Abschlussprojekt
Automatischer Cocktail-Mischer

Tobias Hopp &
Nick Thiele

07.11.2022 – 06.02.2023

—

Fachpraxis

—

Herr Hornbogen

Grundidee

Unsere Grundidee war ein Intelligenter Getränke/Cocktail-Mischer, der mit einer integrierten Datenbank Drinks vorschlägt, die mit den angeschlossenen Getränken möglich sind. Hierfür werden beliebig viele Behälter mit Getränken an das Gerät angeschlossen und im System eingepflegt.

Inhaltsverzeichnis

Inhaltsverzeichnis

Einleitung

In dieser Dokumentation wird der Ablauf des Projekts „iTender“ von Tobias Hopp und Nick Thiele beschrieben und veranschaulicht.

Das Projekt entstand im Unterricht von Herrn Hornbogen für das Fach Fachpraxis und wurde innerhalb der Projektstunden sowie zuhause entwickelt.

Unser Ziel war ein intelligenter Getränke bzw. Cocktail-Mischer, welcher für die Endbenutzer einfach verwendbar ist, und schnell leckere Cocktails mischen kann.

Konzept Version 1

Das erste Konzept haben Tobias Hopp und Nick Thiele zusammen entwickelt.

Es basierte auf einem Kasten, welcher hinten 4 Behälter haben wird und generell ein Getränk ausgeben kann.

Uns kam die Idee auf, ein Display zu verwenden. Zu diesem Zeitpunkt waren wir allerdings noch sehr weit weg von der Umsetzung und konnten erst von einfachen Knöpfen mit einem kleinen zweizeiligen LCD-Display träumen.
Das System sollte auf einem Raspberry Pi mit ein paar Knöpfen und ein bis zwei Pumpen basieren.

Die Behälter mit jeweils 750ml Volumen werden hängend von einem fest fixierten Deckel gehalten.

Die Deckel sind alle ausgestattet mit einem Schlauch und einem Ultraschall-Sensor, der den Inhalt misst. Die Flaschen werden von hinten in das Gehäuse gedreht. Die Schläuche führen zu Pumpen, welche Automatisch die gewollte Menge aus der Flasche pumpen können.

Gesteuert werden sie von einem Raspberry Pi welcher als Prozessoreinheit dient. Über ein Display in der Front des Gehäuses kann der Benutzer das gewünschte Getränk auswählen. Ebenfalls lassen sich hierrüber die angeschlossenen Getränke einpflegen.

Das System filtert dann automatisch alle passenden Drinks, welche mit den angeschlossenen Getränken möglich sind.

Zusätzliche Ideen

- LED-Stripes welche für eine „Ambientebeleuchtung“ sorgen
- Extraschlauch für weitere außenstehende Getränke (eventuell mit Bierfass Adapter)
- Kühlung der Container mittels Peltierelement und Lüftern.

3D Modell V1

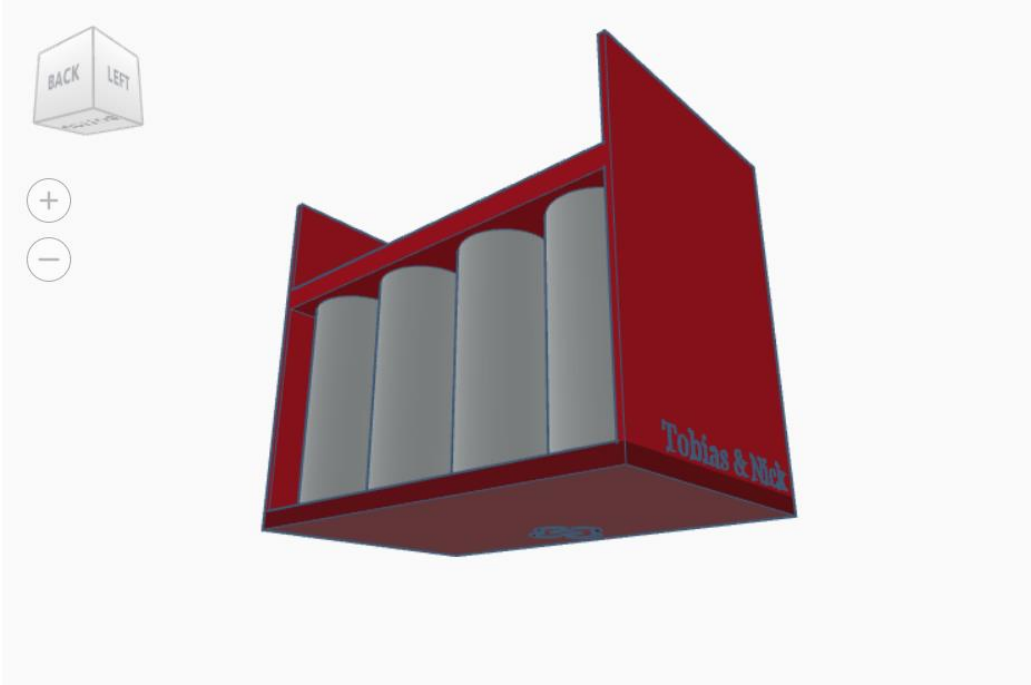


Abbildung 1 3D Modell Hinten

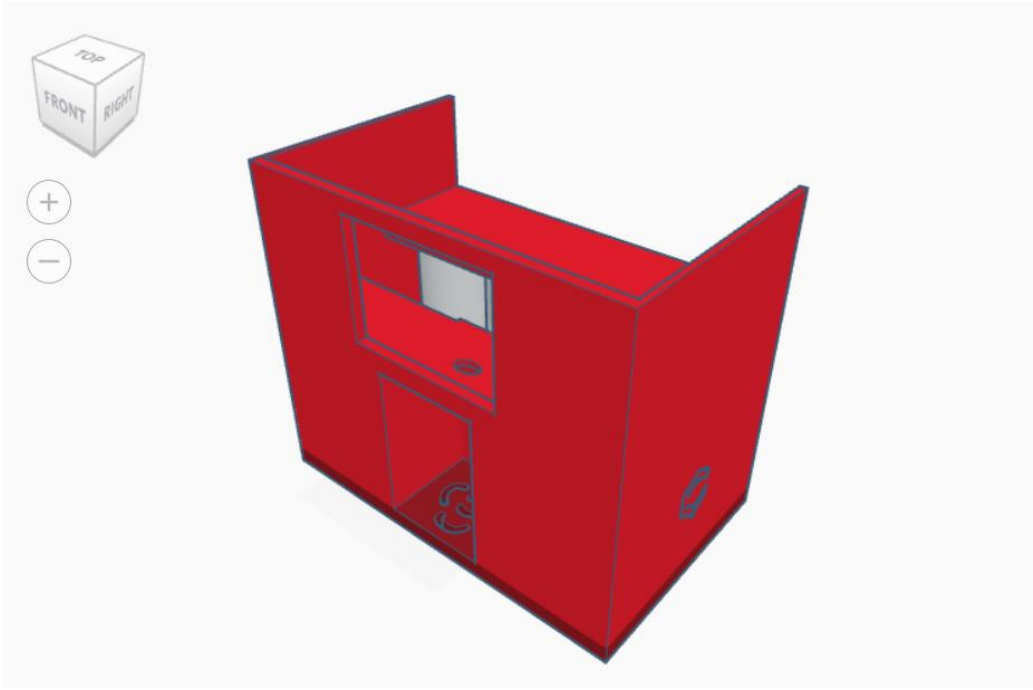


Abbildung 2 3D Modell Vorne

Konzept Version 2

Dieses Update hat weit über 100 Stunden Programmierarbeit in Anspruch genommen und beinhaltet zahlreiche Neuerungen und Optimierungen.

