



Seminararbeit

Information Management Automotive

Seminar Performance Management

Prototyperstellung auf Grundlage des IoT-Szenarios „Mensa“

Betreuende Dozentin: Prof. Dr. Dany Meyer

Verfasst von: Laslo Harry Welz (Matrikel-Nr. 247329)

Abgabetermin: 23.01.2020

Inhaltsverzeichnis

1	Abstract	2
2	Einleitung	2
2.1	Motivation	2
2.2	Methodik	3
2.3	IoT-Architektur	3
3	Hauptteil	6
3.1	Kommunikationsprotokolle	6
3.1.1	REST API	6
3.1.2	PHP	7
3.1.3	MQTT	9
3.2	Datenpersistenz	11
3.2.1	MySQL-Datenbank	11
3.2.2	NoSQL-Datenbank	12
3.2.2.1	MongoDB	12
3.2.2.2	Cloud Firestore	13
3.3	Entwurfsunterlage Prototyp	15
3.4	Vorgehensweise	16
3.4.1	Android App	16
3.4.1.1	Benutzeroberfläche	16
3.4.1.2	MQTT-Client	17
3.4.2	Python	20
3.4.2.1	MQTT-Client	20
3.4.2.2	Datenbank-Connector	21
3.4.3	MySQL-Datenbank	23
3.4.4	Grafana-Dashboard	24
4	Fazit	25
5	Literaturverzeichnis	26

1 Abstract

Das Internet der Dinge (IoT) verkörpert eine weltweite Vernetzung von intelligenten Objekten. Mit der steigenden Anzahl von miteinander kommunizierenden Objekten im Internet wächst auch das Volumen der Daten, die diese produzieren. Insbesondere Firmen sehen es als Herausforderung, diese Daten zu sammeln, verarbeiten und zu analysieren. Einerseits um interne Prozesse zu verbessern und um Kosten zu sparen. Andererseits um die flexiblen Bedürfnisse ihrer Nutzer/Kunden vorherzusagen und so speziell angepasste Produkte anbieten zu können. Jedoch liegen die vielen Datenquellen nicht strukturiert vor, wodurch das Sammeln dieser Daten mit einer hohen Komplexität verbunden ist und in unvollständigen und ungenauen Analysen resultieren kann. Um Verfügbarkeit und Qualität zu garantieren, benötigt man schnelle und sichere Kommunikationsprotokolle, sowie konsistente Datenspeicherung. In dieser Arbeit werden verschiedene Ansätze zu diesen beiden Kriterien untersucht und ein Architektur-Prototyp entwickelt, der gesammelte heterogene Daten in einem Dashboard visualisiert.

2 Einleitung

2.1 Motivation

Heutzutage können mit Mobilgeräten und Sensoren viele Informationen gesammelt und ausgewertet werden. Am anderen Ende kann ein Akteur sitzen, der bei einer bestimmten Information seinen Zustand ändert, oder ein KI-gesteuerter Algorithmus, der aus den vorhandenen Daten eine Vorhersage trifft. Die Ergebnisse sind interessant für BI-Analysen und werden vorzugsweise in Tabellen oder Diagrammen dargestellt.

Das Ziel der Seminararbeit ist es, verschiedene Datenquellen in der Mensa und Cafeteria an der Hochschule Neu-Ulm zu beschreiben und in eine IoT-Architektur zu implementieren. Zusätzlich soll ein Entwurf für eine mögliche Architektur entworfen und getestet werden. Am Ende der Arbeit, soll eine klare Prototyp-Dokumentation mit der Beantwortung folgender Forschungsfrage dargestellt werden:

Kann für das IoT-Szenario „Mensa“ ein Prototyp erstellt werden?

Die Funktionalität des Prototyps wird in einer Live-Demo präsentiert.

2.2 Methodik

Der Hauptteil der Seminararbeit lässt sich in zwei Domäne unterteilen:

- I) Recherche und Evaluation geeigneter Kommunikationsprotokolle und Datenbanken
- II) Prototypentwurf und Dokumentation

Als Erstes wird anhand einer Literaturrecherche Informationen zu IoT-Architekturen gesammelt. Es soll ein Status Quo der derzeitigen Architekturen bestimmt werden. Danach werden verschiedene Ansätze für eine IoT-Anwendung, die in diesem Projekt erstellt werden sollen, betrachtet. Ziel war es, verschiedene Komponenten der Datenübermittlung und der Datenspeicherung zu vergleichen. Sechs Komponenten werden in den Kriterien Aufbau, Schnittstellen und Kompatibilität miteinander verglichen und evaluiert. Nach der Evaluation wird der passende Ansatz in einen Prototyp umgesetzt. Dazu wird die bestehende Entwurfsskizze in eine validierte Entwurfsunterlage spezifiziert. Der Prototyp wird schrittweise aufgebaut. Mittels einer Android Applikation sollen Daten erfasst werden. Dafür werden einfache Texteingaben erstellt, welche die eingegebenen Werte in Variablen abspeichern. Um die Kommunikation zwischen den einzelnen Komponenten zu testen werden Ausgaben über die jeweilige IDE-Konsolen generiert. Für die Datenspeicherung wird ein passendes Datenbankschema erstellt. Damit die gespeicherten Daten analysiert und in verschiedenen Abfragen dargestellt werden können, werden im letzten Schritt zwei Web-basierte Dashboards erstellt.

2.3 IoT-Architektur

Das Internet der Dinge findet in vielen Bereichen, von Gesundheitstracking, über „Smart Home“- Applikationen bis hin zu industrieller Automatisierung, eine Anwendung. Eine Vielzahl von heterogenen Objekten kann in einer flexiblen und schichtartigen Architektur aufgenommen werden [1]. IoT als Konzept kann nicht einheitlich interpretiert werden und kann durch die Anzahl der Anwendungen nicht genau definiert werden [2]. Es gibt mehrere Ansätze zu einer Architektur, welche aber nicht als allgemeingültiges Modell anerkannt werden [1]. Im Jahr 2013 hatte die europäische Kommission ein Projekt in Auftrag gegeben, welches die Erarbeitung eines architektonischen Referenzmodell beabsichtigte. Mit einer Analyse der gegebenen Bedürfnisse in Forschung und Industrie, wurde mit Partnern aus dem wirtschaftlichen Sektor das Projekt IoT-A gestartet [3]. Die Entwicklung einer neuen

Architektur ist fortlaufend, um größere Kapazitäten aufzunehmen und Herausforderungen wie Skalierbarkeit, „Quality of Service“ (QoS) und Daten-/Nutzersicherheit, anzusprechen [1]. Aus den vorgeschlagenen Modellen werden vier spezifische Architekturen abgeleitet, die aus mehreren Funktions-Schichten bestehen. Das Basis-Modell besteht aus drei Schichten, welche in die (i) Wahrnehmungsschicht („Perception“), (ii) die Netzwerkschicht und (iii) die Applikationsschicht unterteilt werden kann [2]. Um die Architektur weiter zu abstrahieren, wird dieses Modell auch als vier- oder fünf-schichtiges Modell dargestellt. Das Fünf-Schichten Modell wird in Abb.1 [4] dargestellt. Als SOA („Service Oriented Architecture“) -basierte IoT-Architektur wird die Implementierung einer IoT-Middleware ermöglicht. Die Middleware dient als Schnittstelle zwischen Technologie- und Applikationsschicht [2]. Die Autoren in [5] beschreiben die Konzipierung einer Middleware, die speziell für mobile IoT-Endgeräte entwickelt wurde.

Um große Datenmengen verarbeiten und speichern zu können, benötigt man viele Computing-Ressourcen. Dabei kommt konventionelle Hard- und Software an ihre Grenzen. Ein Lösungsansatz ist Cloud Computing. Ressourcen (z.B. Server, Datenbanken, Analysetools) werden dabei über das Internet bereitgestellt, um schnell IoT-bezogene Daten zu verarbeiten [6]. Somit können IoT-Anwendungen über Cloud-Services von Drittanbietern laufen, wozu keine eigene IT-Infrastruktur nötig ist [4]. Entlang der beschriebenen IoT-Architekturschichten kann Cloud Computing in zwei weiteren Abstraktionen erweitert werden (Abb.2). Fog und Edge Computing dienen als Verknüpfung für smarte Geräte und Cloud Services. Die Idee dahinter ist die Verschiebung der Ressourcen von einer zentralen Cloud in die mobilen Rand („Edge“) -Applikationen des „Internet of Things“. Durch die Verarbeitung vor Ort (z.B. lokale Datenspeicherung mit SQLite) bieten Services eine verbesserte Performance. Edge Computing wird im IoT eine wichtige Rolle spielen, sobald Aspekte wie Sicherheit, Zuverlässigkeit und Flexibilität standardisiert werden [4].

