() {
  Serial.begin(9600);

  //Start WiFi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  //Interrupts for the buttons
  attachInterrupt(interruptPin_G, interrupt_G, FALLING);
  attachInterrupt(interruptPin_R, interrupt_R, FALLING);
}
```

```
ICACHE_RAM_ATTR void interrupt_G(){
  Serial.println("interrupt_G");
  counter++;
  data_to_send = true;
}

ICACHE_RAM_ATTR void interrupt_R(){
  Serial.write(WiFi.status());
  Serial.println("interrupt_R");
  counter--;
  data_to_send = true;
}



// send data over WiFi
void send_data_to_server(){
  if(WiFi.status()== WL_CONNECTED){
      WiFiClient client;
      HTTPClient http;

      http.begin(client, serverName);
      http.addHeader("Content-Type", "application/json");

      //build json string for http request
      String sCounter = String(counter);
      String httpRequestData1 = ("{\"deviceId\":\"" + deviceId);
      String httpRequestData2 = ("\",\"counter\":\"" + sCounter);
      String httpRequestData3 = ("\"}");
      String httpRequestData = (httpRequestData1 + httpRequestData2 + httpRequestData3);

      int httpResponseCode = http.POST(httpRequestData);
      String payload = http.getString();

      Serial.println(httpResponseCode);
      Serial.println(payload);

      //reset counter and data to send
      counter = 0;
      data_to_send = false;

      // Free resources
      http.end();
      }

      //Error when data could not be send
      else {
        Serial.println("WiFi Disconnected");
    }
}
```