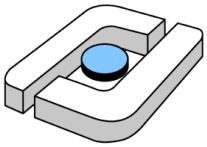

Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Hausarbeit

über das Thema:

Implementierung einer Datenbankverbindung über eine JDBC-Schnittstelle innerhalb einer Android-Applikation

Autoren:	Dominik Feldkamp Matrikelnr.: 587123
	Stefan Pikulik Matrikelnr.: 587363

Modul:	Datenbanken in der Automatisierungstechnik
Studiengang:	Elektrotechnik - Automatisierungssysteme
Prüfer:	Friedhelm Tappe
Abgabedatum:	28.07.2017

I Inhaltsverzeichnis

II Abbildungsverzeichnis

III Abkürzungsverzeichnis

App	Applikation
CO	Kohlenstoffmonoxid
CO2	Kohlenstoffdioxid
DB	Datenbank
ISDP	International Sensor Development Project
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation

1 Einleitung

In Konzert- oder Sporthallen, Unterrichtsräumen oder kleinen Büros führt schlechte Luftqualität oft zu Unbehagen. Mit einem hohen Stickstoff-Gehalt der Luft sind Konzentrationsschwächen meist einhergehend. Auf Basis dieser Problematik entstand an der Hochschule Osnabrück in Kooperation mit der finnischen Universität in Helsinki ein Projektteam, das sich dieser Problematik annahm. Ziel war es ein System zu entwickeln, das die Luftqualität eines Raums misst, speichern und weiterreichen kann.

Diese Projektarbeit resultierte in der Konstruktion eines Luftqualitäts-Messgeräts in Form eines Arduino-Boards. An dieses Board sind Sensoren zum Messen von Luft-Qualitätskriterien wie CO, CO2 oder Rauch angebracht. Diese werden auf einer SD-Karte gespeichert, um auch Zugriff auf Langzeitdaten zu erhalten. Zusätzlich zu diesem Messgerät wurde eine Applikation entwickelt, mit der aufgenommene Messwerte für den Benutzer visualisiert werden können. Über Bluetooth-Kommunikation erfolgt der Datentransport zwischen Applikation und Messgerät, dabei sind sowohl aktuelle Sensorwerte als auch Langzeitdaten abrufbar.

Damit diese Daten auch außerhalb der Reichweite der Bluetooth-Verbindung abrufbar sind, wird in dieser Hausarbeit eine Erweiterung zu der bisher genutzten Applikation entwickelt. So soll es für jeden Nutzer der Applikation in Zukunft möglich sein, Daten z.B. über eine WLAN-Verbindung ohne physikalische Abhängigkeit zum Zugriffsort abrufen. Dieser Zugriff wird in dieser Hausarbeit durch das Einbinden eines Java-Database-Connection-Treibers realisiert. Dabei gilt es zuerst das bisherige Datenspeicher-Konzept zu überarbeiten und anschließend den besagten Treiber lauffähig zu machen.

Diese Hausarbeit fängt mit einer Einführung in die Thematik an, in der das oben genannte Projekt, sowie der Entwicklungsstand der Applikation, erläutert werden. Als Folge wird die Motivation für diese Arbeit thematisiert. Im nachfolgenden Kapitel wird das umzusetzende Konzept erläutert. Dieses Konzept wird im Kapitel zur Implementierung umgesetzt und getestet. Abgerundet wird der Bericht durch einen Ausblick und eine Zusammenfassung.

2 Einführung in die Thematik

Zur Einordnung in die Entwicklungsumgebung und Ziele der Arbeit werden nachfolgend das bisher entstandene Projekt einer abgeleisteten Studienarbeit vorgestellt. Im Anschluss werden Funktionen der implementierten Applikation innerhalb des Projekts erläutert. Abschließend folgt die Motivation zur Weiterführung des Projekts in Kombination mit diesem Studienfach. Für detaillierte Informationen sei auf die Ausarbeitung des Projekts verwiesen. [1]

