Projektdokumentation

SS 2015

an der
Hochschule für Technik Stuttgart

im Studiengang IF, IL

Datenerfassung mit dem Raspberry Pi und Cloud Visualisierung

Autoren:

Abgabe am: 12. Juni2015

Vorwort

In dieser Dokumentation werden bestimmte Begriffe zwecks Hervorhebung speziell formatiert:

- Besondere Bezeichnungen wie der Name von Frameworks, Modulen oder von Softwareprodukten werden bei Ihrem ersten Auftreten stets *kursiv* ausgezeichnet.
- Für Terminal-Kommandos wird die `Typewriter-Schrift` verwendet.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des von Professor Knauth gehaltenen Moduls Ubiquitous Computing, sollen die Studenten die gelernten Inhalte in einem praktischen Projekt anwenden. Ubiquitous Computing beschreibt das Phänomen, dass Computer immer mehr unsere Umgebung durchdringen und deshalb vom Anwender kaum noch bewusst wahrgenommen werden. Dies wird durch eine Verbesserung der Computertechnik und eine stetige Miniaturisierung der Hardware, sowie durch den Einsatz von Sensoren und Funktechnik ermöglicht.

Für die Projektdurchführung fanden sich die Studenten zu Projektgruppen zusammen. Die einzelnen Gruppen konnten sich für ein Thema aus dem Bereich des Ubiquitous Computing frei entscheiden.

Unsere Gruppe setzte sich zusammen aus vom Studiengang Informatik und vom Studiengang Informatik.

Das von uns gewählte Projektthema umfasste die Erfassung von Umgebungsdaten durch den Einsatz von mobilen Datenstationen, mit der anschließenden Datenübertragung zu einem zentralen Server zur Aufbereitung.

Als Datenstation wurde der bekannte Einplatinencomputer Raspberry Pi[1] verwendet. Dieser sollte zunächst über angeschlossene Sensoren mehrere Umgebungsdaten, wie beispielsweise die Lufttemperatur oder die Helligkeit erfassen. Anschließend sollten diese Daten über das Internet an einen zentralen Server übermittelt werden. Der Server sollte die empfangenen Werte aufbereiten um sie anschaulich auf einer Internetseite darstellen zu können. Des weiteren sollte das gesamte System jederzeit durch neue Datenstationen, mit beliebigen Sensortypen erweitert werden können.

[1] https://www.raspberrypi.org

2 Projektplanung

Zu Projektbeginn wurden als erstes alle bereits vorhandenen Kenntnisse und Erfahrungen der beteiligten Projektmitglieder im Bereich der Softwareentwicklung erfasst. Das Ziel dieser Maÿnahme war die spätere Aufgabenverteilung unter Berücksichtigung der Fähigkeiten der einzelnen Projektmitglieder vornehmen zu können.

Im zweiten Schritt wurde eine gemeinsames Projektziel diskutiert und erarbeitet. Der Projektumfang, die Prioritäten und optionale Features wurden dokumentiert. Dabei wurde auch schon eine grobe Architektur der Gesamtlösung festgelegt und die einzelnen Systemkomponenten auf die Projektmitglieder verteilt. Die Gesamtarchitektur umfasste eine Datenstation, einen Applikationsserver mit der Datenbank, einen Webserver zur Ausgabe der Internetseite und die Internetseite selbst. Die Tabelle 1 zeigt die grundlegende Aufgabenverteilung innerhalb des Projektteams.

Architekturkomponente	Zuständigkeit
	Internetseite
	Webserver
	Applikationsserver
	Datenstation

Tabelle 1: Aufteilung der Architekturkomponenten

Im nächsten Schritt wurden die Schnittstellen zwischen den Architekturkomponenten diskutiert und dokumentiert. Durch diese grundlegende Planung war es jedem Projektmigtlied möglich seine Komponente weitestgehend selbstständig und unabhängig zu entwickeln.

2.1 Organisation

Damit eine ausreichende Kommunikation während der Projektlaufzeit gegeben war, entschieden wir uns dafür den Instant Messenger *Telegram*[1] zu nutzen. Ein wichtiger Vorteil gegenüber anderen Kommunikationslösungen war hierbei die Möglichkeit Dateien jeden Formates mit einer Größe von bis zu 1GB versenden zu können. Des weiteren wird der Chatverlauf bei Telegram auf unbegrenzte Zeit gespeichert und verfügt über eine performante Suchmöglichkeit um Informationen aus schon älteren Konversationen schnell und einfach finden zu können.

Des weiteren trafen wir uns jeden Dienstagvormittag im Rahmen der PVC-Praxisstunde und begegneten uns auch im Hochschulalltag sehr häufig. Dadurch war eine nachhaltige Kommunikation gewährleistet und es kam zu keinerlei Problemen aufgrund unzureichender Absprachen.

2.2 Dokumentenverwaltung

Für projektrelevante Dokumente wie der Projektantrag, das Journal und die Dokumentation wurde ein Verzechnis in *Google Drive*[2] angelegt. Durch die Ablage in der Cloud hatte somit jedes Projektmitglied jederzeit Zugriff auf die Dokumente. Google Drive stellt den Service *Google Docs* bereit, durch den webbasierte Textverarbeitungs-, Tabellenkalkulations- und Präsentationsanwendngen genutzt werden können. Da jedes Projektmitglied volle Zugriffsrechte besaß, konnten die Dokumente jederzeit bearbeitet werden. Die Abbildung 1 zeigt das gesamte Projektverzeichnis.

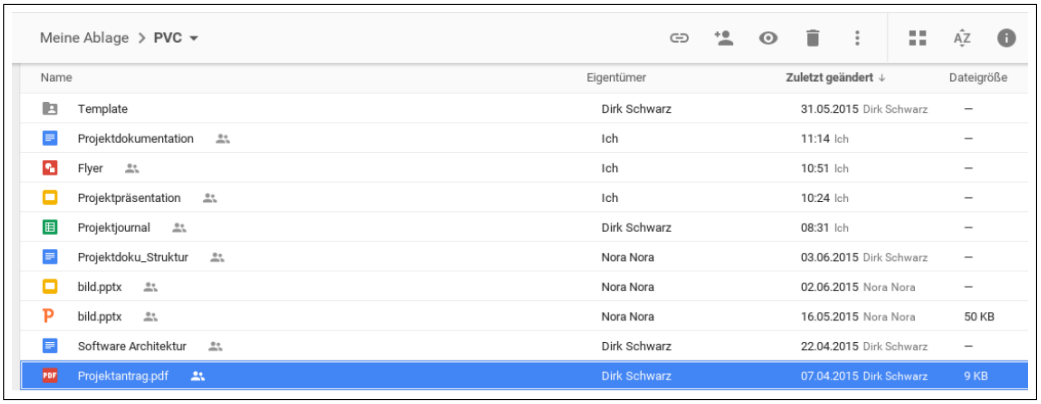

Abbildung 1: Google Drive

[1] https://telegram.org/
[2] https://www.google.com/intl/de_de/drive/

3 Entwicklung

Um möglichst frühzeitig funktionsfähige Ergebnisse zu bekommen, wurde zu Beginn eine öffentlich verfügbare Domäne auf einem vServer im Internet eingerichtet. Hier wurde in möglichst kurzen Zyklen der aktuelle Entwicklungstand online gestellt. Der Vorteil der daraus enstand waren eine höhere Motivation der Entwickler und eine wirkungsvolle Demonstrationsmöglichkeit der bisher entwickelten Komponenten.