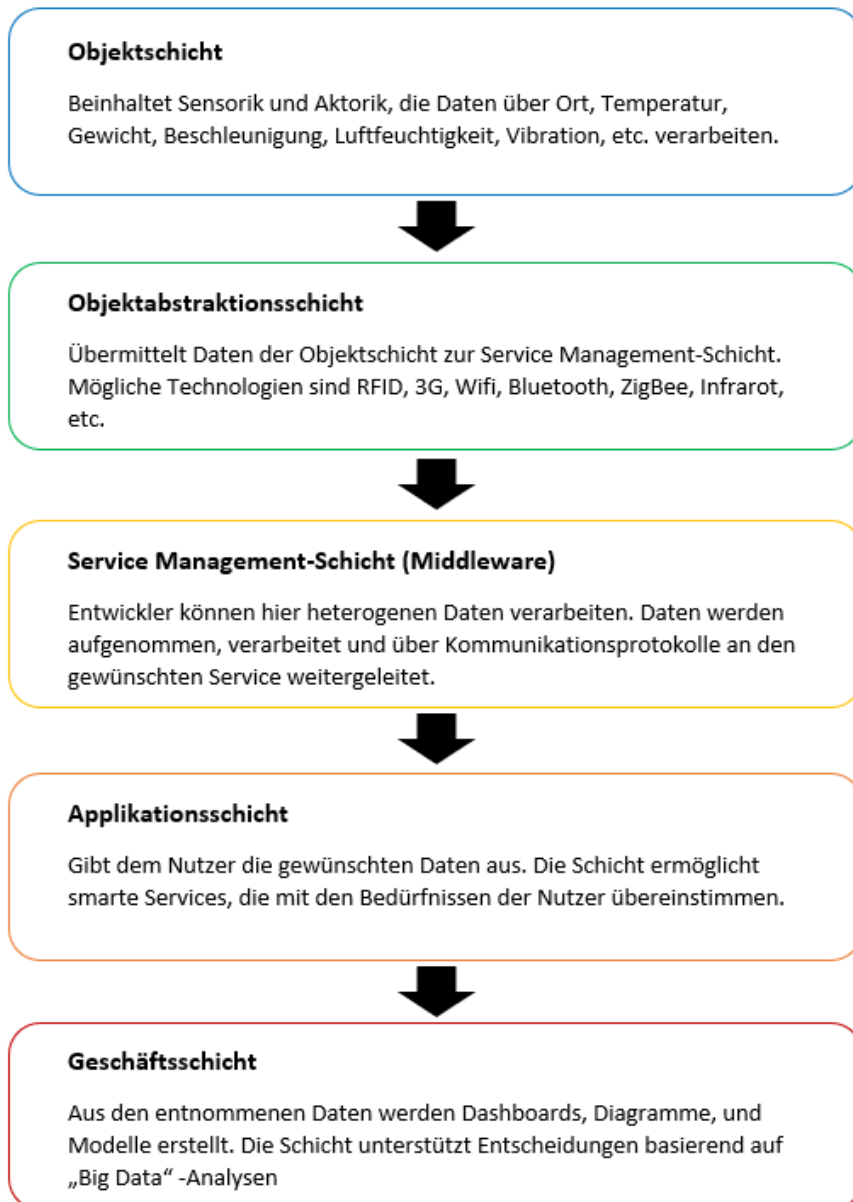


Abbildung 1: Fünf Schichten IoT-Architektur

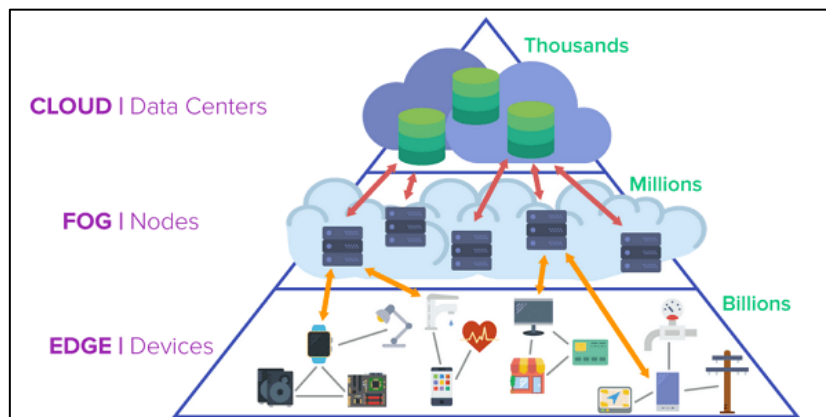


Abbildung 2: Cloud Computing im IoT (<https://innovative-trends.de/2018/01/02/cloud-computing-edge-computing-und-fog-computing-unterschiede-kurz-erklart/>)

3 Hauptteil

3.1 Kommunikationsprotokolle

3.1.1 REST API

RESTful (“Representational State Transfer“) Web Services nutzen Internetschnittstellen (API), um über das Netz mit dem Hypertext Transfer Protokoll (HTTP) zu kommunizieren und Daten im JSON-Format zu übermitteln [7]. Diese können mit anderen Web Services oder Datenbanken interagieren. Der Client übermittelt eine Anfrage (Request) an den Web Service und im Gegenzug antwortet dieser mit einer Antwort, bzw. mit einem „Response-Code“. Die vier HTTP-Request Methoden, die REST implementiert, sind (i) GET – Client bekommt Daten mit der Antwort ausgegeben, (ii) POST – Daten werden an Web Service übermittelt, (iii) PUT – Bestimmte Datensätze werden überschrieben und (iv) DELETE – Daten werden gelöscht. Die Entwicklungsplattform „POSTMAN“ ist dafür geeignet, diese Schnittstellen mit dieser Methodik zu testen. Der Benutzer wählt eine Methode, sendet eine Anfrage als URL an eine Internetadresse und kann bei erfolgreicher Übermittlung die Antwort, oder den erzeugten Code anzeigen lassen. Damit lässt sich, ohne zu programmieren, schnell überprüfen, ob der Request den gewünschten Web Service anfragen kann.

In Android Studio kann man mittels der „Retrofit“-Bibliothek auf REST Web Services zugreifen. Dabei wird die HTTP API in eine Call-Interface-Methode umgewandelt, die in der MainActivity mit einem Objekt der Retrofit-Klasse aufgerufen wird (Abb. 3). Je nach Anwendungsfall kann das Retrofit-Instanz auch als Singleton deklariert werden. Die Call-Methode kann im Request Objekte oder eine Liste von Objekten verarbeiten. Eingegebene Daten werden mit einem Konverter in JSON umformatiert. Das erzeugte Objekt kann so direkt in einer NoSQL-Datenbank (MongoDB) überschrieben werden. Die Annotation im Interface legt die Methode und den Pfad der angegebenen URL fest (Abb. 4). Zusätzlich können Header festgelegt werden, die beispielsweise Parameter (Keys) für die Authentifizierung enthalten. Über das Interface können mehrere Methoden laufen, was die Benutzbarkeit von CRUD („Create, Read, Update, Delete) -Operationen, deutlich vereinfacht.

```
public interface SwApi {  
  
    @Headers("x-apikey:f5580327e027da3a9d2a140a440e3606991ad")  
    @POST("rest/cars")  
    Call<Cars> createCar(@Body Cars cars);  
  
    @Headers("x-apikey:f5580327e027da3a9d2a140a440e3606991ad")  
    @GET("rest/cars")  
    Call<List<Cars>> getCars();  
}
```