2.1 ISDP-Projekt

Zur Überwachung der Luftqualität, z. B. in Büros oder Laborräumen, ist in dem Modul International Sensor Development Project (ISDP) in Kooperation mit der Metropolia Universität aus Helsinki in Finnland ein Messgerät mit unterschiedlichen Sensoren entwickelt worden. Das Messgerät misst die Raumtemperatur, die Luftfeuchtigkeit, den Luftdruck, den Rauch, das Kohlenstoffdioxid (CO2), das Kohlenstoffmonoxid (CO), verschiedene Stickstoffe und Ethanol. Diese Messdaten werden über ein Arduino-Board kontinuierlich über jeweils eine Minute gemittelt und auf einer SD-Card abgelegt. Bei Abschluss eines Tages werden diese Daten (Langzeitdaten genannt) in einem Dokument gespeichert. Durch die Bewertung der Messwerte soll es in der Zukunft möglich sein Aktoren zu steuern, um beispielsweise bei schlechter Luftqualität Fenster zu öffnen. Zur Anzeige von Messwerten ist daher eine Android-Applikation (App) entwickelt worden. Dabei sind aktuelle Messwerte oder Langzeitdaten von der SD-Karte eines Arduinos in JavaScript Object Notation (JSON) mittels Smartphone über eine Bluetooth-Verbindung von der App abrufbar. Hierbei sind bislang Tages- und Langzeitdaten einzelner Sensoren je Arduino darstellbar. In Abbildung 2-1 ist der beschriebene Sachverhalt grafisch dargestellt.

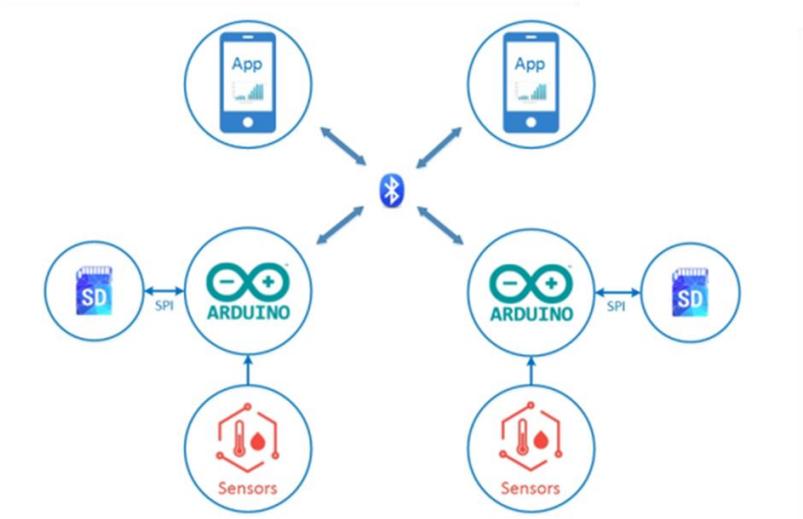

Abbildung 2-1 Übersicht über das ISDP-Projekt

Der Aufbau und die Funktionen der App werden im Folgenden näher erläutert, da die grundsätzliche Funktionsweise für die Implementierung einer Datenbankverbindung in einer App berücksichtigt wird.

2.2 Aufbau und Funktion der ISDP-Applikation

Die App zur Darstellung der Messwerte besteht aus einem Frontend und einem Backend. Das Frontend bildet die Schnittstelle für Benutzer zur Interaktionen mit der App. Durch den Benutzer ausgelöste Ereignisse (z.B. das Drücken eines Knopfes) werden im Backend verarbeitet. Im Backend-Teil erfolgt die Implementierung der Kommunikationsschnittstellen zwischen den einzelnen Klassen der App. Des Weiteren besteht die App aus den Komponenten Bluetooth, Speicherverwaltung und dem JSON-Parser. Der Aufbau der App ist in der Abbildung 2-2 dargestellt und wird im Folgenden beschrieben.