Zunächst wurden einige Features entfernt und verschoben, während gleichzeitig neue hinzugefügt wurden. Eine wichtige Änderung betrifft das Volumen, das nun nur noch in der Zutaten-Auswahl zu finden ist. Bei Auswahl wird dann das eingestellte Volumen als 100% interpretiert und der Wägesensor stellt sich entsprechend ein.

Eine weitere wichtige Änderung betrifft die Ultraschall-Sensoren, die entfernt wurden, da sie nicht wartbar sind und einige Fehler aufweisen. Dazu gehören unter anderem eine ungenaue Messung, keine genauen Ergebnisse ohne Kalibrierung und kein schneller Behälterwechsel. Diese Änderungen haben dazu beigetragen, das Produkt zu optimieren und die Benutzererfahrung zu verbessern.

Ein neues Feature namens Arduino-Proxy wurde hinzugefügt. Dieses Feature ermöglicht es, einen Arduino einzusetzen, der Befehle vom Raspberry Pi aufnimmt und so bis zu 100 Pumpen oder 50 Sensoren steuern kann. Die Kommunikation erfolgt über die serielle Schnittstelle des Arduinos und des USB-Ports des Raspberry Pi. Der Arduino wird beim Installieren sowie Updaten automatisch vom iTender programmiert. Im Setup kann nun auch ein bestimmter Pin bei jedem Behälter ausgewählt werden.

Eine weitere wichtige Änderung betrifft die Erweiterung des Systems auf bis zu 50 Behälter. Das System wurde optimiert, um nun bis zu 50 Behälter zu ermöglichen. Diese Erweiterung ermöglicht es, das Produkt noch flexibler und leistungsfähiger zu machen.

Ein neues iTender-Logo, entworfen von Nick Thiele, wurde hinzugefügt. Es gibt dem Produkt eine frische Optik und unterstreicht die Qualität sowie die Professionalität des Produkts.

In Version 2 wurde die Software bis zu 20% schneller gemacht. Diese Optimierung ermöglicht es, das Produkt noch schneller und effizienter zu nutzen.

Features wie Hotspot wurden vorübergehend pausiert, um sich auf die wesentlichen Features zu konzentrieren.

Ein weiteres wichtiges Feature, das geplant ist, ist ein Säuberungsprogramm. Dieses Feature wird dazu beitragen, das Produkt längerfristig zu erhalten und die Lebensdauer des Produkts zu verlängern.

Im Setup wurde das Einmessen komplett überarbeitet. Wo es vorher kompliziert war, ist es jetzt ein einfacher Tastenklick und nimmt nur ungefähr ein bis zwei Minuten in Anspruch. Diese Änderung erleichtert die Verwendung des Produkts und ermöglicht es, Zeit zu sparen.

Falls keine Sensorik (wie z.B. die Wägezelle) für den Behälter (Container) zur Verfügung steht, wird eine manuelle Zählung durchgeführt, um einen ungefähren Füllstand zu erhalten. Diese Änderung ermöglicht es, das Produkt auch in Situationen zu verwenden, in denen keine Sensorik vorhanden ist.

Insgesamt hat das Versionsupdate von v1 auf v2 dazu beigetragen, das iTender-Produkt zu optimieren und die Benutzererfahrung zu verbessern. Diese Neuerungen und Optimierungen ermöglichen es, das Produkt noch flexibler, leistungsfähiger und benutzerfreundlicher zu machen.

3D-Modell V2

Version 2 ist die verbesserte erste Version. Wir haben kein komplett neues Modell entworfen, sondern sind mit unserer ersten Version sehr zufrieden gewesen.
Da nun aber das von Herrn Hornbogen bestellte Display angekommen ist, war es uns möglich genaue Maße dessen zu erfassen und dementsprechend das 3D-Modell anzupassen.

Außerdem haben wir bemerkt, dass, das Display einen sehr kleinen Sicht-Winkel hat. Somit mussten wir eine leichte schräge in das Modell einbringen, um dem Benutzer das Display bestmöglich anzuzeigen.