3.1 Versionsverwaltung

Als Versionsverwaltung wurde *GitHub*[1] genutzt. Auf GitHub ist die Erstellung eines öffentlichen Repositories kostenlos. Die URL unseres Repositories lautet:

`https://github.com/r0bn/raspberry-data-station`

Zusätzlich zu der einfachen Einbindung eines Remote-Repository in den lokalen Workspace bietet GitHub die Möglichkeit den Source Code komplett online zu betrachten, zu verlinken oder zu kommentieren. Von diesen Funktionen wurde auch durch das Instant Messanging mit Telegram Gebrauch gemacht.

3.2 vServer

Als *vServer* wurde ein bereits bestehender Serververtrag eines Projektmitglieds verwendet. Somit enstanden keine zusätzlichen Mietkosten für Serverhardware. Bei dem vServer handelt es sich um die kleinste Version der verfügbaren Services[2] des Anbieters. Als Serverbetriebssystem wird Linux genutzt. Die Hardwareleistung des Servers war völlig ausreichend für dieses Projekt. Die URL für unsere Intersetseite lautet:

`http://pvc.r9u.de`

3.3 Deployment

Der aktuellsten Entwicklungsstand wurde stets durch eine ssh-Verbindung auf den Server deployt. Eine mögliche Automatisierung des Deployments war vom Aufwand her nicht sinnvoll und wurde deshalb nicht umgesetzt. Auf dem Server wurde ebenfalls ein Git-Repository erstellt.

[1] `https://github.com/`
[2] `https://www.united-hoster.de/server/standard-vserver/vps-linux.html`

Auf dem Server liefen permanent zwei Prozesse[3]:

- Bei dem ersten Prozess handelte es sich um den Applikationsserver. Dieser lief auf Port 7090.
- Der zweite Prozess war der Webserver zur Ausgabe der Internetseite. Dieser lief auf Port 7085.

Beide Prozesse wurden durch das Tool *foreverjs*[4] verwaltet. Die Abbildung 2 zeigt alle serverseitig permanent aktiven Scripte.

Abbildung 2: Aktive Scripte auf dem Server

Nachdem das Git-Repository auf dem Server aktualisert wurde, reichte ein Neustart des jeweiligen Prozesses aus um die neuen Änderungen zu integrieren. Um die Internetseite unter der Domain verfügbar zu machen, wurde *nginx*[5] als Proxy Server eingesetzt.

[3] https://github.com/r0bn/raspberry-data-station/blob/master/config.json
[4] https://github.com/foreverjs/forever
[5] http://nginx.org/

4 Systemarchitektur

Ein wichtiger Aspekt der Systemarchitektur war uns die stikte Trennung der erfassten Daten von der Visualisierung.
Durch diese Trennung sind die von den Datenstationen erfassten und in der Datenbank abgespeicherten Umgebungsdaten auch anderweitig nutzbar. Beispielsweise können diese auch durch andere Arten visualisiert werden.
Umgekehrt ist auch der Webserver mit der integrierten Visualisierungslösung unabhängig von der Datenquelle. Beispielsweise könnten auf der Internetseite somit auch Wetterdaten visualisiert werden.
Die Abbildung 3 veranschaulicht diesen Sachverhalt.

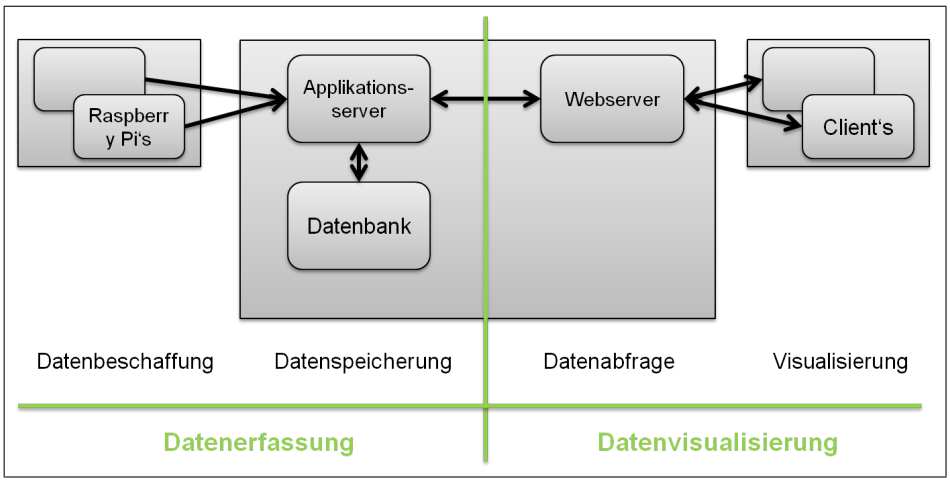

Abbildung 3: System Architektur

Schon zu Projektbeginn entschieden wir uns *Node.js*[1] für die Entwicklung der Architekurkomponenten zu verwenden. Bei Node.js handelt es sich um ein *JavaScript* basiertes Framework für die Entwicklung von Webanwendungen. Das Framework beinhaltet zahlreiche Module die durch den Paketmanager *npm*[2] installiert werden können. Innerhalb dieses Projektes wurde beispielsweise das Modul *express* verwendet, welches ein sehr einfaches Web-Framework bereit stellt.
Die Schnittstellen der einzelnen Komponenten wurden durch REST-Endpunkte erstellt und können durch HTTP-Requests genutzt werden. Übergabe- und Rückgabewerte der Schnittstellen werden im JSON-Format versendet.
In den nächsten Kapiteln werden die einzelnen Architekturkomponenten genauer erläutert.

[1] `https://nodejs.org/`
[2] `https://www.npmjs.com/`

4.1 Clients und Internetseite

Die zentrale Aufgabe des Clients ist die Sensordaten in Diagrammen zu visualisieren. Dafür wählt der Benutzer auf der Internetseite Abfrageparameter aus und bestätigt diese um bestimmte Daten anzufordern und angezeigt zu bekommen.
Als Vergleichsfunktionen wurden der Sensorvergleich, der Vergleich von Datenstationen anhand eines Sensortyps, der Zeitraumvergleich und der Vergleich von zwei Zeiträumen einer Datenstation unter Berücksichtigung eines Sensortyps.