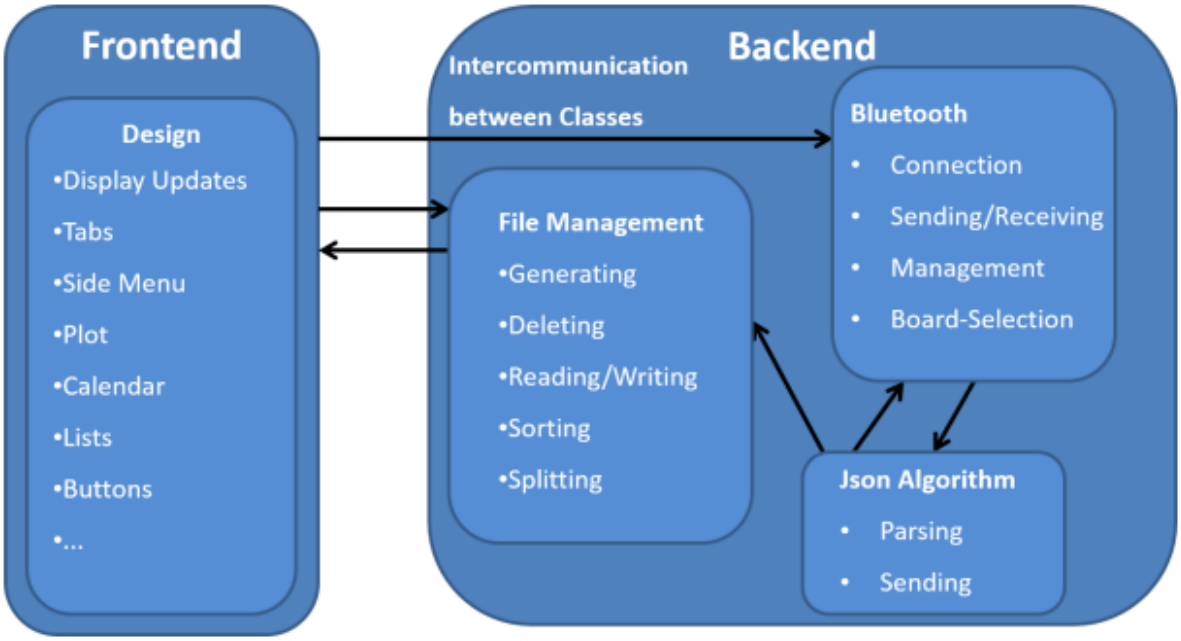

Abbildung 2-2 Aufbau der ISDP-Applikation

Über eine Bluetooth-Schnittstelle stellt die App eine Verbindung zu einem Arduino-Board her. Dadurch werden die aktuellen Sensorwerte oder die Langzeitmesswerte des entsprechenden Messgerätes abgefragt. Diese Messdaten werden im JSON-Format gesendet. Die verwendete Struktur des JSONs ist in Abbildung 2-3 dargestellt. Im JSON-Format erfolgt die Zuweisung von Variablennamen und Werten über das Symbol „:“. So ist z.B. die MAC-Adresse über das Kürzel „MAC“ lokalisierbar und dessen Wert „YY:XX:XX:XX:XX:XX“ hinter dem „:“ eingespeichert.

```
strJson="{\"MAC\":\"YY:XX:XX:XX:XX:XX\",\"NAME\":\"AirQ1\",\"DATE\":\"2017-01-30\"\n, "
+"\"CurrentValues\":{\"DATE\":\"2017-01-
30\",\"TIME\":\"16:30\",\"MQ2\":\"70\",\"MQ7\":\"101\",\"MQ135\":\"102\",\"MQ131\":\"103\",\"
MD62\":\"104\",\"Temperature\":\"13\",\"Pressure\":\"109\",\"Humidity\":\"13\",\"Loudness\":\"10
9\"},"+"\n" + "\"DATA\" :" +
"["+"{\"TIME\":\"16:30\",\"MQ2\":\"70\",\"MQ7\":\"101\",\"MQ135\":\"102\",\"MQ131\":\"103\",
\"MD62\":\"104\",\"Temperature\":\"13\",\"Pressure\":\"109\",\"Humidity\":\"13\",\"Loudness\":\"1
09\"},"+"\n" + "] };";
```

Abbildung 2-3 Beispielhafter JSON-String

Nach diesem Verfahren sind die anderen zu übertragenden Werte ebenfalls abrufbar. Für nähere Informationen zum JSON-Format und die Einbindung unter Android sei auf [2] verwiesen.

