Webhuesli

# Teamaufteilung

* Käser und Jascha
* Aufgabenbereich:
  + C Software („Server-Software“) schreiben welche das Interface zwischen Webhüsli-Hardware und dem Socket bereitstellt.
    - Linux API ansaugen
    - Regelung programmieren
    - Socket öffnen (Server initialisiert den Socket)
* Stoller und Bärtschi
* Aufgabenbereich:
  + Website in HTML und CSS programmieren, welche auf dem Embedded Board im Webhuesli läuft und per Socket die Client-Anfragen/Commands an die Server-Software weiterleitet.
    - Website (HTML und CSS)
    - Funktionalität in JavaScript
    - Socket connecten (JavaScript)

# Projektablauf

Als erster Arbeitsschritt soll möglichst schnell eine Socket Verbindung aufgebaut werden.

## Website

Webseite mit Textbox und Button, welche dann einen Key-Value-Paar über den Socket schickt. Dann noch Textbox mit String, der empfangen wird.

Parallel dazu wird die Website mit Bilder und Canvas populiert.

## Server-Software

C Programm, welches den Socket bereitstellt und verbindet. UI mit Printf und scanf. 🡪 JSON parsing!

parallel dazu wird versucht, die APIs des Webhuesli anzusteuert und verschiedene Interfacefunktionen für modulare saubere programmierung zu schreiben.

# Zeitlan

* XLS File

# Alarmanlagenlogik

Die Logik der Alarmanlage befindet sich beim Client (JavaScript). Die Website beim Client empfängt einzig und allein, ob die Lichtschranke unterbrochen wurde (Burglar = 1). Die Logik für das Auswerten von aktivem oder passivem Alarm und das Ausführen des Alarm-Sounds passiert alles auf dem Client. Dem Server wird nichts zurückgeschickt, weder die Information über aktivem- oder passivem Alarm noch das Auslösen des Sounds.

# JSON Protokoll

* Es wird immer nur EIN Key-Value Paar geschickt!
* Die gemessene Temperatur ist in 1° aufgelöst. Also wird sich diese Temperatur nicht all zu oft ändern. Der Server soll die aktuelle Temperatur immer in eine temp\_old variable speichern und bei einer Änderung ein JSON Paket mit der neuen Temperatur verschicken. So reduzieren wir den Traffic.
* Bei einem Connect sendet der Server ALLE unten beschriebenen Werte an den Client, damit dieser seine Bedienungselemente auf den aktuellen Zustand des Webhueslis initialisieren kann. Dabei soll die Soll-Temperatur gleich der Ist-Temperatur sein.

### Client sends:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Element** | **Key** | **Value min** | **Value max** | **Default** | **Example** |
| *Fernseher* | « TV » | OFF | ON | 0 | «TV:ON» |
| *Stehlampe* | « Lampe » | 0 | 100 | 0 | «Lampe:50» |
| *Kronleuchter* | « Leuchter » | 0 | 100 | 0 | «Leuchter:70» |
| *Soll-Temperatur* | « TempSoll » | 15 | 30 | 20 | «TempSoll:22» |

### Server sends :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Element** | **Key** | **Value min** | **Value max** | **Default** | **Example** |
| *Ist-Temperatur* | « TempIst » | 15 | 30 | 20 | «TempIst:21» |
| *Heizung* | « Heizung » | 0 | 100 | 0 | «Heizung:80» |
| *Lichtschranke* | «Burglar » | 0 | 1 | 0 | «Burglar:1» |

Optional kann der Server beim Startup alle „Keys“ des Clients (also den Default-Zustand des Hauses) dem Client schicken um so auf der Website die aktuelle Darstellung des Hauses ohne vorheriges „Reset“ zu übernehmen. Wird dies nicht gemacht, werden einfach alle Keys mit 0 initialisiert (ausser TempSoll mit 20).

# Struktogramm Server