Abbildung 3: REST Interface

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://randomapp-85ec.restdb.io/")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();  
  
restDb = retrofit.create(SwApi.class);
```

Abbildung 4: Retrofit-Objekt in Android Studio

3.1.2 PHP

Um die Android Applikation aus dem vorherigen Absatz mit einer MySQL-Datenbank auf einem Webserver kommunizieren zu lassen, wird der REST Web Service mit einer weiteren Komponente erweitert. PHP ist eine weit verbreitete Open-Source Skriptsprache in der Webentwicklung, die auf verschiedenen Plattformen und Betriebssystemen benutzt werden kann [8]. Das Hauptfeld von PHP liegt in der serverseitigen Programmierung, weshalb PHP-Skripte meist über einen Webserver laufen und über einen Webbrowser ausgegeben werden können [8]. Abb. 5 zeigt eine Methodik für die Datenübertragung von einer Android Applikation mit einem Webserver, der eine MySQL-Datenbank hostet. Hierbei übermittelt der Retrofit-Client den POST-Request an ein PHP-Skript, welches sich, zur einfachen Darstellung, auf dem gleichen (Localhost-) Webserver befindet, z.B. Xampp Apache Webserver. Das PHP-Skript dient hierbei als Connection-Handler für die MySQL-Datenbank, welche in der ebenfalls php-basierten Webanwendung „PHPMyAdmin“ implementiert ist. Das Skript (Abb.6) prüft die eingehende REST-Methode und extrahiert, bei Übereinstimmung, die Werte aus den mitgegebenen Variablen. Danach wird eine Datenbank-Verbindung über ein externes PHP-Skript (Abb.7) hergestellt. Eine SQL-Query übermittelt die Daten an die angegebenen Felder in der Tabelle. Sollte der Code keine Fehlermeldung produzieren, wird in der HTTP-Response ein JSON mit einem definierten Wert zurückgegeben. Bevor man dies in der Android IDE („Integrated Development Enviroment“) implementiert, kann dies ebenfalls mit der Postman-Applikation getestet werden. Es gilt zu beachten das der POST-Request als URLEncoded deklariert werden muss. Damit können bestimmte Zeichen einer URL in die ASCII %-Darstellung umgewandelt werden.

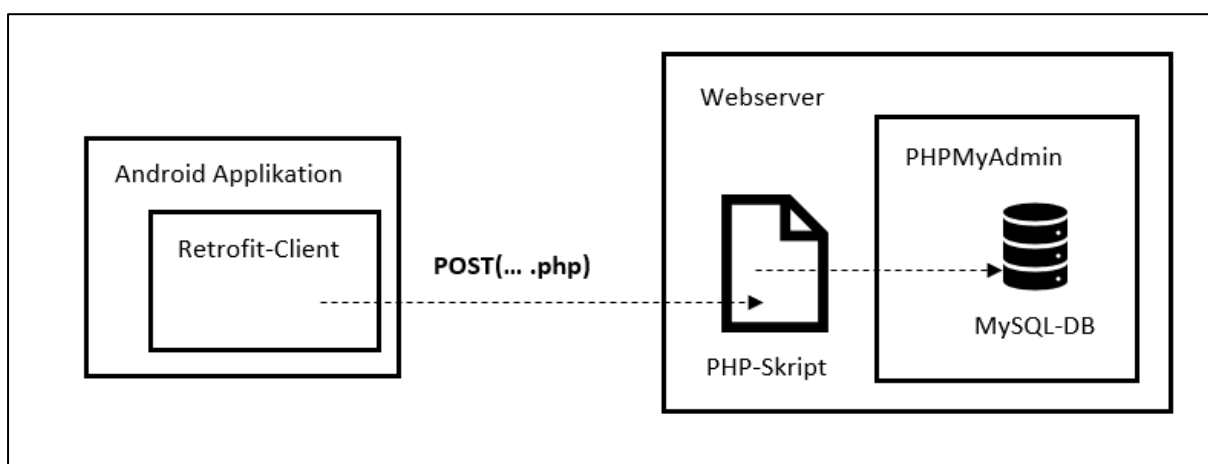


Abbildung 5: Entwurfsskizze REST-Webservice mit PHP und MySQL


```

1  <?php
2
3  if($_SERVER['REQUEST_METHOD'] == "POST") {
4
5      $title = @$_POST['title'];
6      $device = @$_POST['device'];
7
8
9      require_once("connectlocal.php");
10
11      $query = "INSERT INTO `mydata` (title, device) VALUES ('$title', '$device')";
12
13      if (mysqli_query($con, $query)) {
14          $response['success'] = true;
15          $response['message'] = "Successfully";
16
17      } else {
18          $response['success'] = false;
19          $response['message'] = "Failed";
20      }
21  }
22
23
24  } else {
25      $response['success'] = false;
26      $response['message'] = "Error!";
27  }
28
29  echo json_encode($response);
30
31  >

```

Abbildung 6: SQL-Abfrage in PHP

```

/public_html/connectlocal.php
1  <?php
2
3  $con = mysqli_connect("localhost", "id11860771_laslo", "imapro1", "id11860771_android");
4
5  if ($con) {
6      echo "success";
7  } else {
8      echo "error";
9  }

```

Abbildung 7: Datenbankverbindung mit PHP

3.1.3 MQTT

Das „Message Queuing Telemetry Transport“-Protokoll ist speziell auf IoT-Anwendungen zugeschnitten. Es ermöglicht M2M (Machine-to-Machine)-Kommunikation, welche Event-basiert abläuft [9]. Der Datenaustausch läuft über das Publish/Subscribe-Prinzip ab. Dabei fungiert ein Teilnehmer als Client und ein anderer (meist ein Webserver) als Broker. Ein Broker ist meist mit mehreren Clients verbunden und verwaltet den Datenfluss über sogenannte „Topics“. Client 1 kann eine Nachricht über ein Topic an den Broker veröffentlichen („Publish“). Client 2 kann nun dieses Topic abonnieren („Subscribe“) und erhält so die Nachricht von Client 1. Beim gegenseitigen Datenaustausch treffen die Sender eine Vereinbarung, welches die Übermittlungsgarantie einer Nachricht definiert („Quality of Service“). Diese hat drei unterschiedliche Stufen (0-2) und können zwischen zwei Kommunikationspartnern unterschiedlich sein.

In Android kann so ein MQTT-Client über die HiveMQ-Bibliothek implementiert werden [10]. Der Client verwendet in diesem Ansatz den „BlockingClient“, welcher Ergebnisse sofort ausführt und den Thread anhält bis diese erhalten sind [11]. Diese Vorgehensweise eignet sich für Clients, die überwiegend Publish-Methoden aufrufen. In der ConnectToClient-Methode in Abb. 8 wird

```
public void connectToClient(View v) {
    Mqtt5BlockingClient client = Mqtt5Client.builder()
        .identifier(UUID.randomUUID().toString())
        .serverHost("broker.mqttdashboard.com")
        .buildBlocking();

    client.connect();

    client.publishWith().topic("mensaiot")
        .qos(MqttQos.AT_LEAST_ONCE)
        .payload(message.getBytes()).send();