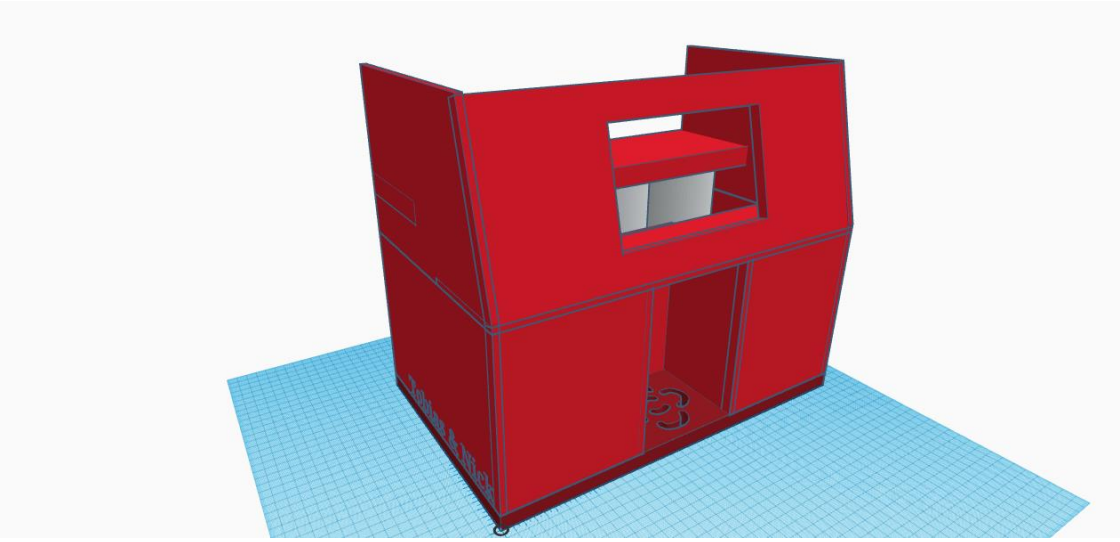


Abbildung 3 3D Modell V2 Vorne

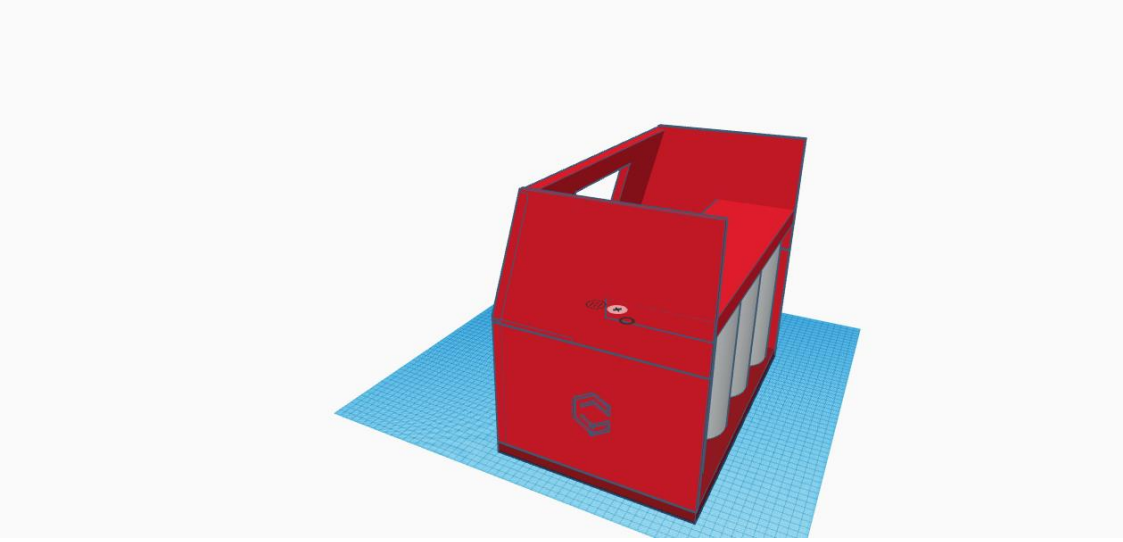
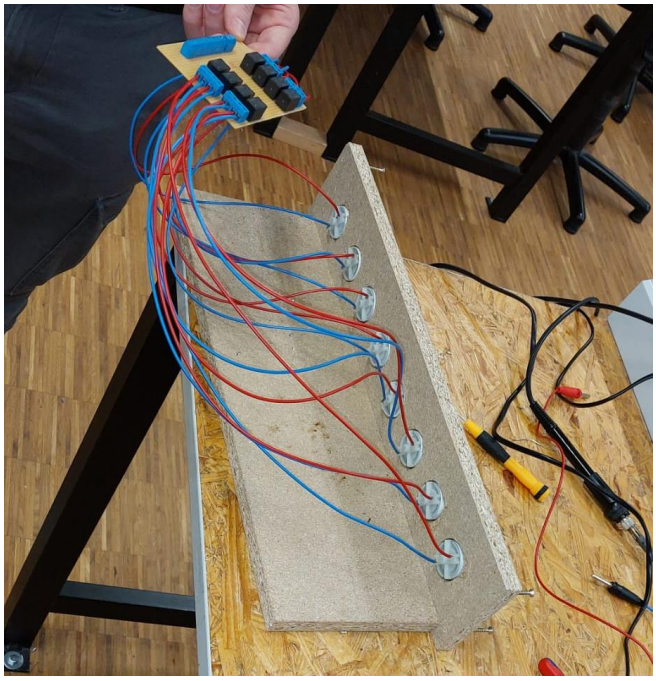
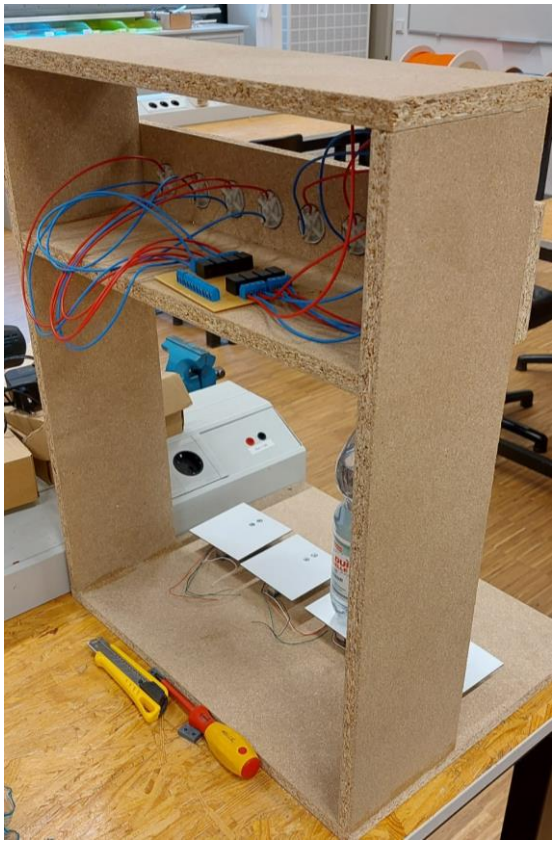
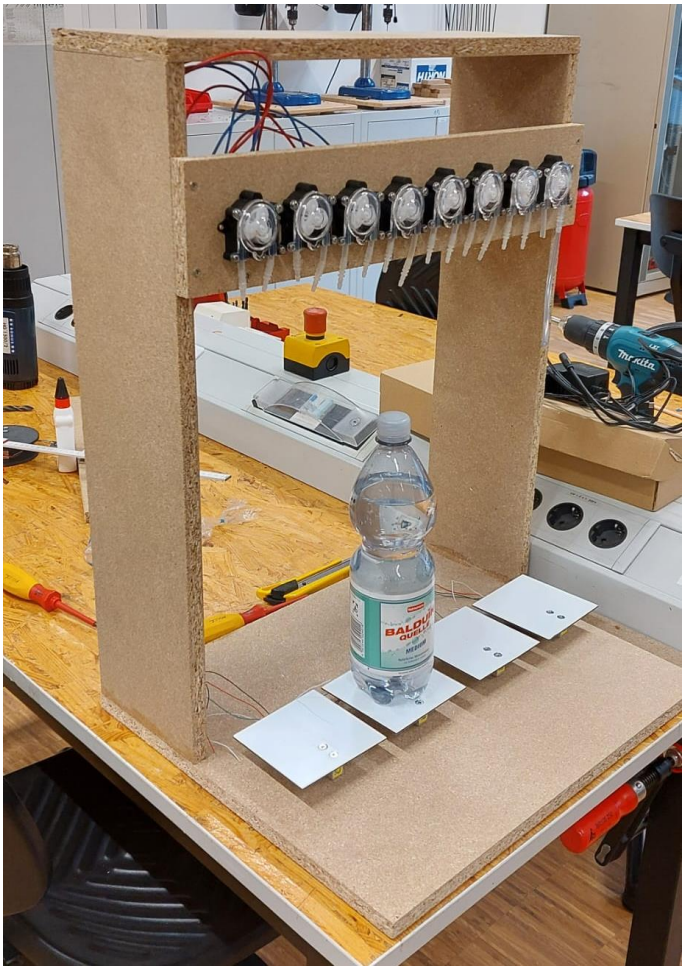


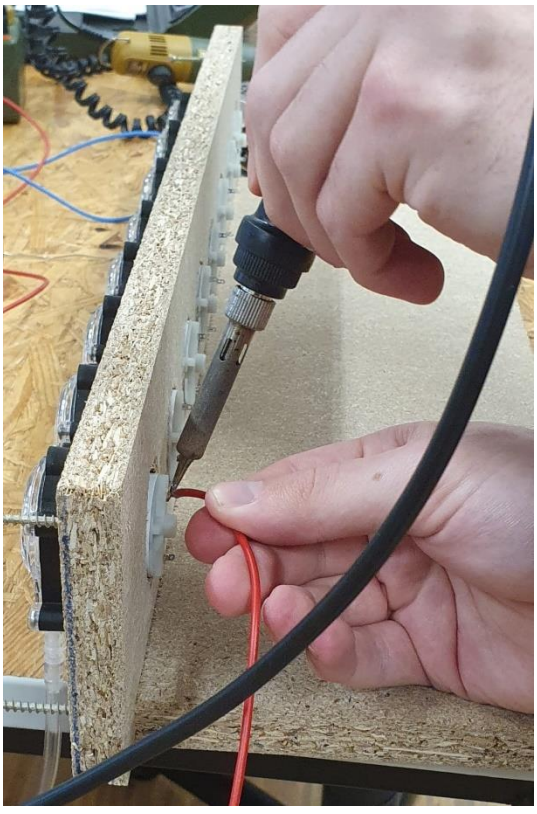
Abbildung 4 3D Modell V2 Seite

Zusammenbau und Fortschritt









Die Programmierung

Die Software wurde von Tobias Hopp entwickelt.