Durch den JSON Parser werden die empfangenen Daten entschlüsselt und während der Laufzeit in generisch erstellte Textdokumente innerhalb der App gespeichert. Dazu wurde innerhalb der App eine Speicherverwaltung angelegt. Damit die unterschiedlichen Arduinos zugeordnet werden können, werden die gekoppelten Arduinos im Textdokument „CoupledDevices.txt“ und alle verfügbaren Arduinos in „Arduino.txt“ abgespeichert. Die aktuellen Sensordaten werden mit der Kennzeichnung des jeweiligen Arduinos in einer Textdatei „AirQX-CurrentValues.txt“ abgelegt. Das „X“ kennzeichnet die unterschiedlichen Namen der Arduinos (z.B. AirQ**1**). Die Langzeitdaten werden ebenfalls in einem Textdokument abgespeichert. Die Kennzeichnung des Textdokumentes ist dabei „AirQX-dd-mm-yyyy“. AirQX entspricht dem Namen des Messgerätes und dd-Tag mm-Monat yyyy-Jahr dem aufgenommenen Datum. Zusätzlich wird für jedes Messgerät eine Textdatei angelegt/aktualisiert, in welcher die jeweiligen vorhandenen Langzeitdaten stehen (AirQX.txt). Die Daten der Speicherverwaltung können innerhalb der App gelöscht, aus dem Textdokument herausgelesen und hingeschrieben werden. Damit die App die Langzeitdaten und die aktuellen Sensorwerte darstellen kann, greift das Frontend auf die Speicherverwaltung zu. Abbildung 2-4 zeigt den Screen der Speicherverwaltung mit beispielhaft vorhandenen Textdokumenten innerhalb der App an.

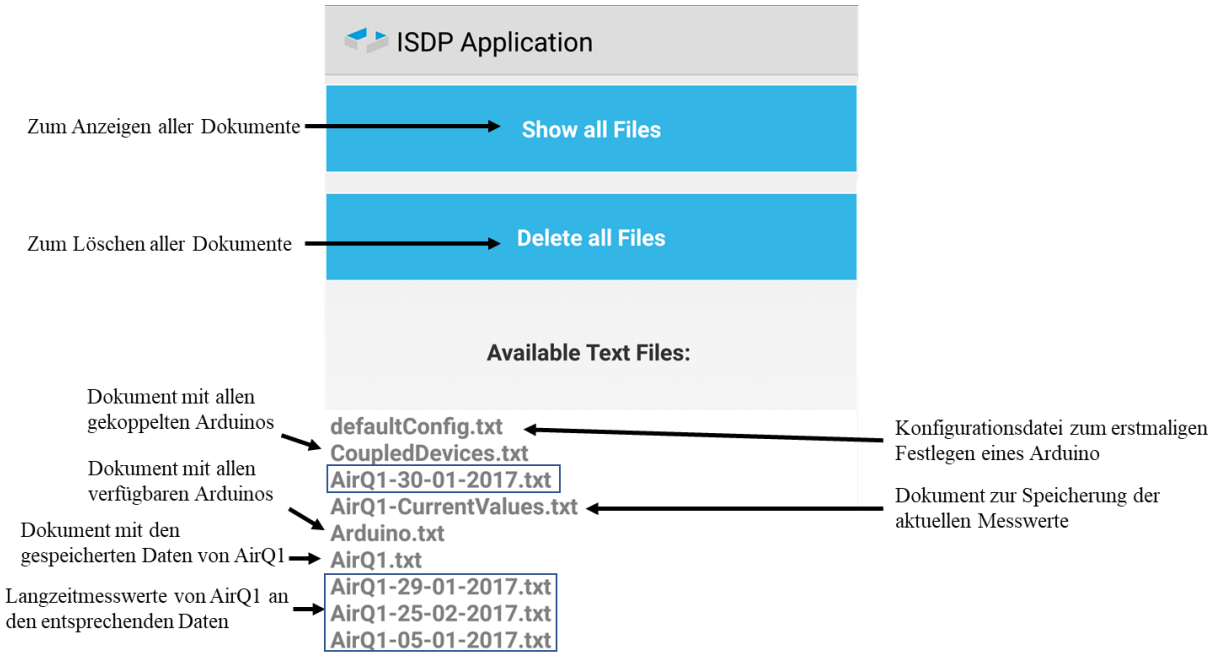

Abbildung 2-4 Screen zur Speicherverwaltung

Zur Darstellung der Sensorwerte ist ein Design für das Frontend der App entwickelt worden. In der Abbildung 2-5 ist der Screen für die Darstellung der aktuellen Tageswerte abgebildet.