    client.disconnect();
}
```

Abbildung 8: MQTT-Publish Methode in Android Studio

zuerst ein Objekt des Mqtt5BlockingClient (Die derzeitige Version von MQTT ist 5.0) erstellt, welche die Client-Builder-Methode aufruft. Diese definiert einen zufällig generierter „Universally Unique Identifier“ (UUID) und die Broker-Zieladresse. Das Objekt öffnet eine Verbindung zum Webserver und ruft die publishWith()-Methode auf eine bestimmtes Topic auf. Neben dem Topic enthält das Nachrichtenpaket die Übertragungsqualität („At least once“ = QoS-Level 1, eine Nachricht wird mindestens einmal gesendet) [10] und die Message-Payload (meist ein formatierter String). Danach wird die Verbindung wieder terminiert, um den Thread zu beenden. Eine Möglichkeit die Verbindung zu testen, ist die Benutzung eines Webclient in einem Browser (Abb.9). Um Daten über den Broker in einer Datenbank abzuspeichern, benötigt man einen Datenbank-Connector, der lokal oder über einen Webserver läuft. Je nach Anwendungsfall können zwei Methoden implementiert werden. Node.js ist eine serverseitige Plattform und wird in der Webentwicklung vorwiegend als Verknüpfung zwischen JavaScript und NoSQL-Datenbanken genutzt. Daten werden

überwiegend im JSON-Format übergeben, wodurch die Daten zwischen den Komponenten nicht umgewandelt werden müssen. Da in diesem Projekt Daten mittels einer relationalen SQL-Datenbank gespeichert werden, wird für diesen Schritt ein Python-Connector benutzt, der sich für die Methodik einfacher darstellen lässt. Die genaue Funktion wird in Absatz 3.4.2 erklärt.

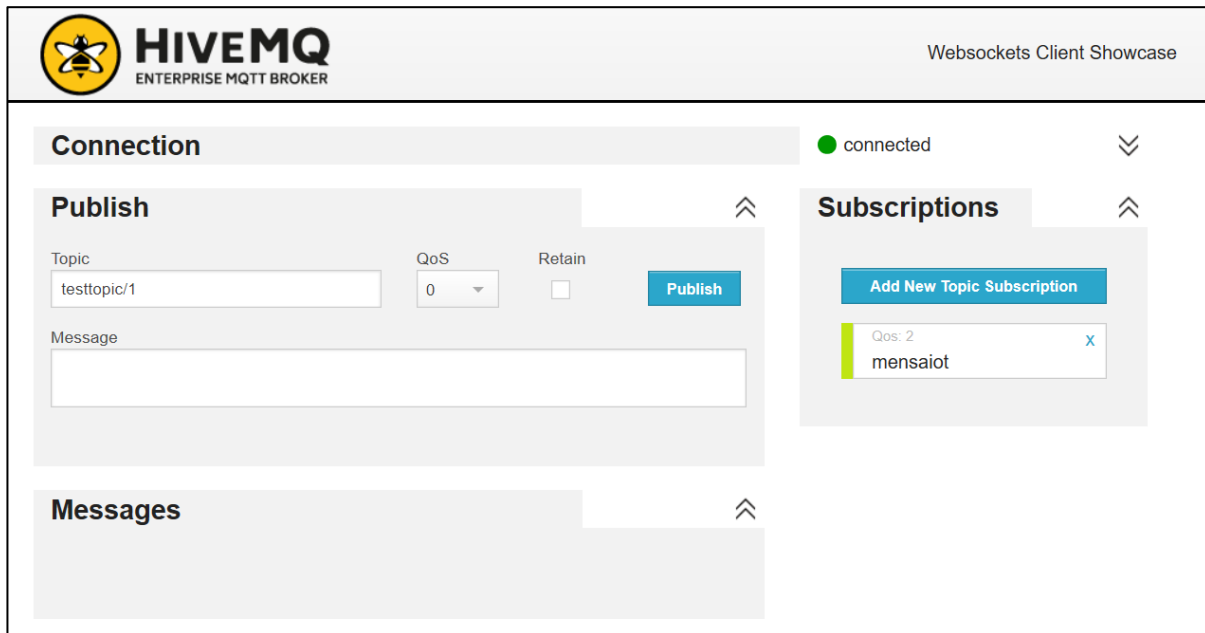


Abbildung 9: HiveMq-Webbroker Benutzeroberfläche

3.2 Datenpersistenz

3.2.1 MySQL-Datenbank

Unter den relationalen Datenbanksystemen ist das von Oracle entwickelte MySQL das meist benutzte auf dem Markt. MySQL gibt es sowohl als Open-Source Produkt, als auch als kostenpflichtige Enterprise-Version. Es wurde für die Entwicklung von webbasierenden Applikation entwickelt und wird von vielen Firmen in unterschiedlichen Branchen benutzt [12]. Daten werden über die „Structured Query Language“ (SQL) in Tabellen eingefügt und können über Abfragen

geändert, gelöscht, gefiltert oder aggregiert werden. Neben MySQL gibt es eine weitere Version namens MariaDB, die einem MySQL-Entwicklungszweig entsprungen ist. Diese teilt mit MySQL viele Gemeinsamkeiten, wie SQL-Programmierung, Konfigurationsdateien und API-Client Protokolle. Doch im Gegensatz zu MySQL weist MariaDB mehr Features auf und wird außerdem von großen Tech-Konzernen, wie Facebook oder Google, weiterentwickelt, weshalb der Marktanteil gegenüber dem herkömmlichen MySQL größer ist [12]. In Absatz 3.1.2 werden Daten von einem PHP-Skript auf einer MariaDB-basierenden Webdatenbank eingelesen (Abb. 11). MySQL nutzt Konnektoren für Java, PHP, C, Perl, Python, .NET und Ruby um eine direkte Anbindung zu ermöglichen [12]. Für Java-Applikationen gibt es den JDBC-Treiber, der als jar-Datei in die Programmierumgebung implementiert werden kann. Dieser wird aber bei Android-Applikationen, welche mit Webservern kommunizieren, nicht unterstützt, da diese aus Sicherheitsgründen keine ungeschützte direkte Verbindung zur Datenbank zulässt.

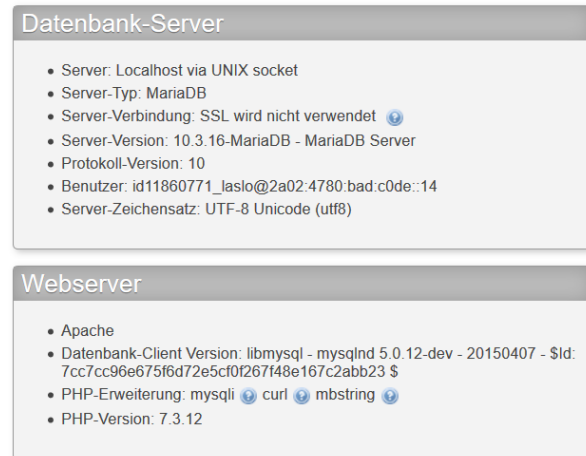


Abbildung 10: MySQL-Webserver

id	title	device	created
2	Hallo	Samsung	2019-12-08 14:46:28
3	Test	LG	2019-12-08 14:48:40
4	hallo	test	2019-12-08 17:10:42
5	Halo Weit	troiololo	2019-12-08 17:13:07
6	Halo	wow gehts	2019-12-08 17:45:21
7	Hallo	na du	2019-12-08 17:46:56
10	Tolles Essen	Samsung	2019-12-08 17:57:32
11	hallo	gruzi	2019-12-08 18:01:18
12	hallo	gruzi	2019-12-09 11:43:41
13	GG	Tolle Sache	2019-12-09 11:51:49
14	Hey	na du	2019-12-09 12:06:54
15	hallo	weit	2019-12-09 12:10:11
16	hallo	weit	2019-12-11 12:14:53
17	hallo	weit	2020-01-03 10:43:58

Abbildung 11: MySQL-Tabelle mit gespeicherten Werten

3.2.2 NoSQL-Datenbank

3.2.2.1 MongoDB

Durch ihre nicht-statische Struktur können NoSQL-Datenbanken heterogene Datenmengen weitaus effizienter verarbeiten als relationale Datenbanken [13]. Durch die ansteigende Nutzung von IoT-Applikationen, steigt auch der Bedarf an NoSQL-Lösungen [13]. Es gibt vier verschiedene Ansätze für NoSQL-Datenbanken (Abb. 12), wodurch man als Webentwickler mehrere Vorgehensweisen testen kann, seine Datenstruktur anzupassen. Diese Arbeit behandelt ausschließlich die Dokumenten-basierten Datenbank. Die meist benutzte Dokumenten-Datenbank ist MongoDB [14]. Sie wird von vielen Firmen in der Cloud-Entwicklung genutzt und speichert Daten in einem JSON-ähnlichen Dokumentenformat.

Restdb.io ist eine Open-Cloud Datenbank basierend auf einer MongoDB [15]. Als Nutzer kann man kostenfrei bis zu zwei Datenbanken erstellen. Jede Datenbank erhält eine generierte URL, um sich mit REST APIs zu verbinden. Anstelle von Tabellen

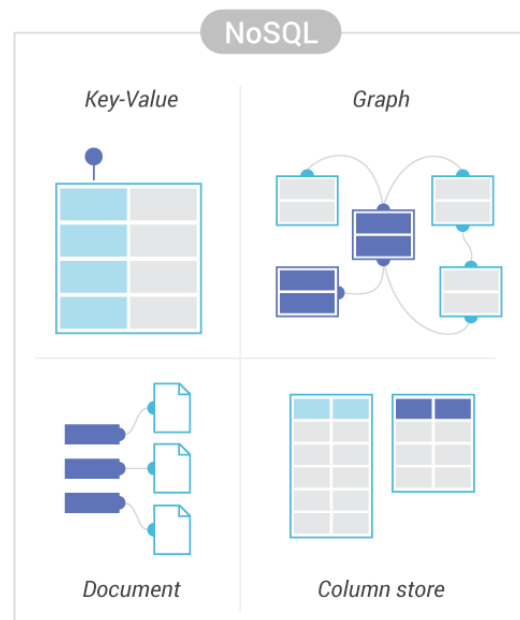


Abbildung 12: NoSQL-Datentypen (<https://1bpezptkft73xxds029zrs59-wpengine.netdna-ssl.com/wp-content/uploads/differences-between-sql-databases-and-nosql-databases.png>)

beinhalten dokumentenbasierte Datenbanken Collections. Jeder Collection können mehrere Fields enthalten, welche Variablen der Collection sind (Abb. 11). Restdb.io ist sehr benutzerfreundlich und gibt dem Nutzer vorgefertigte Verbindungsmöglichkeiten über verschiedene Treiber, wie Android, JavaScript und Swift (iOS) [15]. Ebenso ist es möglich eine Collection mit zufällig generierten Daten zu füllen, wodurch einfache Tests gemacht werden können. Dieser Ansatz referenziert auf Absatz 3.1.1, in dem Daten über einem Android REST-Client in einem JSON an die RestDB gesendet wird. Die Daten wurden über den Pfad „rest/cars“ in drei Fields übernommen.

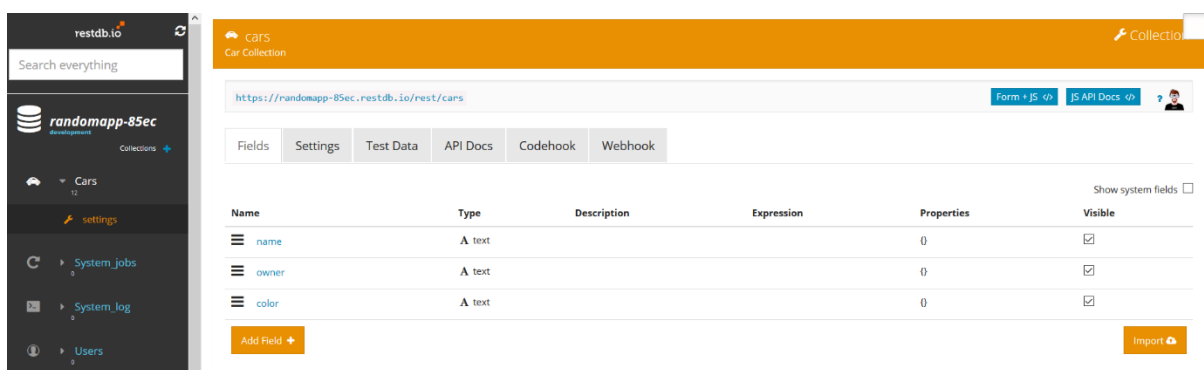


Abbildung 13: Restdb.io Collection

3.2.2.2 Cloud Firestore

Eine weitere NoSQL Cloud Datenbank ist Googles Cloud Firestore. Firestore ist Teil der App-Entwicklungsplattform Firebase und beinhaltet wichtige Webtools, wie Analytics, Vorhersage-Modelle und Hosting. Ebenso können Google-interne Dienste wie Google Ads und Google Marketing Plattform genutzt werden [16]. Der Cloud Firestore ist eine weitere Datenbank neben Googles Real-Time Database, welche nur durch eine andere Datenstruktur unterschieden werden kann. Daten in Cloud Firestore werden als Dokument in Field-Variablen gespeichert. Dokumente sind in Collections eingebettet die wie ein Verzeichnis aufgebaut sind, dabei kann ein Dokument ohne eine Collection nicht existieren. Jedes Dokument hat einen Namen, welcher manuell deklariert werden kann, oder automatisch generiert wird. Dokumente sind schemalos, was bedeutet, dass Dokumente in derselben Collection unterschiedliche Variablen besitzen können. Einerseits ermöglicht dies einen flexibleren Aufbau als bei MySQL-Datenbanken, andererseits sollten für Abfragevorgänge möglichst gleiche Strukturen vorhanden sein. Dokumente können auf ein oder mehrere „Subcollections“

referenzieren, welche ebenfalls Dokumente enthalten können, sodass eine wurzelartige Beziehungsstruktur entsteht [16]. Um ein Firebase-Projekt mit Android Studio zu verbinden, kann man dies innerhalb der IDE über das Menü „Tools -> Firebase“ in wenigen Schritten erledigen. Um das Projekt zu synchronisieren benötigt Firebase nur den Paketnamen des Android-Projekts und erstellt ein API-Key und eine Public ID. Damit der Firestore-Zugang erstellt werden kann, muss man die Abhängigkeiten in der Gradle-Datei implementieren.

Nachfolgend wird eine Methode, zur Datenbankbeschreibung dargestellt:

```
Firestore db = FirebaseFirestore.getInstance();
CollectionReference notebookRef = db.collection("Notebook");

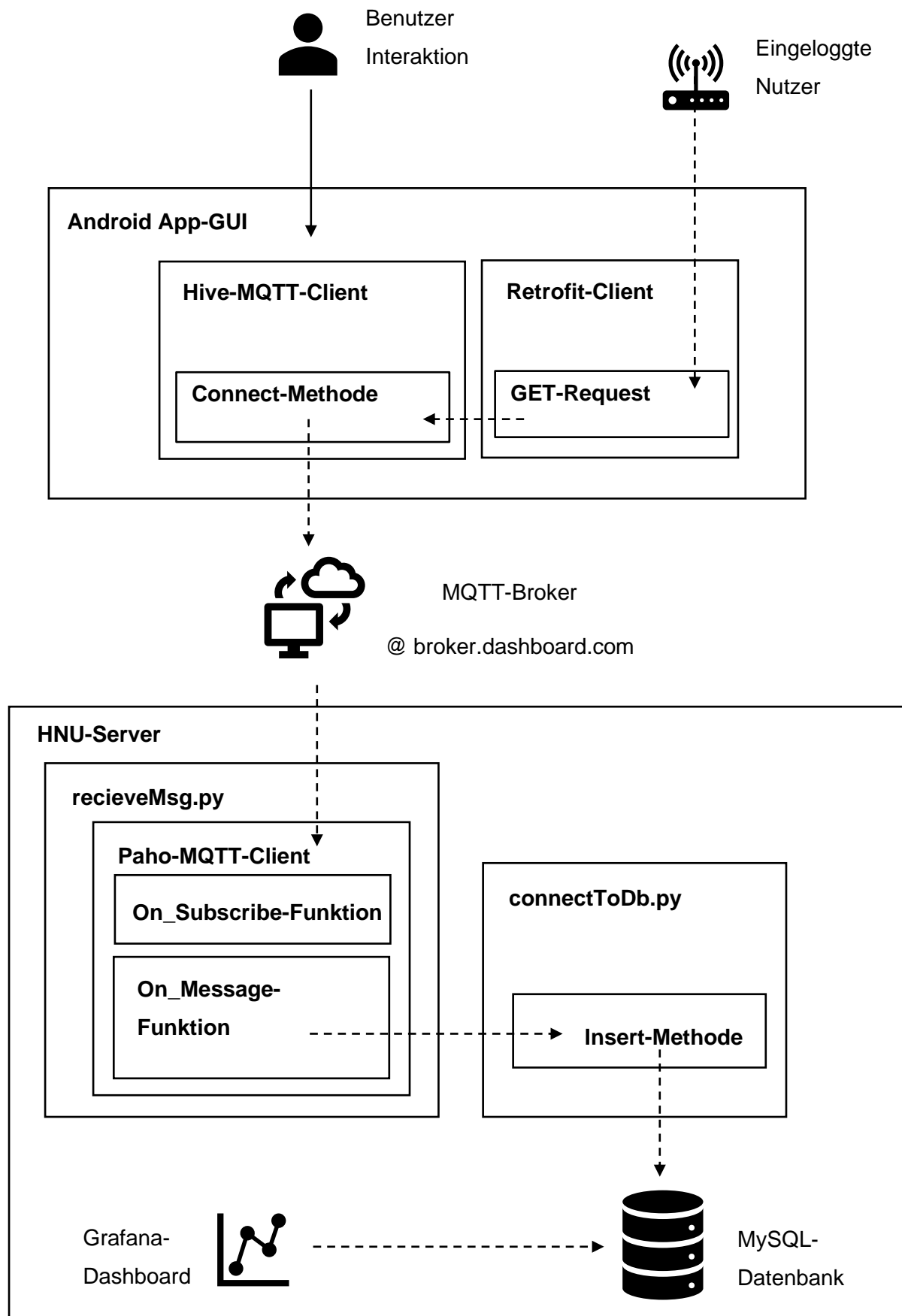
public void addNote() {
    Note note = new Note(title, description, priority);