Die beiden folgenden Abbildungen 4 und 5 zeigen das Layout der entwickelten Internetseite einmal auf einem Desktop und auf einem Mobilgerät

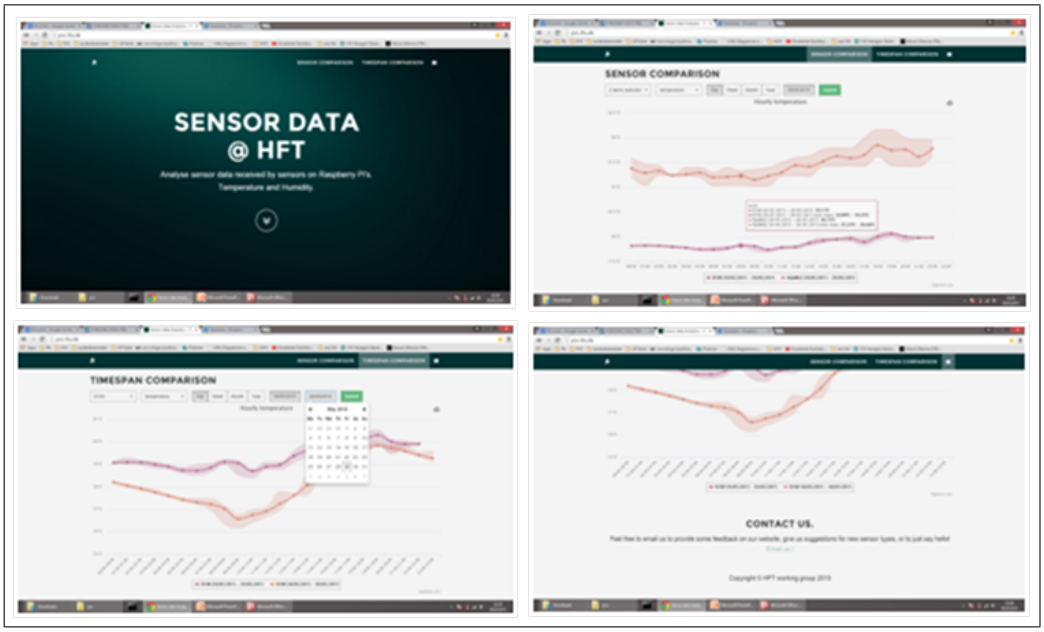

Abbildung 4: Respnonsive Desktop

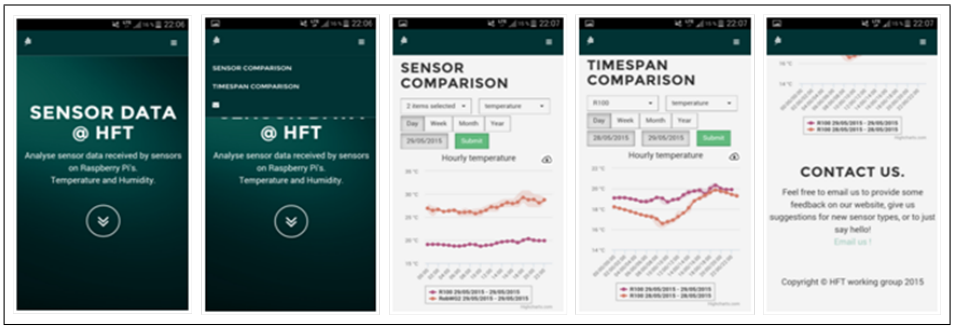

Abbildung 5: Respnonsive Mobile

4.1.1 Umsetzung

Für das Layout wurde das HTML/CSS/JS-Framework *Bootstrap*[3] verwendet. Es bietet die Möglichkeit responsive Internetseiten zu erstellen. Desweiteren wurde das Bootstrap-Template *Grayscale*[4] genutzt. Nennenswert ist hier eine responsive Navigationsbar, deren Layout sich den kleineren Displays von mobilen Endgeräten anpasst. Ebenfalls von Bootstrap ist das *Datepicker*-Plugin[5] für die Auswahl des Datums und einen *Selectpicker*-Plugin[6] für die Auswahl der Datenstationen und Sensortyps.

Die Diagramme werden clientseitig mit *Highcharts*[7] generiert. Highcharts ist eine umfangreiche JavaScript-Bibliothek, welche dynamische Diagramme im SVG-Format erstellt. Zusätzlich wird *highcharts-more.js* verwendet um Flächendiagramme zu unterstützen und *exporting.js* um den Export der Daten in verschiedenen Bildformaten (PNG,JPEG,PDF,SVG) und eine Druckfunktion zu ermöglichen. Highcharts bietet auf jQuery-Basis die Aktualisierung des Diagrammtitels, der Achsen, der Daten und der Gestaltung der Datenreihen an. So können die Diagramm-Daten durch die Nutzung von AJAX aktualisiert werden.

4.1.2 Ablauf

Eine initiale Abfrage nach verfügbaren Sensortypen und Datenstationen erfolgt beim Laden der Internetseite. Per Selectpicker werden die Beschreibungen und IDs in die Drop-Down-Listen geschrieben. Somit können für die Datenabfrage die IDs ausgelesen werden.

Es werden die IDs der Datenstationen, des Sensortyps, die Zeitspanne und das Startdatum bzw. die Startdaten per jQuery ausgelesen. Die Abfragen für den Sensor- und den Zeitraumvergleich werden beide an den "/data"-Endpunkt des Webservers gesendet. Mit Hilfe der bereits genannten Highcharts-Funktionen wird das Diagramm generiert. Zusätzlich wird hierbei die Sensortyp-ID und die Datenstation-ID anhand der Daten aus der initialen Abfrage in deren Beschreibung umgewandelt.
Es werden für jeden Sensor zwei Datenreihen im Diagramm erzeugt.
Die erste Datenreihe beschreibt ein Liniendiagramm ("spline"). Dafür benutzt Highcharts die "y"-Werte der formatierten Sensordaten.
Die zweite Datenreihe beschreibt ein Flächendiagramm ("areasplinerange"). Es benutzt die "low"- und "high"-Werte. Diese Datenreihe wird mit "min-max" beschriftet und wird zu der ersten Datenreihenreihe verlinkt. Außerdem werden die Dateneinheiten in dem Tooltip und den Achsen aktualisiert.
Die Abbildung 6 zeigt ein beispielhaftes Diagramm.

[3] http://getbootstrap.com/
[4] http://startbootstrap.com/template-overviews/grayscale/
[5] https://bootstrap-datepicker.readthedocs.org/en/latest/
[6] http://silviomoreto.github.io/bootstrap-select/
[7] http://www.highcharts.com/

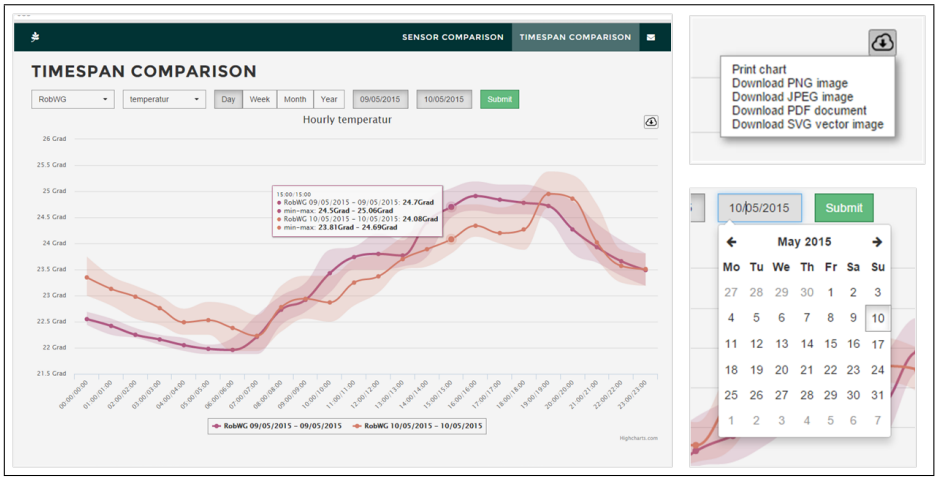

Abbildung 6: Beispieldiagramm

4.1.3 Entwicklungsumgebung

Als Entwicklungsumgebung für die Entwicklung der Clients wurde *Netbeans* verwendet. Für die Frontend-Entwicklung bietet Netbeans einen Überblick und eine leichte Navigation in der Ordnerstruktur. Außerdem ist die IDE mit GitHub kompatibel.