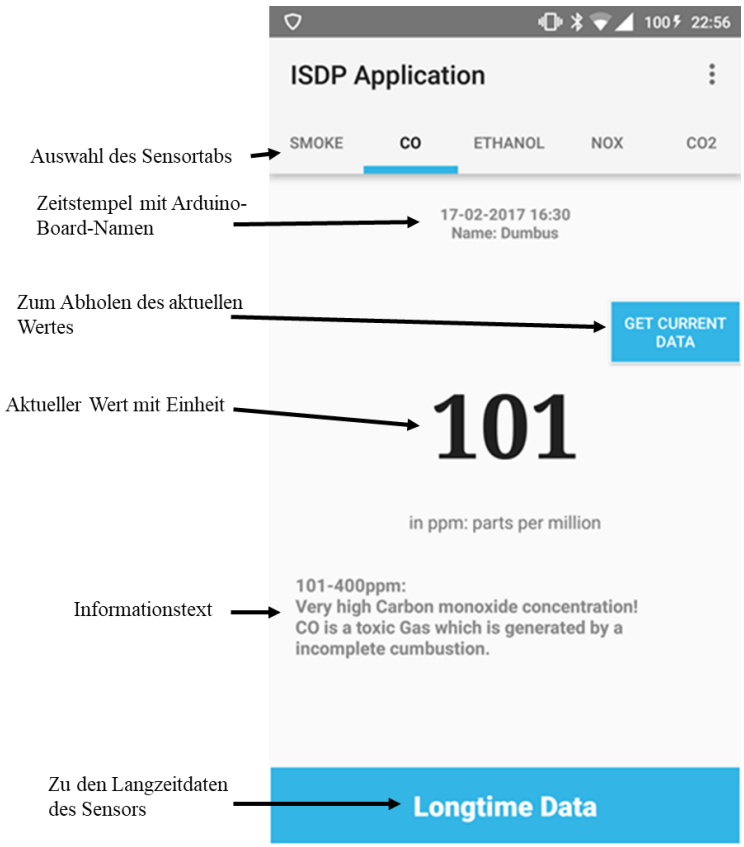

Abbildung 2-5 Screen zur Darstellung des aktuellen CO-Sensorwertes

Die Erläuterung wie ein Arduino ausgewählt wird, erfolgt später in diesem Unterkapitel. Im oberen Bereich von Abbildung 2-5 sind mehrere Tabs erstellt worden, in denen die jeweiligen Sensoren eines Messgeräts ausgewählt werden können. Der aktuell ausgewählte Sensor wird durch die blaue Linie im Tab gekennzeichnet. Unterhalb der Tabs wird der Zeitstempel mit dem dazugehörigen Namen des Arduino-Boards dargestellt. Durch den Button „Get Current Data“ wird eine Bluetooth-Verbindung zum ausgewählten Messgerät aufgebaut und aktuelle Sensorwerte abgeholt. Der aktuelle Sensorwert mit entsprechender Einheit wird in der Mitte des Screens abgebildet. Im unteren Bereich werden Informationstexte über einen Wertebereich mit Klassifikation des jeweiligen Sensorwertes angezeigt. Dadurch kann der aktuell angezeigte Messwert mit der Einheit eingeordnet werden. Mit dem Button „Longtime Data“ kann in den Screen der Langzeitdaten des ausgewählten Sensors gewechselt werden, der die Langzeitmesswerte innerhalb eines Diagrammes von einem Tag oder von den letzten drei Tagen abbildet. Der Screen für die Darstellung der Langzeitdaten für den jeweiligen Sensor ist in der Abbildung 2-6 im linken Teil dargestellt.