    notebookRef.add(note);
}
```

Die FirebaseFirestore-Klasse erstellt eine Instanz der Firestore-Datenbank. Die Klasse CollectionReference fügt der Datenbank eine deklarierte Collection an. Es wäre ebenfalls möglich auf ein Dokument zu referenzieren, wobei in der Deklaration die Collection enthalten sein sollte („Name der Collection/Name des Dokuments“). Die Attribute des Note-Objekts können dabei als Strings, Integer, Booleans oder Listen (In Firebase: Maps) deklariert werden. Anfragen können nach dem CRUD-Prinzip, sowohl einzeln als auch gruppiert als Batch (nur schreiben), oder Transaktion (lesen und schreiben), ausgeführt werden.

Firebase bündelt viele Entwicklertools für native und hybride (Web-) Applikationen. In Verbindung mit dem MQTT und HTTP-Protokoll kann Firebase als Datenbank/Datenvisualisierung in eine IOT-Architektur implementiert werden. Noch sind viele Funktionen in der Beta-Version, doch für startende Entwickler ist es eine gute Möglichkeit mit dem Thema NoSQL und IoT in Berührung zu kommen.

3.3 Entwurfsunterlage Prototyp



3.4 Vorgehensweise

3.4.1 Android App

3.4.1.1 Benutzeroberfläche

Die Funktion der Android App ist es, Informationen aus zwei Datenquellen aufzunehmen und für die Weiterleitung über das MQTT-Protokoll zu formatieren. Eine Datenquelle resultiert aus der manuellen Eingabe eines Nutzers in der Android GUI, bestehend aus zwei Variablen: (1) Zufriedenheit des Besuchs skaliert von 1-5 und (2) Wahl der Mahlzeit (3 Optionen). Der andere Messwert gibt die Gesamtzahl der Netzwerkbenutzer der HNU an die App weiter. Die Benutzeroberfläche besteht aus zwei Fenstern. Abb. 14 ist als Startbildschirm festgelegt und Abb. 15 als Eingabefeld, welches nach Bestätigung wieder auf den Ausgangspunkt (Abb. 14) zurückversetzt wird. Um eine Bewertung zu erstellen muss die App mit dem Internet verbunden sein. Android Studio Projekte verbinden sich nicht automatisch mit dem Internet. Dies muss für den Nutzer über das AndroidManifest.xml freigeschaltet werden:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Der Netzwerkstatus wird über einen Connectivity-Manager bei der Button-Eingabe auf der Startseite geprüft. Sollte keine Verbindung erkannt werden, wird eine Fehlermeldung über ein Toastfeld ausgegeben. Die Eingabe bei der ersten Frage erfolgt über einen Zahlenauswahlrad und in der zweiten Frage über interaktive „Radiobuttons“, bei denen genau eine Option ausgewählt werden kann. Sollte der Benutzer den Bestätigungsknopf drücken, ohne eine Auswahl getroffen zu haben, so erscheint ein Toast mit einer Fehlermeldung.

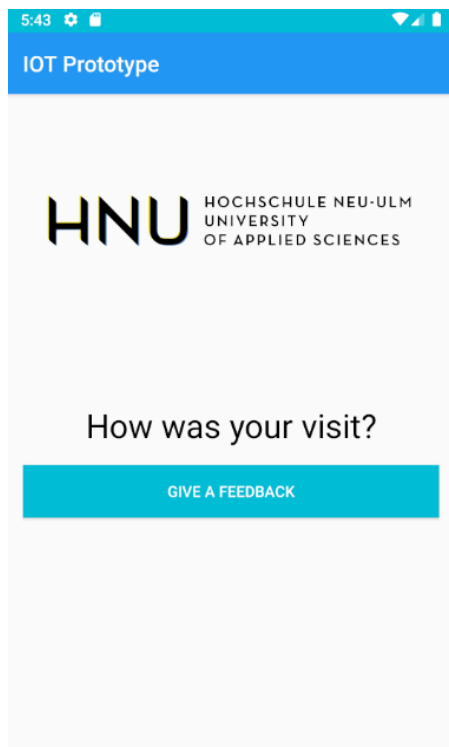


Abbildung 14: App-Startseite Version 1

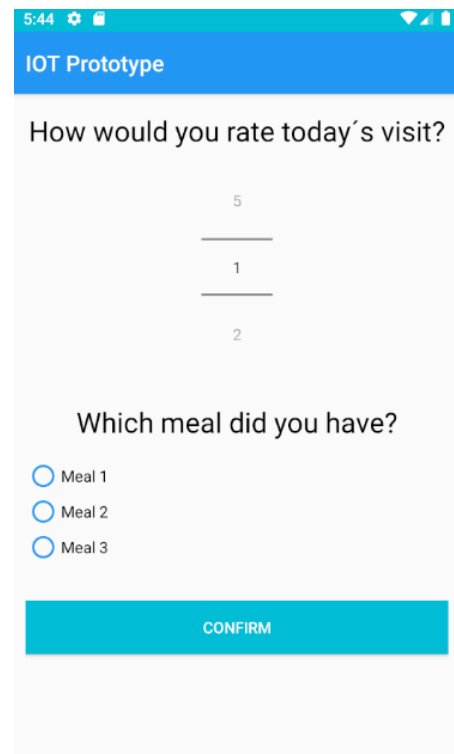


Abbildung 15: Benutzereingabe Version 1

3.4.1.2 MQTT-Client

Der MQTT-Client soll die eingegeben Daten aus dem Eingabefeld und die Antwort aus dem http-Request an den MQTT-Broker weiterleiten. Die entsprechenden Klassen und Methoden werden über das Gradle-Modul in Android Studio implementiert:

```
implementation group: 'com.hivemq', name: 'hivemq-mqtt-client', version: '1.1.3'
```

MQTT benötigt einen Java- Compiler in der Version 8 oder höher und muss implementiert werden:

```
compileOptions {
    sourceCompatibility = 1.8
    targetCompatibility = 1.8
}

packagingOptions {
    exclude 'META-INF/INDEX.LIST'
    exclude 'META-INF/io.netty.versions.properties'
}
```

Die Funktionsweise des MQTT-Clients ist dieselbe wie in Absatz 3.1.3. Innerhalb der Methode werden die Eingaben aus dem „Number Picker“, dem ausgewählten „Radio Button“ und der „http-Response“ als Integer- oder String-Variablen gespeichert. Diese werden in einen String zusammengefügt, welcher in der Nachrichten-Payload implementiert wird und mit der publishWith-Methode an das beigefügte Topic veröffentlicht wird. Die komplette Implementierung der startClient-Methode wird wie folgt dargestellt:

```
public void startClient(View v) {
    Mqtt5BlockingClient client = Mqtt5Client.builder()
        .identifier(UUID.randomUUID().toString())
        .serverHost("broker.mqttdashboard.com")
        .buildBlocking();

    client.connect();

    int rating = numberPicker.getValue();
    radioButton = findViewById(radioGroup.getCheckedRadioButtonId());
    int radioValue = radioGroup.indexOfChild(radioButton) + 1;

    if (radioValue == 0) {
        Toast.makeText(this, "Please select an option", Toast.LENGTH_SHORT)
            .show();
        return;
    }