Die ersten Prototypen der Diagramme entstanden mit der online Entwicklungsumgebung *JSFiddle*[8]. Highcharts bietet über dieses Tool bereits Basis-Diagramme zur Modifizierung an. JSFiddle bietet die Möglichkeit online JavaScript-Code, sowie HTML und CSS schnell zu modifizieren und zu interpretieren. Die Online-Entwicklungsumgebung wurde weiterhin als parallele für eine schnelle Anpassung der Diagramme genutzt. Die Abbildung 7 zeigt die Oberfläche von JSFiddle.

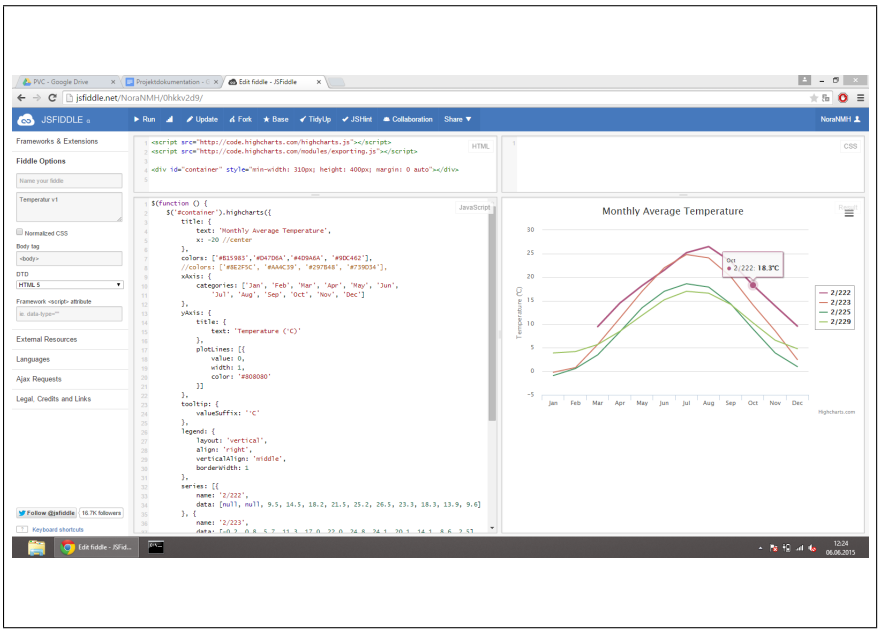

Abbildung 7: JSFiddle

[8] https://jsfiddle.net/NoraNMH/0hkkv2d9/

4.2 Webserver

Um auf dem Client möglichst wenig Logik implementieren zu müssen, wurde ein Webserver zwischen dem Client und dem Applikationsserver eingefügt. Dieser formatiert die Daten des Applikationsservers in Formate die mit Highcharts kompatibel und somit einfach integrierbar sind. Eine weitere Funktion ist die initiale Abfrage von Informationen zu den Datenstationen und Sensortypen.

4.2.1 Umsetzung

Die Anforderungen an den Webserver werden durch mehrere REST-Endpunkte umgesetzt. In der Nachfolgenden Aufzählung werden diese erläutert.

- **GET**: "**/init**"

  Dieser Endpunkt liefert alle Informationen zu den Datenstationen und den Sensortypen aus der Datenbank als JSON-Objekt. Diesem Endpunkt werden keine Parameter übergeben. Im Folgenden ist eine beispielhafte Rückgabe zu sehen.

Rückgabebeispiel init-Endpunkt

```
1
2  {
3     "datastations": [{
4        "ID": 100,
5        "Area": "RobWG"
6     }, {
7        "ID": 101,
8        "Area": "RobWG2"
9     }],
10       "sensortypes": [{
11       "ID": 1,
12       "Name": "temperatur",
13       "Unit": "C"
14    }, {
15       "ID": 2,
16       "Name": "pressure",
17       "Unit": "hPa"
18    }]
19 }
```

- **GET: "/data"**

  Dieser Endpunkt liefert die Daten für einen Vergleich der Sensoren und Zeiträume. Es werden die folgenden Parameter benötigt:

  - **datastationID**: Ein oder mehrere IDs der ausgewählten Datenstationen, separiert durch Komma.
  - **sensortypeID**: Die ID des gewünschten Sensortyps.
  - **timespan**: Der gewünschte Zeitraum.
  - **date**: Ein oder mehrere IDs der gewünschten Datumsangaben, separiert durch Komma.

  Die Rückgabe dieses Endpunktes beinhaltet die benutzte Zeitraum-Intervall-Einheit, die Sensortyp-ID für den Diagrammtitel und formatierte Zeitraum-Intervalle für die x-Achse. Pro angefragtem Sensor werden Informationen zur Datenstation-ID und deren Zeitspanne für die Beschriftung der Datenreihe und der minimale und maximale Messwert geliefert. Nachfolgend wieder eine beispielhafte Rückgabe.

Rückgabebeispiel data-Endpunkt

```
1
2  {
3     "data": {
4     "title": "Hourly ",
5     "titlesensortypeID": "1",
6     "timeframes": ["00:00", "01:00",...],
7     "dataPerArea": [{
8     "name": "100",
9     "nameTimeframe": "09/05/2015 - 09/05/2015",
10    "data": [{
11    "y": 22.55,
12    "low": 22.44,
13    "high": 22.69
14 }, {..........
```

4.2.2 Frameworks

Auch für den Webserver wurde *Node.js* verwendet, um mit den vorhandenen Modulen den Server schnell programmieren und sich dann auf die Datenverarbeitung konzentrieren zu können. Durch die Verwendung des Moduls *express* kann die Internetseite mit folgender Codezeile ausgeliefert werden:

```
app.use(express.static(path.resolve(__dirname +'/../PVCWebsite')));
```

Auch REST-Endpunkte lassen sich durch express leicht festlegen:

```
app.get('/data', function (req, res) { //doSomething }
```

Um die Anfragen an den Applikationsserver weiterzuleiten wurde das *Request*-Modul eingesetzt. Hiermit lassen sich auf einfache Weise REST-Anfragen generieren:

```
request('http://www.google.com',
function (error, response, body) {//doSomething})
```

Außerdem wurde noch das *Async*-Modul verwendet. Dieses vereinfacht den Umgang mit asynchronen Methoden und damit ließ sich eine Verschachtelung von Methoden die bei aufeinander aufbauenden asynchronen Methoden entsteht vermeiden.