Programmiersprache & Bibliotheken

Als Programmiersprache haben wir TypeScript verwendet.
TypeScript ist eine Abwandlung von JavaScript, welche aber strikte Typen hat.
Typen sind in Programmiersprachen Eigenschaften von Variablen. Eine Variable kann eine Zeichenkette (sogenannten String) enthalten, aber auch eine Nummer (integer) sein.
Wenn Typengleichheit besteht, kann eine Variable (beispielsweise x) nur einmal definiert werden und behält dann den Typ bei. Wenn man beispielsweise x = 5 beschreibt, dann kann x nicht mehr eine Zeichenkette oder was anderes werden.
Typensicherheit ist gerade in größeren Projekten von Nöten, da so garantiert werden kann, welcher Entwickler wo und was bei einer sogenannten Methode (wie eine Funktion/Formel) zurückgibt.
Da JavaScript, von Haus aus keine Typensicherheit bietet, verabscheuen es einige.
TypeScript vereint nun die Typensicherheit mit der eigentlich sehr schönen Sprache JavaScript.

TypeScript hat den Vorteil, dass es sehr anpassbar und dynamisch ist. Es gibt bereits viele Bibliotheken, um LEDs und GPIO-Pins über den Raspberry Pi anzusteuern.
Außerdem lässt sich in wenigen Schritten ein kleiner Webserver einrichten, welcher nun das Hauptstück des iTenders ist.
Da es ebenso anpassbar ist, konnten wir den kleinen Webserver sehr erweitern und ihn genau zu unseren Zwecken Programmieren.
Die Sprache ist außerdem eine Kompilierung und Interpretierungssprache, ähnlich wie Java, weshalb sie sehr schnell ist und viele Asynchrone-Tasks (Aufgaben im Hintergrund) ausführen kann.

Genau das war uns wichtig, dass der Nutzer nicht darauf warten muss, dass das Programm fertig berechnet hat, sondern direkt weiter die Bedienung fortführen kann.

Für den Server selbst haben wir nicht nur natives Type- bzw. JavaScript verwendet, sondern auch Node.

NodeJs ist die Serverseitige Programmierung, des eigentlich für das Web entwickelten Javascripts.
Hier lassen sich die eben besagten Bibliotheken installieren und schnell zu einer Lösung programmieren.

Für Bibliotheken (Programmschnipsel anderer Programmierer) benutzen wir den Package Manager Yarn.
Über ihn lassen sich einfach Bibliotheken installieren, kompilieren und verwenden.
Für das Kompilieren selbst benutzen wir den TSC (Typescript Compiler), für das Verpacken für die Benutzeroberfläche Webpack.

Für die Datenbank, welche alles Speichert, haben wir uns für MongoDB entschieden.
MongoDB als No-SQL Datenbank arbeitet im Gegensatz zu Datenbanksystemen wie MySQL mit Dokumenten anstatt mit richtigen Tabellen.
Es bestehen keine festen Relationen zu anderen Dokumenten, es gibt auch keine wirklichen Tabellen.
Jegliche Datensätze sind dynamisch, das heißt, es besteht keine feste Datenstruktur.
Das klingt erstmal schlecht, ist aber mit dem richtigen Programm hilfreich.

Ein MongoDB Server verwaltet mehrere logische Datenbanken, die wiederum einen oder mehrere logische Namensräume enthalten, die sogenannten Collections. In einer Collection werden die einzelnen Datensätze, Dokumente genannt, verwaltet.

Durch die Benutzung von BSON als Dokument, lässt sich ohne großen Aufwand ein ganzes selbst erstelltes Paket in der Datenbank abspeichern.
MongoDB ist durch die Einfachheit auch deutlich schneller und ermöglicht es größere Datensätze mehr im Programmierkontext zu erfassen.
Mit der Bibliothek Mongoose ist es auch möglich das einzige Problem der Typensicherheit in den Griff zu bekommen.
So ist MongoDB nun schnell, Typensicher und bereit für große Mengen an Daten.

Versionen:

- Yarn: 1.22.19
- NodeJS: 18.9.1
- TypeScript: 4.8.4
- WebPack: 5.74.0
- Mongodb/Mongoose: 5.11.97

Aufbau

Das Programm des iTenders ist getrennt in 3 Teile:

- iTender Basis
  - Die Basis besteht aus der Kommunikation zwischen den Pumpen, LEDs und jeglicher Hardware.
  - Außerdem gibt es Timer, automatische Events, Prüfung und Aktualisierung von Getränken etc.
  - Sie übernimmt z.B. das Starten vom Getränke-Füllen, stoppen sowie Berechnen von den Zutaten für ein Getränk.
  - Dieser Teil arbeitet seh eng mit dem Websocket-Server zusammen, um so auf Events vom Endnutzer zu reagieren.
- iTender Webserver
  - Der Webserver ist einfach ein statischer Webserver welche Dateien "serviert", die dann von der Oberfläche geladen werden können.
  - Der Client-Browser oder iTender-Display lädt dann diese Seite, erstmal passiert dann noch nichts.
- iTender Websocket-Server
  - Der Websocket-Server dient zur eigentlichen Live-Kommunikation zwischen Client und Gerät.
  - Der Server und die Webseite (Endgerät), bauen eine Ende-zu-Ende Verbindung auf.
  - Die Kommunikation zwischen Client und Websocket-Server basiert auf JSON (Javascript-Objekt-Notation).
  - Da der Websocket in früheren Client-Versionen anfällig für Fehler beim Übertragen von nicht alphabetischen Zeichen war, wird der gesendete Inhalt noch in Base64-Kodiert.
  - Base64 dient dazu, die Daten binär zu kodieren, um sie auf der Gegenstelle wieder zu entkodieren.
  - Außerdem wird beim Übertragen eine Checksumme mitgegeben, um bei Fehlerhaften Paketen ein neues anzufragen.

Alle Kommunikationen zwischen dem WebSocket und der UI basieren auf JSON (Javascript Objekt Notation).
Dieses Datenformat ist sehr robust und ermöglicht das einfache Verpacken und Senden von Datentypen wie Zeichenketten, Wahr/Falsch Werten (Boolische Werte) sowie Nummern, Objekten und Listen.
Damit die Kommunikation ausfallssicher ist, wird das JSON in Base64 umgewandelt. Base64 ist eine Kodierung welche nur im gesamten wieder umgewandelt werden kann. Mit einzelnen Teilstücken dieser Enkodierung lässt sich nicht auslesen was dort einmal war.
Das ist zum einen eine kleine Sicherheit, aber auch unsere Möglichkeit zu prüfen, ob die Kommunikation gestört ist.
Sollte die Gegenstelle das jeweilige Paket nicht wieder in JSON kodieren können, fordert es das Paket erneut an.
Falls dann immer noch ein Problem besteht wird ein Alarm erstellt, welcher im Fehlerspeicher des iTenders gesichert wird.
Dort kann er vom Support-Team ausgelesen werden bzw. behandelt werden.