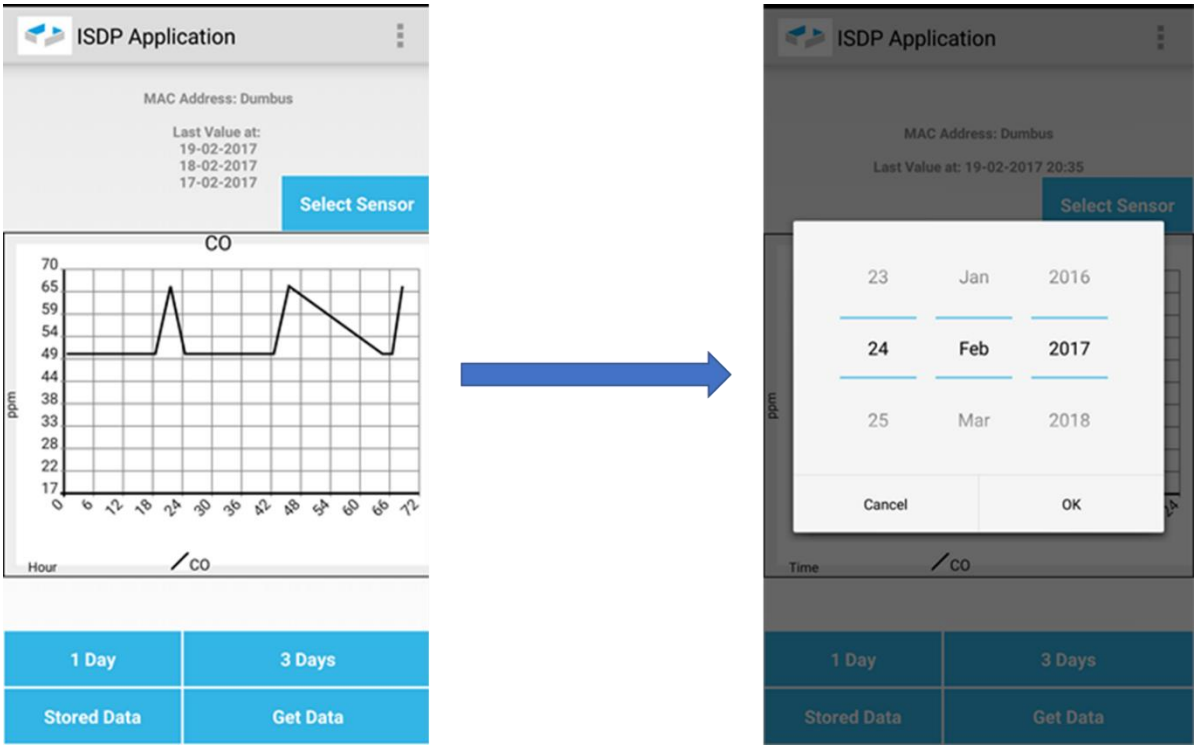

Abbildung 2-6 Screen zur Darstellung der Langzeitdaten des CO-Sensors (links) und der implementierten Kalenderfunktion (rechts)

Im oberen Bereich des Screens zur Darstellung der Langzeitdaten eines Sensors wird anzeigt, von welchem Arduino an welchem Datum die Sensorwerte im Diagramm dargestellt werden. Durch den Button „Select Sensor“ können die unterschiedlichen Sensoren ausgewählt werden.

Mit den unteren Buttons kann ein Tagesplot, ein Dreitagesplot bzw. durch „Stored Data“ mit Hilfe eines Pop-Up-Kalendermenüs Tageswerte für ein beliebiges Datum ausgewählt werden. Das Auswählen eines Datums ist in der Abbildung 2-6 im rechten Teil dargestellt. Sind die Sensorwerte nicht in Form eines Textdokuments innerhalb der App vorhanden, so können sich die Tagesdaten für ein ausgewähltes Datum direkt von einem Arduino-Board über Bluetooth geladen werden.

Über die Punkte im rechten oberen Bereich im linken Teil der Abbildung 2-7, gelangt der Benutzer in das Seitenmenü. Die Auswahlmöglichkeiten die durch das Seitenmenü zu erreichen sind, sind im rechten Teil der Abbildung 2-7 dargestellt.