    String message = "";
    message += rating + ";" + radioValue + ";" + users;

    client.publishWith().topic("mensaiot")
        .qos(MqttQos.AT_LEAST_ONCE)
        .payload(message.getBytes()).send();

    client.disconnect();
    finish();
}
```

```

    Toast.makeText(this, "Thanks for your rating!", Toast.LENGTH_LONG)
        .show();
}

```

Die Methode für den http-GET-Request ruft über ein Interface ein Callback mit dem erhaltenen „Response-Body“ auf. Bei einer erfolgreichen Verbindung wird die Antwort als String in der globalen Variablen „users“ gespeichert. Sollte die Verbindung fehlgeschlagen sein, wird eine Fehlermeldung ausgegeben. Der dazugehörige Android-Studio Code ist wie folgt:

```

private void getUsers() {
    Call<ResponseBody> responseBodyCall = restApi.getUsers();
    responseBodyCall.enqueue(new Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
            try {
                users = response.body().string();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
        Toast.makeText(ClientActivity.this, t.getMessage(), Toast.LENGTH_SHORT).show();
    }
}
}

```

3.4.2 Python

3.4.2.1 MQTT-Client

Um die Daten über den MQTT-Broker in einer MySQL-Datenbank abzuspeichern, werden zwei Python-Klassen benötigt. Über den MQTT-Client von Paho [17] kann ein Topic abonniert werden und vom Broker veröffentlichte Nachrichten empfangen werden. Der MySQL-Connector kann sich mit der Datenbank verbinden und über ein Insert-Statement Daten in die Tabelle übertragen. Die beiden Klassen werden als Paket mit dem „Python Package Installer“ (pip) über die Windows-Kommandozeile installiert.

Die implementierten Methoden werden in der Python IDE „PyCharm“ in zwei Skripte unterteilt. Das receiveMsg.py-Skript erstellt einen Client der MQTT-Operationen aufrufen kann. Der Client verbindet sich mit einem Broker („broker.mqttdashboard.com“) über einen Methodenaufruf am Objekt „client“ und abonniert das Topic „mensaiot“ über die Funktion „on_subscribe“. Die empfangenen Broker-Nachrichten wird über die „on_message“-Funktion aufgenommen und gibt die beigefügte „Payload“ als Print-Statement aus. Die „Payload“ und das „Topic“ werden parallel in einer importierten Funktion („sensor_data_handler“) übernommen, welche die MySQL-Datenbankabfrage ausführt. Die Verbindung zum Broker wird, solange das Skript ausgeführt wird, aufrechterhalten („loop_forever“).

```
import paho.mqtt.client as paho
from database import sensor_data_handler

def on_subscribe(client, userdata, mid, granted_qos):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))

def on_message(client, userdata, msg):
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))
    sensor_data_handler(msg.topic, msg.payload)

client = paho.Client()
```

```

client.on_subscribe = on_subscribe
client.on_message = on_message
client.connect("broker.mqttdashboard.com", 1883)
client.subscribe("mensaiot", qos=1)

client.loop_forever()

```

3.4.2.2 Datenbank-Connector

Die "DatabaseManager"-Klasse initiiert die Datenbankverbindung über eine „Connector“-Schnittstelle und erzeugt ein Objekt der Cursor-Klasse. Der Cursor erlaubt Python Datenbankoperationen auszuführen [18]. Um Daten in einer Tabelle zu speichern benötigt man die INSERT INTO SQL-Operation, welche unter der Angabe des Tabellennamen und der dazugehörigen Variablen, die zu speichernden Daten überträgt. Die vom MQTT-Broker gesendete Nachricht beinhalten einen Datensatz, der zuerst in einen String formatiert wird. Der String wird entsprechend angepasst, geteilt und in drei einzelne Integer gespeichert, um mit dem Datentyp in der Datenbank übereinzustimmen. Nach der erfolgreichen Query-Abfrage wird der Cursor geschlossen und die Verbindung beendet.

```

import mysql.connector

class DatabaseManager:
    def __init__(self):
        # Connect to MySQL-database
        self.cnx = mysql.connector.connect(user='IMA',
                                           password='ima',
                                           host='192.168.141.46',
                                           port='3306',
                                           database='MensaDB')

        self.cnx.commit()
        self.cursor = self.cnx.cursor()

```

```

def on_update_db(self, query, value=()):
    self.cursor.execute(query, value)
    self.cnx.commit()
    return

def __del__(self):
    self.cursor.close()
    self.cnx.close()

def mqtt_data_handler(data):
    message = str(data)
    message_raw = message[2:-3]
    message_split = message_raw.split(';')
    rating = int(message_split[0])
    meal = int(message_split[1])
    users = int(message_split[2])
    print("Data transfer successful!")

    db = DatabaseManager()
    db.on_update_db("INSERT INTO DataWH (rating, meal, users) VALUES (%s,%s,%s)",
[rating, meal, users])
    del db
    print("Inserting into Database...")