4.2.3 Entwicklungsumgebung

Bei der Arbeit am Webserver wurde der Texteditor Notepad++ als Entwicklungswerkzeug verwendet. Es war keine besondere Einrichtung notwendig.

4.2.4 Ablauf und Schwierigkeiten

Da das Aufsetzen des Webservers und die Auslieferung der Internetseite durch die Verwendung der Node.js-Module schnell erledigt war, wurde die Hauptaufgabe bei der Entwicklung des Webservers die Kommunikation zwischen dem Client und dem Applikationsserver zu regeln.
Der einfachere Teil war es die Anfragen des Clients an den Applikationsserver weiterzuleiten.
Der schwierigere Teil war die Antwort des Applikationsservers in ein für den Client geeignetes Format umzuwandeln. Hierbei mussten die aggregierten Daten korrekt auf einem Zeitstrahl abgebildet werden. Dadurch, dass mehrere Datenstationen abgefragt werden können und drei verschiedene Aggregationen unterstützt werden, erwies sich das Zuordnen der Daten als nicht einfach. Schlussendlich wurde dies durch die Verwendung einer Map in der die Datenstationen mit Datenlisten verknüpft sind umgesetzt. In diesen Listen werden die Daten in chronologisch richtiger Reihenfolge gespeichert. Mit der Möglichkeit des Zeitraumvergleichs entstand eine zusätzliche Schwierigkeit, da jetzt auch ein zweiter Zeitraum abgefragt werden konnte. Die Lösung bestand darin eine zweite Anfrage zu erstellen und die zwei Rückgaben des Applikationsservers zu vereinigen.

Eine weitere Schwierigkeit war, dass beim Verarbeiten der rückgelieferten Daten vom Applikationsserver leicht Fehler entstanden, die nicht sofort bemerkt und daher erst einige Zeit später erkannt und behoben wurden.

4.3 Applikationsserver

Der Applikationsserver bildet neben dem Webserver die zweite zentrale Komponente in der Systemarchitektur.
Der Applikationsserver umfasst zwei Funktionen:

1. Der Empfang und die Speicherung von Umgebungsdaten der Raspberry Pi-Stationen.
2. Die vorhandenen Daten dem Webserver für die Anzeige auf der Website zur Verfügung stellen.

Für beide Funktionen stellt der Applikationsserver mehrere Endpunkte zur Verfügung. Die Daten werden in einer Datenbank abgespeichert und bei Bedarf abgerufen. Die Schnittstellen für das Einfügen und Anfordern von Daten werden über REST-Endpunkte realisiert.

4.3.1 Umsetzung

Um den Applikationsserver aufzusetzen und die REST-Endpunkte einzurichten wurde (analog zum Webserver) *express* verwendet. Für die Unerstützung des JSON-Formats für die Übergabe von Parametern wurde das Modul *body-parser* genutzt.

Um die Umgebungsdaten der Datenstationen zu speichern war die Verwendung einer Datenbank notwendig. Wir wollten dabei eine möglichst leichtgewichtige Technologie nutzen. Deshalb entschieden wir uns gegen die Installation eines separaten MySQL-Servers und wählten stattdessen das *SQLite3*-Modul[9] für *Node.js*.
Mit dieser SQL-Engine ist es möglich eine SQL-Datenbank innerhalb einer einzigen Datei zu erstellen und zu konfigurieren.

Um mit *SQLite3* beispielsweise eine temporäre Datenbank mit einer Tabelle welche zehn Tupel enthält zu erstellen und auszugeben ist dieser JavaScript-Code notwendig:

[9] https://github.com/mapbox/node-sqlite3

Erstellung einer temporären Datenbank mit SQLite3

```
var sqlite3 = require('sqlite3').verbose();
var db = new sqlite3.Database(':memory:');

db.serialize(function() {
   db.run("CREATE TABLE lorem (info TEXT)");

   var stmt = db.prepare("INSERT INTO lorem VALUES (?)");
   for (var i = 0; i < 10; i++) {
      stmt.run("Ipsum " + i);
   }
   stmt.finalize();

   db.each("SELECT rowid AS id, info FROM lorem", function(err, row) {
      console.log(row.id + ": " + row.info);
   });
});

db.close();
```

Bei der Entwicklung der Datenbank wurde zuerst ein ERM-Modell erstellt. Dieses Modell ist in der Abbildung 8 zu sehen.

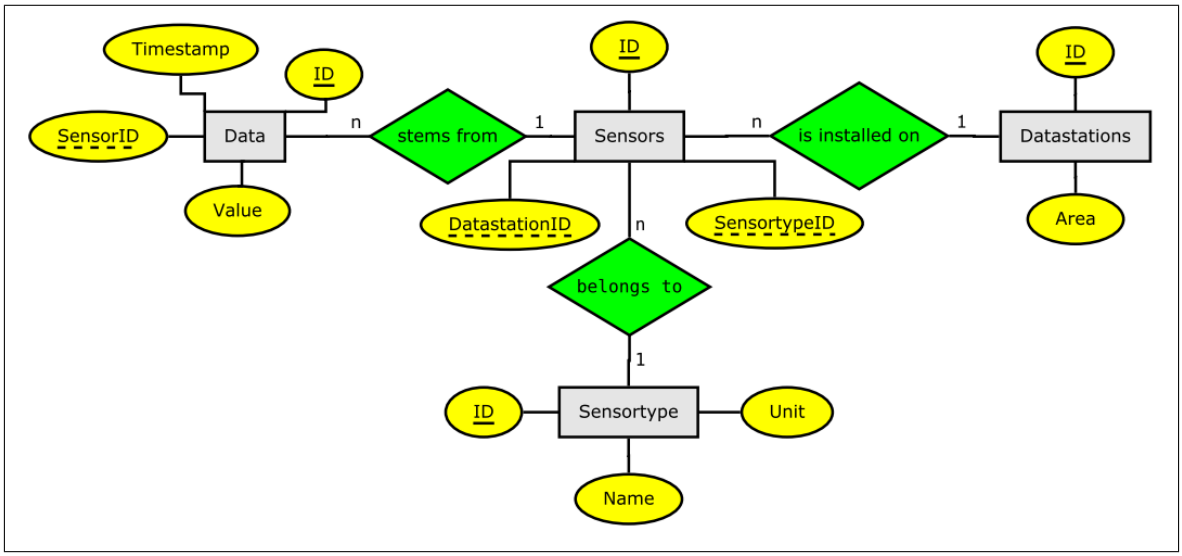

Abbildung 8: ERM-Modell

Wie zu sehen ist besteht die Datenbank aus vier Relationen:

- **Data**

  In dieser Tabelle werden alle empfangenen Umgebungsdaten gespeichert. Zu jedem Datenwert (*Value*) wird auch der Zeitpunkt der Messung (*Timestamp*) festgehalten. Die *SensorID* ist ein Fremdschlüssel der auf die Tabelle *Sensors* verweist. Über diese Verbindung kann der Sensortyp und die Datenstation von der diese Messung stammt ermittelt werden.