Da JSON eine Objekt Notation ist, müssen wir uns hier noch auf ein gemeinsames Protokoll einigen.
Unser Protokoll ist ein selbst gewähltes **Event** -> *Data* Prinzip.
Jedes Paket hat somit ein Event an das es geknüpft ist, die Gegenstelle weiß dann damit richtig umzugehen.
Beispiele hierfür sind das Event STATUS oder REQUEST.
Ein STATUS hat wieder einzelne sub-Status, welche den Status des iTenders wiedergeben.
Ein Status ist beispielsweise READY (iTender ist bereit und wartet auf Benutzerinteraktion) oder DOWNLOADING (iTender lädt neue Getränke herunter).
Der Status ist derzeit noch oben links im Rohformat in der UI sichtbar, soll aber früher oder später anders dargestellt werden.
Eine REQUEST hingehen ist nicht wie die anderen "Protokolle" nur senden und der andere empfängt, sondern ein die Oberfläche kann hier gezielt Daten anfragen und verarbeiten.
Ein Beispiel hierfür ist die Konfiguration, welche im Setup-Menü angezeigt wird. Damit alle Felder bereits ausgefüllt sind, wie der Benutzer sie eben konfiguriert hat, wird eine REQUEST mit dem Typen "CONFIG" an den WebSocket (iTender Basis) gesendet. Er antwortet dann mit der Konfiguration und die UI kann die Felder ausfüllen.

Um alle Getränke, Zutaten, Jobs und Vorgänge zu speichern, hat der iTender eine Datenbank.
Bei der Datenbank haben wir uns für MongoDB entschieden.

Alle 10 Minuten sowie vor und nach einem Job (Auftrag fürs Mischen) wird ein sogenannter Health-Check durchgeführt.
In ihm wird überprüft, ob ein vorheriger Job fehlgeschlagen ist und vielleicht feststeckt.

Für die Aktualisierung der Getränke aus der Cloud wird auch JSON benutzt.
Hier wird eine Abfrage an den Cloud-Server gestellt, falls eine Internetverbindung gefunden wurde.
Der Cloudserver antwortet hier, falls erfolgreich, mit einer Liste aller Zutaten und Getränke.
Der iTender geht dann die lokalen Getränke und Zutaten durch und schaut, ob er den Eintrag auch auf der Server-Version finden kann. Falls das nicht der Fall sein sollte, wird das Element aus dem lokalen Speicher entfernt.

Danach geht die Applikation die Cloud-Elemente durch. Jeweils wird geprüft, ob das Element bereits existiert und Änderungen vorliegen.
Für das Prüfen von Änderungen am Thumbnail (Vorschaubild des Getränks) sorgt eine Prüfsumme (Checksum).
Wir benutzen hier den HASH-256 Algorithmus, welcher einen eindeutigen Prüfschlüssel für einen bestimmten Inhalt erstellt. Passt dieser nicht, fordert der iTender das Thumbnail erneut von der Cloud an und speichert dieses lokal.

Da wir andere Schriftarten verwenden, müssen diese auch irgendwie geladen werden.
Anfangs haben wir die Schriftarten über den von Tobias H. bereitgestellten Font-Server geladen.
Da im späteren Anwendungsfall aber nicht immer eine Internetverbindung bereitsteht, können wir keine Schriftarten von Cloud-Services laden. Somit sind alle Schriftarten nun lokal heruntergeladen und werden auch dementsprechend lokal geladen.

Damit es später auch möglich ist, den iTender von einem anderen Gerät aus zu steuern, ist der Webserver auch über andere Geräte im Netzwerk erreichbar.
Das lässt sich im Setup anfangs über die "Remote-Zugriff" Option erlauben.
Falls aktiviert kann einfach auf die IP-Adresse des Raspberry Pis zugegriffen werden.
Die Verbindung wird dann allerdings der Haupt-Instanz getrennt.
Diese Vorkehrung haben wir getroffen, um jegliche Fehler mit Doppel-Events zu vermeiden.
Es wäre theoretisch möglich alles auf mehreren Endgeräten anzuzeigen, ist aber nicht das, was wir haben wollen.

Für LEDs nutzen wir eine Bibliothek aus NPM (Node Package Manager), welche das "Schreiben" auf die LEDs sehr einfach macht. Hier wird die Farbe je nach Status des iTenders angepasst. Im Ruhezustand (READY) ist sie auf der Ambiente-Farbe, welche man ebenfalls im Setup festlegen kann.

Im folgendem ist der Code für die Arduino-Programmierung zu sehen.

```
#include <ArduinoJson.h>
#include "HX711.h"
^
```

Dieser Teil importiert erstmal die Bibliotheken ArduinoJson und HX711.
ArduinoJson wird für den Austausch von Anfragen zwischen dem Arduino und dem

Raspberry Pi über die Serielle-Verbindung genutzt.
HX711 ist die Bibliothek für die Wägezelle, welche verwendet wird, um das Gewicht in Gramm auszurechnen.

```
// Define the size of the JSON buffer
#define JSON_BUFFER_SIZE 256

// Create a JSON object for incoming messages
StaticJsonDocument<JSON_BUFFER_SIZE> incomingJson;

// Create a JSON object for outgoing messages
DynamicJsonDocument outgoingJson(JSON_BUFFER_SIZE);
```

Hier wird die Buffer-Größe definiert, welche angibt wie groß das JsonDocument sein kann, welches zwischen Raspberry Pi kommuniziert.

Die Größe liegt hier bei 256-Bytes, welche derzeit ausreicht, um die Anfragen vom Pi zu verarbeiten und zu beantworten.

Danach erstellen wir ein StaticJsonDocument-Objekt incomingJson, welches das eingehende JSON verarbeiten soll. Hier geben wir mit <> unsere Buffer-Größe an.

Zuletzt gibt es das DynamicJsonDocument outgoingJson, welches das ausgehende Json enthält.

```
void setup() {
  // Initialize serial communication
  Serial.begin(9600);
}

void (*resetFunc)(void) = 0;  //declare reset function @ address 0
```

Hier befinden sich zwei Funktionen. Zum einen die void setup()-Methode, welche ausgeführt wird, sobald der Arduino startet, zum anderen die eher merkwürdig aussehende void (*resetFunc)...-Methode.

In der setup() Methode haben wir nur die Initialisierung der Seriellen Verbindung auf 9600-Baud. 9600 Baud gibt die Geschwindigkeit der Übertragung zwischen Pi und Arduino an. Wir testen erstmal mit 9600, können aber bis 115200 erweitern.

Die resetFunc dient zum Neustart des Arduinos per Software. Es gibt leider keine restart Funktion oder ähnliches, weshalb wir uns hier das Prinzip eines Sektor-

Schreibens zu Nutze machen und eine Funktion auf Adresse 0 im Speicher des Arduinos liegen.

Er stürzt quasi kontrolliert ab und startet neu.
Danach beginnen wir den Loop, welcher automatisch ausgeführt wird, der Arduino gestartet ist.

```
void loop() {
 if (Serial.available()) {
  // Read the incoming JSON message
  DeserializationError error = deserializeJson(incomingJson, Serial);
  if (error) {
   // Handle error
  } else {
   // Extract the "type" and "data" fields from the JSON object
   String id = incomingJson["id"];
   int type = incomingJson["type"];
   JsonVariant data = incomingJson["data"];

   // Create a nested object in the root object
   JsonObject outgoingData = outgoingJson.to<JsonObject>().createNestedObject("data");
```

Hier wird zuerst geprüft, ob Serielle Daten vorliegen.
Sollten Sie vorliegen "Deserialisieren" wir die JSON-Daten, welche gerade über die Serielle Verbindung gesendet werden.

Sollte es dabei einen Fehler geben, können wir diesen abfangen.
Derzeit würde die Anfrage vom Raspberry Pi -> Arduino einfach nach einer Sekunde eine Zeitüberschreitung erhalten, was im Prinzip dasselbe ist wie, Arduino hat nicht geantwortet oder etwas stimmt nicht.