def sensor_data_handler(topic, data):
    if topic == "mensaiot":
        mqtt_data_handler(data)

```

3.4.3 MySQL-Datenbank

Die MySQL-Datenbank wird über die MySQL-Workbench-Desktopanwendung erstellt. Dazu wird eine Verbindung mit einer Serveradresse hergestellt. Da diese nur über das HNU-Netzwerk verfügbar ist, muss bei einem externen Zugriff ein Zugang mittels VPN hergestellt werden. Die erstellte Tabelle beinhaltet fünf Attribute. Die eindeutige ID gibt jedem gespeicherten Datensatz eine automatisch-erstellte („auto-incremented“) Kennzeichnung und ist zusätzlich als Primärschlüssel vermerkt. Damit können JOIN-Operationen mit anderen Tabellen ausgeführt werden. In den Spalten „Rating“, „Meal“ und „Users“ werden die Daten aus unseren Datenquellen als Integer gespeichert. Das Attribut „created“ enthält eine Zeitmarke (engl. „Timestamp“), welche jedem Datensatz hinzugefügt wird. Damit lässt sich zu jedem Eintrag das Datum und die genaue Uhrzeit feststellen. Über einen Konnektor wird die Datenbank mit der Visualisierungsanwendung Grafana verbunden und in einem Dashboard dargestellt.

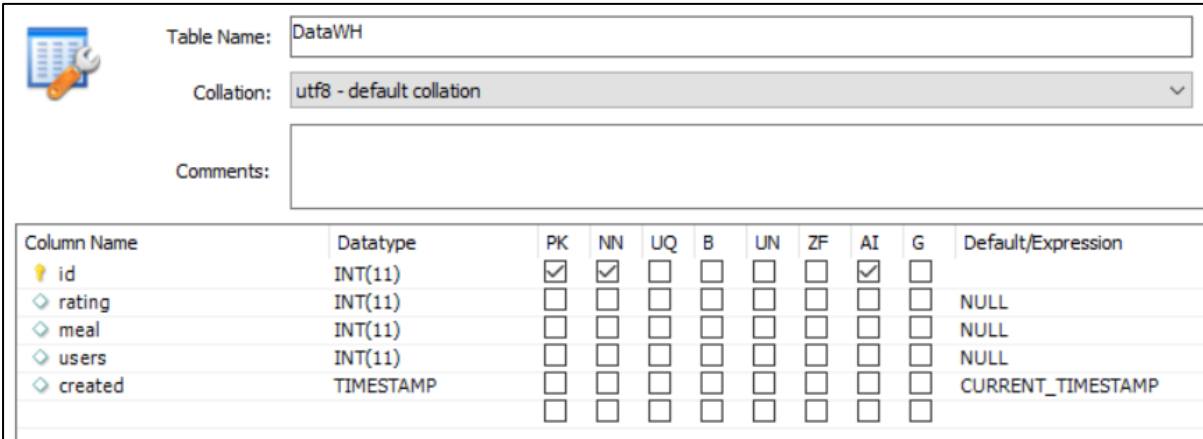


Table Name: DataWH

Collation: utf8 - default collation

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
rating	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
meal	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
users	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
created	TIMESTAMP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP

Abbildung 16: MySQL-Datenbankschema

3.4.4 Grafana-Dashboard

Grafana ist eine Open-Source Analyseplattform, die Daten aus heterogenen Datenquellen in sogenannten Dashboards visualisieren kann. Der Einsatz von Grafana ist plattformübergreifend und neue Features werden fortlaufend durch eine stark wachsende Nutzer-Community erschaffen [19]. Es werden über 30 Open-Source Datenquellen nativ unterstützt. Die daraus entnommenen Daten können in einer Vielzahl von Dashboards visualisiert werden und können über Abfragen und Filter je nach Präferenz angepasst werden. Damit sie als Graph dargestellt werden können, muss für ein Datensatz eine Zeitmarke gegeben sein, da Grafana die Daten zeitorientiert darstellt. Ein Dashboard kann mehrere Benutzer haben, weshalb Grafana insbesondere für BI-Projekte geeignet ist [19].

Um Grafana zu nutzen muss man die benötigten Dateien auf der Grafana Labs Startseite als zip-Datei herunterladen. Nach der Installation und der Extrahierung der Daten, muss man im Konfigurationsordner die Datei „custom.ini“ öffnen. Dort stellt man den Standard-Port auf 8080. Nun kann die Anwendung über die grafana-server.exe-Datei gestartet werden. Das Grafana Dashboard muss in einem Browser über die Localhost-Adresse (127.0.0.1:8080) geöffnet werden. Nach der Anmeldung wird der Nutzer Schritt für Schritt in die Erstellung eines Dashboards eingewiesen.

Um die MySQL Datenbank mit Grafana zu verknüpfen, benötigt man den Namen und die Verbindungsparameter der Datenbank. Die Verbindung ist permanent und muss bei einem Neustart nicht neu initiiert werden. Die Daten der entsprechenden Tabelle werden in einer oder mehrerer Abfragen dargestellt. Das Dashboard bildet die Anzahl der Internetnutzer (y-Achse) an der HNU über einen bestimmten Zeitraum (x-Achse) von 15 Minuten ab. Dabei wird die Dashboard-Ansicht alle fünf Sekunden aktualisiert. Die Abfrage in Abb.17 ist in einer SQL-Ansicht dargestellt und kann auf alle sich in der Tabelle befindlichen Parameter zugreifen.

FROM	DataWH	Time column	created	Metric column	none
SELECT	Column: users	Alias: users	+		
WHERE	+				
GROUP BY	time (1s, none)	+			
Format as	Time series		Edit SQL	Show Help	

Abbildung 17: Grafana SQL-Query

4 Fazit

Mit dem IoT wird man sich noch in den nächsten Jahren explizit auseinandersetzen. Die Materie bietet viele Möglichkeiten, Dinge untereinander zu vernetzen. Die daraus anfallenden Datenmengen können mittels Analysetools in konkrete Aussagen und Erwartungen transformiert werden. Durch die wissenschaftliche Literaturrecherche konnte die Schichten einer IoT-Architektur erläutert werden. Durch die fortlaufende Erweiterung lässt sich aber kein allgemeingültiger Architekturtyp bestimmen. Die Strukturen sind meist flexibel und werden je nach Art der Anwendung und den benötigten Anforderungen angepasst.

Für das IoT-Szenario „Mensa“ wurde ein Prototyp erfolgreich erstellt. Daten aus einer Benutzereingabe und die genaue Anzahl der Netzwerknutzer der HNU können in einem Dashboard dargestellt werden. Die Erstellung der einzelnen Ansätze erwies sich in einigen Teilen als recht kompliziert. Durch Recherche in Internetforen, wie „Stackoverflow“ konnten Methoden erfasst und getestet werden, diese waren aber nicht immer umsetzbar. Das MQTT-Protokoll ist für diese Art der Datenübertragung am besten geeignet. In Zukunft wäre es möglich den Broker lokal über einen Raspberry Pi laufen zu lassen, da der Webserver nicht permanent zu erreichen ist. Die Übermittlung vom Broker zur Datenbank läuft noch über einen privaten Rechner. Das zuständige Python-Skript könnte in den Hochschulserver implementiert werden, um eine dauerhafte Verfügbarkeit zu garantieren.

In Zukunft wäre eine weitere wissenschaftliche Forschung als Anknüpfung an diese Seminararbeit möglich. Die Erweiterung der Anwendung mit anderen Datenquellen, beispielsweise Umgebungssensoren, könnte realisiert werden. Die App erfüllt die funktionalen Anforderungen und könnte designtechnisch weiterentwickelt werden. Die auszuwählenden Optionen sind noch fest im Code integriert. So könnte man ein JSON-File mit der aktuellen Speisekarte in einer Web-basierten NoSQL-Datenbank abspeichern und dieses über ein GET-Request in die Applikation lädt. NoSQL-Datenbanken haben enormes Potenzial für die Speicherung IoT-bezogener Daten. Durch ständige Erweiterungen im Bereich BI und Monitoring, werden Sie eine agile Alternative zu relationalen Datenbanken.

5 Literaturverzeichnis

- [1] D. E. Boubiche, H. Hamdan und F. Hidoussi, Hg., *IML'17: Proceedings of the International Conference on Internet of Things and Machine Learning : October 17-18, 2017, Liverpool John Moores University (LJMU), Liverpool city, United Kingdom*. New York, New York: The Association for Computing Machinery, 2017.
- [2] A. Alreshidi und A. Ahmad, „Architecting Software for the Internet of Thing Based Systems“, *Future Internet*, Jg. 11, Nr. 7, S. 153, 2019.
- [3] *IOT-A*. [Online] Verfügbar unter: <https://cordis.europa.eu/project/id/257521>.
- [4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari und M. Ayyash, „Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications“, *IEEE Commun. Surv. Tutorials*, Jg. 17, Nr. 4, S. 2347–2376, 2015.
- [5] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos und P. Christen, „MOSDEN: An Internet of Things Middleware for Resource Constrained Mobile Devices“ in *2014 47th Hawaii International Conference on System Sciences*, Waikoloa, HI, Jan. 2014 - Jan. 2014, S. 1053–1062.
- [6] *Cloud Computing*. [Online] Verfügbar unter: <https://azure.microsoft.com/de-de/overview/what-is-cloud-computing/>.
- [7] E. Pencheva und I. Atanasov, „Engineering of web services for internet of things applications“, *Inf Syst Front*, Jg. 18, Nr. 2, S. 277–292, 2016.
- [8] *PHP Documentation*. [Online] Verfügbar unter: <https://www.php.net/manual/de/intro-what-is.php>.
- [9] S. Jaloudi, „Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study“, *Future Internet*, Jg. 11, Nr. 3, S. 66, 2019.
- [10] *HiveMQ Essentials*. [Online] Verfügbar unter: <https://www.hivemq.com/blog/>.
- [11] *HiveMQ Github Documentation*. [Online] Verfügbar unter: <https://hivemq.github.io/hivemq-mqtt-client/>.
- [12] D. Bartholomew, *Mariadb vs. MYSQL*. [Online] Verfügbar unter: http://rozero.webcindario.com/disciplinas/fbm/abd3/MariaDB_vs_MySQL.pdf.
- [13] A. Celesti, M. Fazio und M. Villari, „A Study on Join Operations in MongoDB Preserving Collections Data Models for Future Internet Applications“, *Future Internet*, Jg. 11, Nr. 4, S. 83, 2019.

- [14] *MongoDB*. [Online] Verfügbar unter: <https://www.mongodb.com/nosql-explained>.
- [15] *RestDB Documentation*. [Online] Verfügbar unter: <https://restdb.io/docs/>.
- [16] *Google Firebase Dokumentation*. [Online] Verfügbar unter:
<https://firebase.google.com/docs/firestore/>.
- [17] *Paho MQTT-Client*. [Online] Verfügbar unter: <https://pypi.org/project/paho-mqtt/>.
- [18] *Python Cursor*. [Online] Verfügbar unter: <http://initd.org/psycopg/docs/cursor.html>.
- [19] *Grafana Dashboard*. [Online] Verfügbar unter: <https://grafana.com/grafana/>.

Disclaimer

Hiermit erkläre ich, dass ich die vorliegende Seminararbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Seminararbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ulm, 22.01.2020

Ort, Datum



Unterschrift