- **Datastations**

  In der Tabelle Datastations werden alle Raspberry-Pi-Stationen welche Daten gesendet haben abgespeichert. Für jede Station wird Ihr Standort (*Area*) festgehalten.

- **Sensors**

  In dieser Tabelle wird zu jedem Sensor der dazugehörige Sensortyp und die Station gespeichert. Dementsprechend gibt es einen Fremdschlüssel auf die Tabelle *Sensortype* und einen Fremdschlüssel auf *Datastations*.

- **Sensortype**

  Hier sind alle verschiedenen Sensortypen gespeichert. Es wird jeweils der Name des Sensortyps und die Maßeinheit für die Messwerte festgehalten.

Die Umsetzung des Datenbank-Schemas mit SQL wird in Abbildung 9 gezeigt.

```
CREATE TABLE "Data" (
        `Timestamp`  TEXT NOT NULL,
        `ID`         INTEGER,
        `SensorID`   INTEGER NOT NULL,
        `Value`      NUMERIC NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY(`SensorID`) REFERENCES "Sensors" ( ID )
)
CREATE TABLE "Datastations" (
        `ID`         INTEGER,
        `Area`       TEXT NOT NULL,
        PRIMARY KEY(ID)
)
CREATE TABLE "Sensors" (
        `ID`         INTEGER,
        `DatastationID`          INTEGER NOT NULL,
        `SensortypeID`           INTEGER NOT NULL,
        PRIMARY KEY(ID),
        FOREIGN KEY(`DatastationID`) REFERENCES "Datastations" ( ID ),
        FOREIGN KEY(`SensortypeID`) REFERENCES "Sensortype" ( ID )
)
CREATE TABLE "Sensortype" (
        `ID`         INTEGER,
        `Name`       TEXT NOT NULL,
        `Unit`       TEXT NOT NULL,
        PRIMARY KEY(ID)
)
```

Abbildung 9: DB Schema mit SQL

4.3.2 Schnittstellen

Der Applikationsserver stellt vier Schnittstellen bereit welche über REST-Endpoints genutzt werden können. Im Detail handelt es sich um zwei GET- und zwei POST-Endpunkte die mit JavaScript programmiert wurden:

- **GET**: "/allDatastations"

  Dieser Endpunkt gibt alle Datenstationen der Datenbank im JSON-Format zurück. Im Prinzip wird der gesamte Inhalt der Tabelle Datastations ausgegeben. Im Folgenden der JavaScript-Code:

Implementierungscode allDatastations-Endpunkt

```
1  //Send all Datastations
2  app.get('/allDatastations', function(req, res){
3
4     var sqlite3 = require('sqlite3').verbose();
5     var db = new sqlite3.Database('database.db');
6
7     var resultObj = [];
8     db.each("SELECT * FROM Datastations", function (err, row){
9        resultObj.push(row);
10    },
11    function(err, numberOfRows){
12       res.send(JSON.stringify(resultObj));
13    });
14 })
```

  Das zurückgelieferte JSON-Objekt sieht beispielsweise so aus:

Rückgabebeispiel allDatastations-Endpunkt

```
1  [
2     {
3     "ID": 100,
4     "Area": "RobWG"
5     }, {
6     "ID": 101,
7     "Area": "RobWG2"
8     }
9  ]
```

- **GET: "/allSensortypes":**

  Dieser Endpunkt sendet den gesamten Inhalt der Tabelle Sensortypes im JSON-Format zurück. Es werden also alle Sensortypen mit Name und Einheit geliefert. Im Folgenden wieder der JavaScript-Code:

Implementierungscode allSensortypes-Endpunkt

```
1  //Send all Sensortypes
2  app.get('/allSensortypes', function(req, res){
3
4     var sqlite3 = require('sqlite3').verbose();
5     var db = new sqlite3.Database('database.db');
6
7     var resultObj = [];
8     db.each("SELECT * FROM Sensortype", function (err, row){
9        resultObj.push(row);
10    },
11    function(err, numberOfRows){
12       res.send(JSON.stringify(resultObj));
13    });
14
15 })
```

  Nachfolgend wieder ein beispielhaftes JSON-Objekt:

Rückgabebeispiel allSensortypes-Endpunkt

```
1  [
2     {
3     "ID": 1,
4     "Name": "temperature",
5     "Unit": "C"
6     }, {
7     "ID": 2,
8     "Name": "pressure",
9     "Unit": "hPa"
10    }
11 ]
```

- **POST**: “/query“:

  Durch diesen Endpunkt kann eine Datenbank-Abfrage in Form eines JSON-Objektes an die Datenbank gestellt werden. Das Abfrageergebnis wird ebenfalls als JSON-Objekt zurück geliefert. Das dem Endpunkt übergebene Objekt enthält folgende Properties:

  - **DatastationID**
  - **SensortypeID**
  - **StartDate**
  - **EndDate**
  - **Aggregation**

  Die Datumsangaben *StartDate* und *EndDate* werden im Format *YYYY-MM-DD HH:mm:ss* angegeben. Für die zeitliche Aggregation der abgefragten Messwerte stehen die Properties "*Y*" (für Jahr), "*m*" (für Monat), "*d*" (für Tag), “*H*" (für Stunde), "*M*" (für Minute) oder "*S*" (für Millisekunden) zur Verfügung.

  Da der Implementierungscode dieses Endpunktes etwas umfangreicher ist, ist dieser nicht in der Dokumentation enthalten. Um den Code trotzdem einsehen zu können wird an dieser Stelle auf die JavaScript-Datei im GitHub-Repository[10] dieses Projektes verwiesen.

  Nachfolgend ein beispielhaftes Abfrageergebnis im JSON-Format:

Rückgabebeispiel query-Endpunkt

```
[
   {
      "ID": 100,
      "Average": 22.554999999999996,
      "Minimum": 22.437,
      "Maximum": 22.687,
      "Timestamp": "2015-05-09 00:09:02"
   }, {
      "ID": 100,
      "Average": 22.418466666666657,
      "Minimum": 22.25,
      "Maximum": 22.562,
      "Timestamp": "2015-05-09 01:06:03"
   }, {
      "ID": 100,
      "Average": 22.246566666666656,
      "Minimum": 22.187,
      "Maximum": 22.375,
      "Timestamp": "2015-05-09 02:04:02"
   }
]
```

[10] https://github.com/r0bn/raspberry-data-station/blob/master/applicationserver/db.js

- **POST: "/insert":**

  Über diesen Endpunkt können neue Daten in die Datenbank eingefügt werden. Die einzufügenden Daten werden der Schnittstelle als JSON-Objekt übergeben. Das Objekt enthält die Properties "DatastationID", "Timestamp", "Value", "Sensortype", "Area", "Unit". Alle Variablen sind vom Typ String. Der Timestamp-String wird im Format YYYY-MM-DD HH:mm:ss angegeben. Bei einem erfolgreichen Einfügen der Daten in die Datenbank wird der HTTP-Statuscode 200 ansonsten 500 zurück geliefert. Im Folgenden ist ein beispielhaftes Übergabeobjekt zu sehen.