Falls erfolgreich speichern wir uns die Anfrage-ID, den Anfrage-Typ sowie ein data-Objekt in Typ JsonVariant.

Die ID ist die Anfrage-ID, welche für unsere Antwort verwendet werden muss. Ansonsten kann Antwort und Anfrage nicht zugeordnet werden.

Der Type ist der Typ der Anfrage. Es gibt GET_PIN, SET_PIN, GET_SENSOR und RESTART.

Danach wird ein outgoingData Objekt erzeigt, welches dazu dient, die ausgehenden Daten zu speichern und zu setzen.

```
// Handle the message based on the "type" field
    switch (type) {
      case 1:
        // Handle SET_PIN message
        pinMode((int)data["pin"], OUTPUT);
        if (data["mode"] == "DIGITAL") {
          digitalWrite((int)data["pin"], data["value"]);
        } else {
          analogWrite((int)data["pin"], (data["value"] == 255) ? HIGH : LOW);
        }
        break;
      case 2:
        // Handle GET_VAL message
        pinMode((int)data["pin"], INPUT);
        int val;
        if (data["mode"] == "DIGITAL") {
          val = digitalRead((int)data["pin"]);
        } else {
          val = analogRead((int)data["pin"]);
        }
        break;
      case 3:
        // Handle GET_SENSOR message

        /*
        (WEIGHT_VAL - NO_WEIGHT_VAL) / Sensitivitätsfaktor = Gewicht in Gramm
        (WEIGHT_VAL - NO_WEIGHT_VAL) / (100g_val /100) ist die Formel zur Berechnung des Gewichts in Gramm nach der Kalibrierung mit einem 100 Gramm Gewicht.
        100g_val /100 gibt den Sensitivitätsfaktor an, der angibt, wie viel sich der Sensorwert ändert, wenn sich das Gewicht um ein Gramm ändert.
        Durch die Division von 100g_val durch 100 wird der Sensitivitätsfaktor berechnet, und durch die Division von (WEIGHT_VAL - NO_WEIGHT_VAL) durch den Sensitivitätsfaktor wird das Gewicht in Gramm berechnet.

        Beispiel:
        (WEIGHT_VAL - NO_WEIGHT_VAL) / (100g_val /100) = Gewicht in Gramm
        (2400 - 2000) / (2450 /100) = 80 Gramm
        */
        // HX711 circuit wiring
        const int LOADCELL_DOUT_PIN = (int)data["pin_dout"];
        const int LOADCELL_SCK_PIN = (int)data["pin_sck"];
        HX711 scale;
        scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
```

```
      // Get the weight value from the scale
      long weight_val = scale.get_units();
      outgoingData["value"] = weight_val;
      break;
    case 4:
      resetFunc();  //call reset
      break;
    default:
      // Handle unknown message type
      outgoingData[""] = 0;
      break;
  }
```

Es wird gestartet mit einem Switch-Statement.

Hier finden sich die einzelnen Types wieder, welche anstatt beispielsweise ENUM's wie GET_VALUE in Arduino nur integer sind, welche diese repräsentieren.

In dem case 1 (SET_PIN) befindet sich der Code um einen Arduino-Pin Digital und analog auf einen Wert (data["value"] zu setzen.

In case 2 (GET_PIN) wird ein Arduino-Pin Digital oder analog ausgelesen.

Bei case 3 (GET_SENSOR) wird ein HX711-Sensor abgefragt.

In die Anfrage-Daten kommen als Parameter pin_dout sowie pin_sck mit, danach wird ein Wagen-Objekt initialisiert und danach versucht auszulesen.

Bei case 4 (RESTART) soll ein Reset des Arduinos ausgeführt werden. Wie oben erklärt wird die resetFunc ausgeführt.

Default sendet einfach leere Daten als Fehler zurück.

```
// Prepare the outgoing JSON message
  outgoingJson["id"] = id;
  outgoingJson["type"] = 0;
  outgoingJson["data"] = outgoingData;

  // Send the outgoing JSON message
  serializeJson(outgoingJson, Serial);
}
```

Zuletzt wird die outgoingJson-ID, -Type sowie die Daten gesetzt.

Das JSON wird nun serialized, also in Text-Gewandelt und dann über das Serial-Objekt gesendet.

Statistik des Programmiercodes

Programmiercode
23810 Zeilen

Zeitaufwand
114 Stunden und 21 Minuten

Importierte Bibliotheken
30 Stück

Dateien (Klassen und Objekte)
~96 Dateien

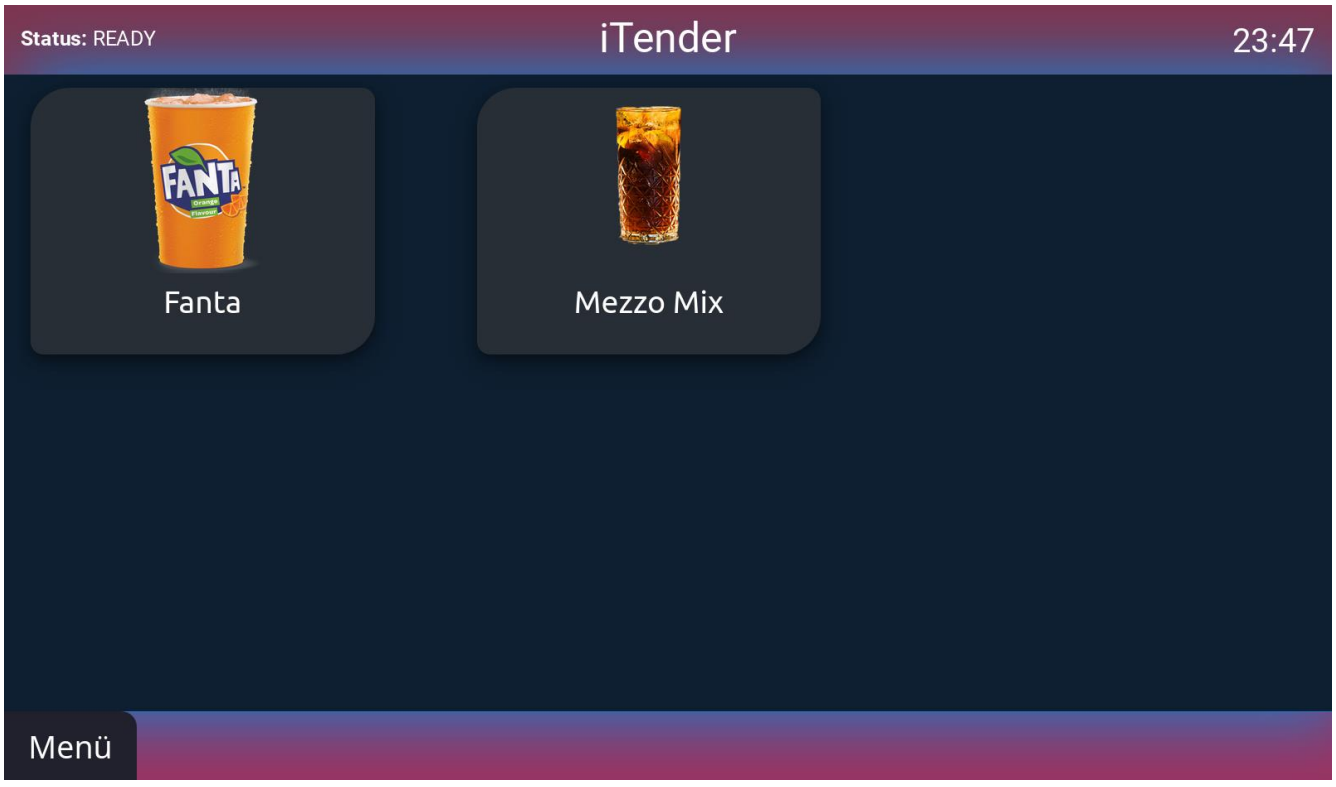
Die Oberfläche

Der Server und der Client kommunizieren über einen WebSocket, welchen man sich als eine Art Chat-Kanal vorstellen kann.
In der Füll-Fortschritt Anzeige wird beispielsweise READY oder FILLING angezeigt, um dem Nutzer einen ungefähren Anhaltspunkt zu bieten, was der aktuelle Status seines Drinks ist.
Die Oberfläche ist sowohl über das Display im iTender, aber auch über ein Tablet steuerbar.
Die Oberfläche sendet hierfür Befehle an den Server und dieser verarbeitet sie und gibt ggfs. eine Antwort.