Beispielhaftes Übergabeobjekt insert-Endpunkt

```
1    {
2       "Area": "RobWG2",
3       "Timestamp": "2015-06-05 08:35:07",
4       "Value": "984",
5       "Sensortype": "pressure",
6       "DatastationID": "101",
7       "Unit": "hPa"
8    }
```

Da der Implementierungscode zu diesem Endpunkt ebenfalls sehr umfangeich ist, erfolgt an dieser Stelle wieder ein Verweis auf die JavaScript-Datei[11] im GitHub-Repository.

4.3.3 Entwicklungsumgebung

Für die Entwicklung des gesamten JavaScript-Codes wurde das Programm *TextWrangler*[12] genutzt. Um die vier Endpunkte zu testen wurde das Chrome-Plugin *Postman - REST Client*[13] verwendet. Für die Erstellung des Datenbank-Schemas und die Administration der Datenbank wurde das Open-Source-Tool *DB Browser for SQLite*[14] genutzt. Eine spezielle Einrichtung der einzelnen Tools war nicht notwendig.

[11] https://github.com/r0bn/raspberry-data-station/blob/master/applicationserver/db.js
[12] https://itunes.apple.com/de/app/textwrangler/id404010395?mt=1
[13] https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=de
[14] http://sqlitebrowser.org/

4.4 Datenstation

Die Datenstation erfasst durch den Einsatz verschiedener Sensoren Umgebungsdaten und sendet diese an den Applikaitonsserver. Für das Projekt wurden 2 Raspberry Pi's[15] als Datenstaionen eingesetzt.

4.4.1 Umsetzung

Für die Programmierung wurde die Programmiersprache *Python*[16] verwendet. Der Vorteil dieser Sprache ist, dass es sich um eine interpretierbare Sprache handelt die zur Laufzeit ausgeführt und nicht kompiliert wird. Außerdem ist es die bevorzugte Sprache auf dem Raspberry Pi und verfügt somit über eine umfangreiche Unterstützung für Sensoren und die *GPIO* Leiste.

Um den Raspberry Pi nutzen zu können muss zunächst ein Betriebssystem installiert werden. Dafür wird eine SD-Karte verwendet auf welche ein Betriebssysteminstallationsprogramm kopiert wird. Die SD-Karte wird daraufhin in den SD-Kartensteckplatz des Pi's gesteckt und das Betriebssystem installiert.
Nach der Betriebssysteminstallation wird ein Ethernet-Kabel an den Netzwerkanschluss angeschlossen, damit nach dem Start der Datenstation über IP-Adresse über DHCP zugewiesen wird. Daraufhin ist der Pi netzwerkfähig und es ist möglich sich von einem Desktop-PC aus über eine ssh-Verbindung am Betriebssystem des Pi anzumelden.
Für die Nutzung des Wlan's muss lediglich der Wlan-Stick mit einem USB Port verbunden werden. Danach muss das entsprechende Wlan-Netzwerk in der folgenden Datei eingetragen werden:

/etc/wpa_supplicant/wpa_supplicant.conf

Die Abbildung 10 zeigt das Raspberry Pi mit angeschlossenem Wlan-Stick.

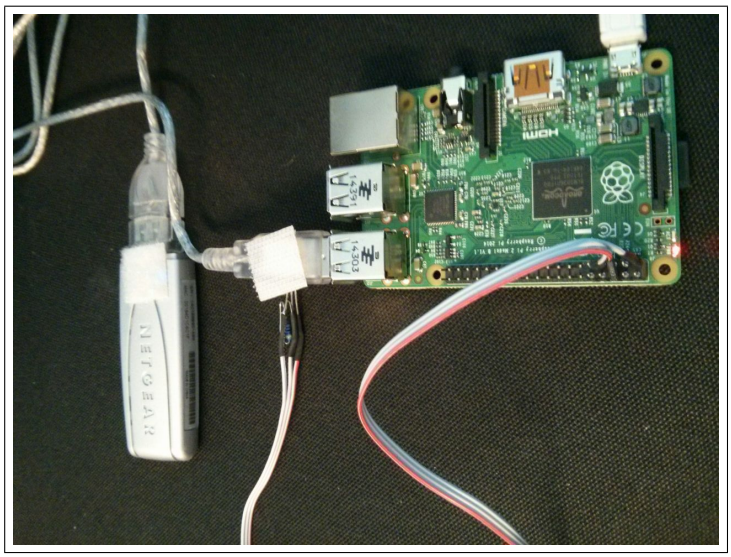

Abbildung 10: Wlan-fähiger Raspberry Pi

[15] https://www.raspberrypi.org/
[16] https://www.python.org/

Um die angeschlossenen Sensoren auslesen zu können, muss die GPIO Leiste auf dem Raspberry PI für Python verfügbar gemacht werden.Dafür müssen folgende Konfigurationsschritte durchgeführt werden:

- Kernel Module laden:

```
sudo modprobe wire
sudo modprobe w1_gpio
sudo modprobe w1_therm
```

- Kernel Module dauerhaft über autostart verfügbar machen:

```
sudo vi /etc/modules
wire
w1_gpio
w1_therm
```

Im Folgenden werden nun alle während dem Projekt eingesetzten Sensortypen kurz erläutert.

Temperatur Sensor DS1820

Der Temperatursensor stand nach der Konfiguration als eine auslesbare Datei zur Verfügung. Der Verzeichnispfad lautet:

`(/sys/bus/w1/devices/10-000802b56552/w1_slave)`

Luftdruck Sensor BCM2835

Der Luftdruck-Sensor wird über das Xtrinsic-Sense-Board[17] angesprochen. Für dieses Sensor Board existieren[18] mehrere Bibliotheken und Beispiele speziell für den Raspberry Pi. Da die verfügbaren Beispiele in Python geschrieben sind und vorkompilierte C-Bibliotheken verwenden, konnten diese sehr einfach in das bestehende Programm integriert werden. In der Abbildung 11 ist der Raspberry Pi mit angeschlossenem Luftdruck-Sensor zu sehen.

Feuchtigkeits Sensor DHT11

Dieser Sensor wurde für dieses Projekt ohne Basisboard geliefert und musste deswegen manuell verkabelt werden. Es konnte ein erfolgreicher Auslesevorgang vorgenommen werden. Die Abbildung 12 zeigt den verkabelten Sensor.