Oberfläche

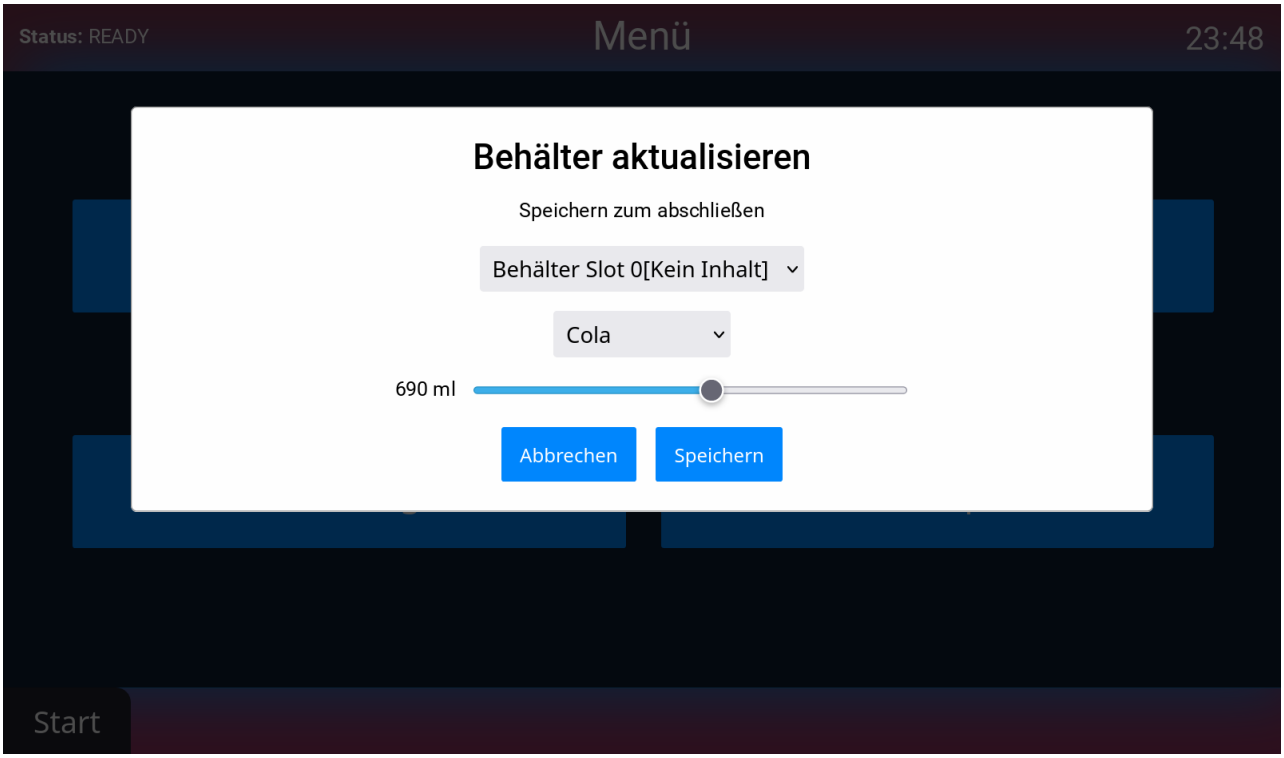
Auf der Startseite werden alle aktuell verfügbaren Drinks angezeigt.
Diese können dann einfach mit einem Klick ausgewählt und hergestellt werden.

Die verfügbaren Drinks aktualisieren sich automatisch mit den aktuell ausgewählten Getränken.



Das Menü dient als Navigationsherz um alle Panels zu erreichen.

In den Containern können die Behälter Inhalte aktualisiert werden.
Man wählt die "Zutat" aus und danach, wie voll der Behälter nun ist.
In der Regel kann das auch automatisch eingemessen werden, wenn alle Sensoren eingestellt sind.



Während des Befüllens wird ein Pop-up angezeigt, welches den ungefähren Füllstatus anhand einer Animation anzeigt.

Das Setup-Menü ist das erste Menü, was nach dem Einrichten erscheint, und dient zur Grundkonfiguration.

Unter den Einstellungen können wir aktuell nur einen Menü-Punkt hinterlegen. Derzeit befindet sich dort der Taster, um die Getränke von der Cloud zu aktualisieren. In Zukunft soll hier aber beispielsweise die Ambiente-Farbe sowie das WiFi eingerichtet werden.

Die Hardware

Hauptplatine

Wie bereits oben erwähnt möchten wir einen Raspberry Pi als "Hauptplatine" nutzen. Diese Entscheidung fiel uns leicht, da wir bereits im vergangenen mit diesen kleinen PCs gearbeitet hatten und er alle Zwecke erfüllte.

Ein Arduino bzw. ioT Entwicklungsboard hätte nicht genug Rechenleistung, um die Webseite darzustellen und die einzelnen Drinks zu berechnen.
Er hätte schlichtweg einfach zu wenig Arbeitsspeicher, um die Drinks zu verarbeiten.

Da wir allerdings Ausgänge in Form von Anschlusspins benötigen, blieb uns nur noch der Raspberry Pi übrig.

Peristaltik Pumpen

Wieso?

Peristaltik Pumpen sind eine Art von Pumpe, die auf der natürlichen Wellenbewegung des menschlichen Körpers basiert. Sie sind besonders nützlich für Anwendungen, bei denen ein präziser Flüssigkeitsfluss und eine geringe Verunreinigung von entscheidender Bedeutung sind. Im Folgenden werden einige der Vorteile von Peristaltik Pumpen und mögliche Nachteile beschrieben.

Einer der größten Vorteile von Peristaltik Pumpen ist ihre Fähigkeit, Flüssigkeiten präzise und kontinuierlich zu dosieren. Da die Pumpe durch den Druck des Schlauchs betrieben wird, ist sie in der Lage, sehr kleine Flüssigkeitsmengen zu fördern, was sie ideal für Anwendungen wie die Medizintechnik, die Analytik oder die Pharmazie macht.

Peristaltik Pumpen sind auch besonders gut geeignet für Anwendungen, bei denen eine geringe Verunreinigung erforderlich ist. Da die Flüssigkeit nicht mit beweglichen Teilen in Kontakt kommt, ist die Gefahr von Verunreinigungen oder Abnutzung geringer als bei anderen Arten von Pumpen.

Ein weiterer Vorteil von Peristaltik Pumpen ist ihre Fähigkeit, aggressive oder viskose Medien zu fördern. Da die Flüssigkeit nur durch den Schlauch befördert wird, kann die Pumpe auch Medien mit hohem Schwebstoffgehalt oder hoher Viskosität problemlos handhaben. Dies macht sie ideal für Anwendungen wie die Chemie- oder Lebensmittelindustrie.

Peristaltik Pumpen sind auch sehr zuverlässig und langlebig, da sie keine beweglichen Teile haben und somit weniger anfällig für Verschleiß und Ausfälle sind. Sie sind einfach zu warten und zu reparieren, was die Gesamtbetriebskosten senkt.

Ein möglicher Nachteil von Peristaltik Pumpen ist, dass sie eine höhere Anschaffungskosten haben können als andere Arten von Pumpen. Jedoch ist es zu berücksichtigen, dass die längere Lebensdauer und die geringeren Wartungskosten diesen höheren Anschaffungspreis auf lange Sicht ausgleichen können. Ein weiterer Nachteil kann sein, dass sie nicht geeignet sind für Anwendungen mit sehr hohem Drücken oder sehr hohen Temperaturen.

In Zusammenfassung bieten Peristaltik Pumpen eine präzise und kontinuierliche Dosierung von Flüssigkeiten, eine geringe Verunreinigung, die Fähigkeit, aggressive oder viskose Medien zu fördern, Zuverlässigkeit und Langlebigkeit, und sind einfach zu warten und zu reparieren. Obwohl sie höhere Anschaffungskosten haben können, sind sie auf lange Sicht eine kosteneffiziente Wahl für bestimmte Anwendungen.

HX711 Wägezellen

Die HX711 Wägezelle besteht aus zwei Teilen: dem HX711 IC selbst und dem Wägezellen-Modul. Das Wägezellen-Modul enthält den HX711 IC sowie die nötigen Verstärker, Filter und Schaltungen, um das Signal der Wägezelle zu verarbeiten. Es kann direkt an eine Vielzahl von Wägezellen angeschlossen werden und ermöglicht es so, die Lastaufnahme von Wägezellen zu verbessern.

Der HX711 IC selbst verfügt über zwei Eingänge, die für die Verarbeitung des Signals der Wägezelle verwendet werden. Der DAT-Eingang wird verwendet, um das analoge Signal der Wägezelle zu empfangen, während der CLK-Eingang verwendet wird, um das Signal zu synchronisieren und die Daten auszulesen.
Das Modul kann mit einer Vielzahl von Mikrocontroller-Systemen wie Arduino, Raspberry Pi usw. verwendet werden. Es gibt auch eine Bibliothek, die es ermöglicht, den HX711 IC einfach in die Anwendung zu integrieren und die Daten auszulesen.

Die HX711 Wägezelle bietet eine hohe Genauigkeit und Auflösung und eignet sich daher besonders für Anwendungen, bei denen eine hohe Genauigkeit erforderlich ist, wie z.B. in der Lebensmittelindustrie, der Medizintechnik oder im Laborbereich.

Es ist jedoch wichtig zu beachten, dass die Genauigkeit auch von der Wägezelle selbst und von der Kalibrierung abhängt. Eine regelmäßige Kalibrierung und Überprüfung des Skalierungsfaktors und der Nullpunktlast ist daher erforderlich, um eine hohe Genauigkeit zu gewährleisten.

Ein weiteres wichtiges Thema ist die Umgebungsbedingungen, wie z.B. Temperatur, Feuchtigkeit und Vibrationen können die Genauigkeit beeinflussen, es ist daher empfehlenswert, die Wägezelle in einer stabilen Umgebung zu betreiben und diese Faktoren zu berücksichtigen.

Insgesamt ist die HX711 Wägezelle ein leistungsfähiges und zuverlässiges Werkzeug, um die Lastaufnahme von Wägezellen zu verbessern und ist in vielen Anwendungen einsetzbar, vor allem in denen die hohe Genauigkeit erfordert wird.

In unserem Arduino-Code verwenden wir die Bibliothek Hx711, welche die Kommunikation mittels HX711-Wägezelle deutlich erleichtert.
Einige Funktionen werden nun erklärt:

Die Methode "tare()" der HX711-Bibliothek wird verwendet, um die Wägezelle auf null zu setzen. Dies bedeutet, dass die aktuelle Last auf der Wägezelle als Nullpunkt angesehen wird und die Messungen, die danach mit der Methode "get_units(5)" durchgeführt werden, relativ zu diesem Nullpunkt erfolgen.

Wenn die tare() Methode aufgerufen wird, berechnet das HX711-Modul einen Mittelwert aus einer bestimmten Anzahl von Messungen und speichert diesen als Nullpunkt. Danach werden alle Messungen, die mit der Methode "get_units(5)" durchgeführt werden, relativ zu diesem Nullpunkt berechnet.

Wieso nicht?

Es gibt mehrere Gründe, warum es nicht sinnvoll ist, einen Ultraschall-Sensor wie den HC-SR04 in einem Getränkespender wie dem iTender zu verwenden. Einer der größten Nachteile ist die niedrige Skalierbarkeit des Sensors. Da der Ultraschall-Sensor auf einer bestimmten Distanz basiert, funktioniert er nur innerhalb einer begrenzten Reichweite. Dies bedeutet, dass er nicht in der Lage ist, genaue Messungen von Behältern zu erhalten, die sich weiter entfernt befinden.

Ein weiterer Nachteil ist die ungenauen und unzuverlässigen Ergebnisse, die der Sensor liefert. Der Ultraschall-Sensor ist anfällig für Fehler, die durch Hindernisse oder Reflektionen verursacht werden. Dies führt dazu, dass der Sensor ungenaue Messungen liefert, die nicht zuverlässig sind.

Ein weiteres Problem ist, dass der Sensor immer einmalig eingemessen werden muss, damit er ordnungsgemäß funktioniert. Dies bedeutet, dass jedes Mal, wenn ein neuer Behälter hinzugefügt wird, der Sensor erneut eingemessen werden muss, um sicherzustellen, dass die Messungen korrekt sind. Dies kann Zeit und Ressourcen verschwenden.

Schließlich ist der Ultraschall-Sensor nicht dynamisch genug, um mit anderen Behältern im iTender-System zu arbeiten. Da der Sensor auf eine bestimmte Distanz basiert, funktioniert er nicht gut, wenn mehrere Behälter im System vorhanden sind. Dies kann zu Fehlern führen, wenn der Sensor versucht, die Füllstände von mehreren Behältern gleichzeitig zu messen.

Zusammenfassend lässt sich sagen, dass der Ultraschall-Sensor HC-SR04 für den Einsatz in einem Getränkespender wie dem iTender nicht die beste Wahl ist. Aufgrund seiner niedrigen Skalierbarkeit, ungenauen Ergebnissen, der Notwendigkeit der Einmessung und der Unfähigkeit, dynamisch mit anderen Behältern im System zu arbeiten, ist es wahrscheinlich besser, eine andere Technologie zu verwenden.

Wir haben uns anfänglich für den Ultraschall-Sensor entschieden. Er sollte über dem Glas hängen und den Abstand zum gefüllten Messen.
Leider treten dabei einige Nachteile und Probleme auf, die wir bis dato nicht bedacht haben.
Unter anderem muss vor jedem wechseln eines Behälters (Größen Wechsel) eine neue Einmessung gestartet werden.
Diese Einmessung muss bei den Wägezellen beispielsweise nur alle paar Tage/Wochen ausgeführt werden.

Eidesstattliche Erklärung

Hiermit versichern wir (Tobias Hopp, Nick Thiele), dass wir die Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht.

Sankt Augustin, 05.02.2023

Nick Thiele

Tobias Hopp