[17] http://de.farnell.com/xtrinsic-sensor-board
[18] ttp://www.raspberrypi-spy.co.uk/2014/06/farnell-xtrinsic-mems-sensor-board-for-raspberry-pi/

Abbildung 11: Raspberry Pi mit Luftdruck Sensor

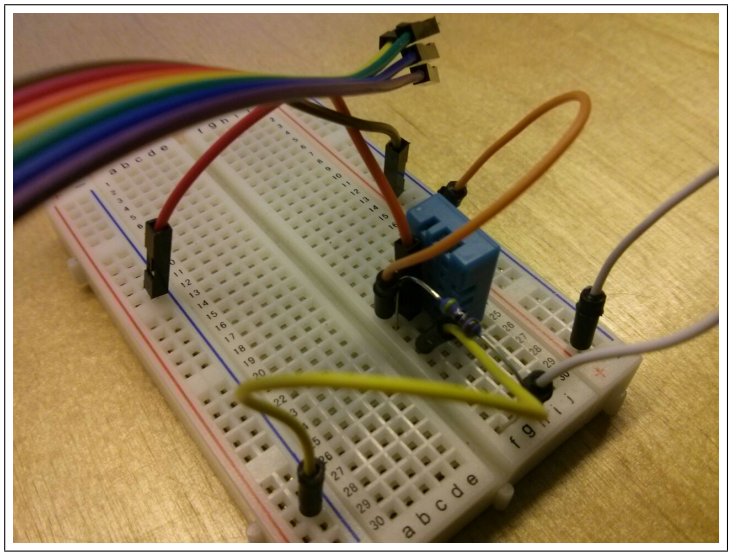

Abbildung 12: Feuchtigkeitssensor

Helligkeits Sensor TSL45315

Der TSL45315 wird hautpsächlich in Arduino[19]-Umgebungen eingesetzt. Deshalb gibt es keine funktionierende Beispiel-Implementierung für den Raspberry Pi. Leider konnte der Helligkeitssensor nicht ausgelesen werden.
Die Abbildungung 13 zeigt den eingesetzten Helligkeitssensor.

[19] http://www.arduino.cc/

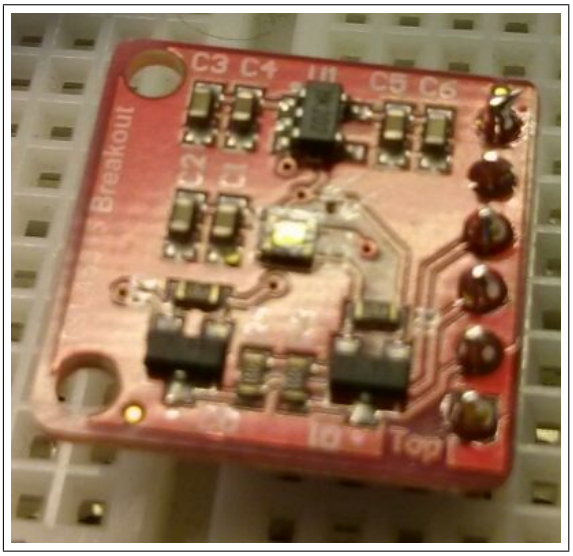

Abbildung 13: Helligkeitssensor

4.4.2 Software

Die Softwarelösung auf den Datenstationen sollte leicht installierbar und gut erweiterbar sein. Deshalb wurde als Programmiersprache Python gewählt, da es sich hierbei um eine interpretierbare Sprache handelt.
Jeder Sensor wird als ein Python-Modul implementiert und dann von der Hauptdatei *programm.py* importiert. Die programm.py wird durch mehrere Übergabeparameter konfiguriert:

```
$ python programm.py [Datenstation Id] [Sensor Typ] [area]
```
[20]

Diese Variante ermöglicht einen flexiblen Einsatz von einzelnen oder meheren Sensoren an einer Datenstation.

Um die Einlese-Periode festzulegen wird empfohlen ein simples Shell-Script anzulegen das die *programm.py* mit den entsprechenden Parametern aufruft. Falls mehrere Sensoren ausgelesen werden sollen, wird zwischen den einzelnen Sensoren eine Pause von mindestens 2 Sekunden benötigt! Dies kann mithilfe des Shell-Befehls *sleep* erreicht werden. Grund dafür sind mögliche Kollisionen unter den einzelnen Sensoren. Zum Beispiel auf dem I^2C Bus. Aber auch der exklusive Zugriff auf bestimmte Auslese-Funktionen. Außerdem kann es auf dem Applikationsserver zu Verarbeitungsproblemen kommen wenn zu viele Anfragen nahzu zeitgleich eingehen.

Das Shell-Script kann dann per *Cron-Job* mit einer Wiederholungszeit von beispielsweise einer Minute konfiguriert werden. Es empfiehlt sich die Ausführung entweder auf dem Applikationsserver zu überwachen oder direkt auf der Datenstation mit folgenden Kommando:

```
tail -f /var/log/syslog
```

[20] Detaillierte Informationen zur Schnittstelle zwischen den Datenstationen und dem Applikationsserver sind bei der Beschreibung des insert-Endpunktes auf Seite 19 zu finden.

4.4.3 Entwicklungsumgebung

Als Entwicklungsumgebung wurde *GVIM*[21] verwendet. Dabei handelt es sich um eine grafische Version des Linux-Texteditors *vi*.Der Zugriff auf die Datenstation und die Konfiguration des Raspberry Pi wurde über eine ssh-Verbindung vorgenommen.

[21] http://www.vim.org/

5 Projektjournal

Während der Laufzeit des Projektes wurde eine Google-Sheet-Datei für die Erfassung der Aufwände der einzelnen Teammitglieder genutzt. Die Datei liegt im Projektverzeichnis in Google-Drive. Der Übersichtlichkeit halber wurde die dortige Tabelle nicht direkt in diese Dokumentation übernommen, sondern es wird an dieser Stelle auf die Excel-Datei verwiesen:

Projektjournal[1]

[1]Falls Probleme mit der Datei auftreten kann das Journal auch mit folgendem Link im Internet eingesehen werden: Journalverweis

6 Projektfazit und Ausblick

Das Projekt konnte erfolgreich umgesetzt und die Projektziele erreicht werden. Die Teammitglieder konnten Kenntnisse in der Webentwicklung, Datenbankkonzeption und im Bereich des Raspberry Pie erwerben und vertiefen. Desweiteren konnte ein Einblick in die Projektarbeit und deren Kommunikationsabläufe gewonnen werden.

Für die weitere Verwendung der Projektergebnisse sind nachfolgend einige Ideen aufgelistet.

- An die Datenstationen können weitere Sensoren (wie beispielsweise Helligkeitssenoren) angeschlossen werden.
- Für einen Produktiveinsatz ist noch die Entwicklung eines Fehlermonitorings im Falle eines Sensorausfalls erstrebenswert.
- Für einen Einsatz in einem größeren Umfang, wie beispielsweise an der HfT ist ein größerer Abstraktionsgrad sinnvoll. Dafür müsste eine hierarchische Struktur implementiert werden (Gebäude - Stockwerk - Raum).
- Wenn zusätzlich zwei- oder dreidimensionale Positionsdaten zur Verfügung stehen würden, könnte man die Sensordaten in einem Raum pro Zeitverlauf darstellen.

Eine weitere Verwendung der Projektergebnisse im Rahmen von EMAS wäre eine hervorragende Möglichkeit das bestehende System zu optimieren und weitere Ideen zu verwirklichen.

Abbildungsverzeichnis

Tabellenverzeichnis