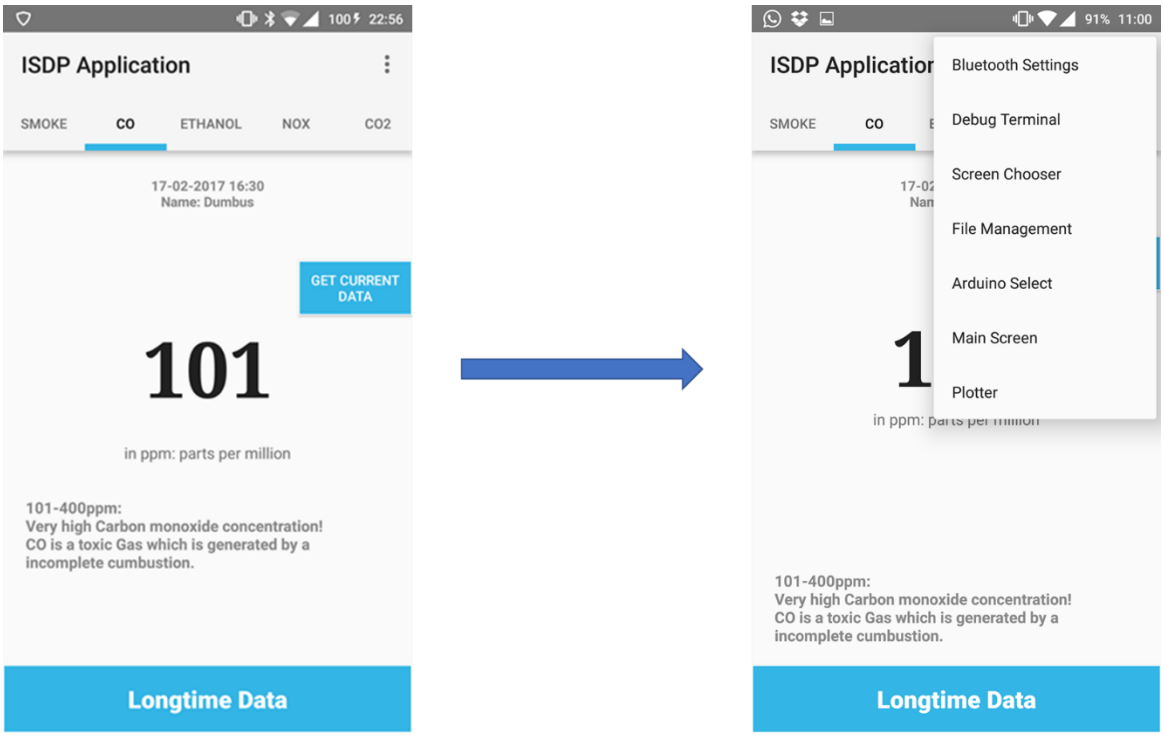

Abbildung 2-7 Auswahlmöglichkeiten im Seitenmenü

In dem Screen „Bluetooth Settings“ können verschiedene Einstellungen zu der Bluetooth-Verbindung eingestellt werden, wie z.B. die Übertragungsrate. Mit dem zweiten Menüpunkt „Debug Terminal“ gelangt der Nutzer in einen Screen, der für Debugzwecke in der App vorhanden ist. Durch das Klicken „Screen Chooser“ wird der Nutzer zu einem anderen Auswahlmenü geführt, in dem er in die Screens der Speicherverwaltung, der Anzeige der aktuellen Sensordaten oder der Langzeitdaten wechseln kann. Der Menüpunkte „File Management“ führt den Nutzer direkt zu dem Screen der Speicherverwaltung. Ein Arduino mit dem sich die App verbinden soll, wird in „Arduino Select“ ausgewählt. Dazu wird der Benutzer auf einen anderen Screen

geleitet, indem er anhand einer Listenansicht einen Arduino auswählen kann. Diese Listenansicht ist in der Abbildung 2-8 abgebildet. Mit den letzten beiden Menüpunkten kommt der Nutzer durch „Main Screen“ in den Screen zur Anzeige der aktuellen Sensordaten und durch „Plotter“ zu dem Diagramm zur Darstellung von Langzeitdaten.

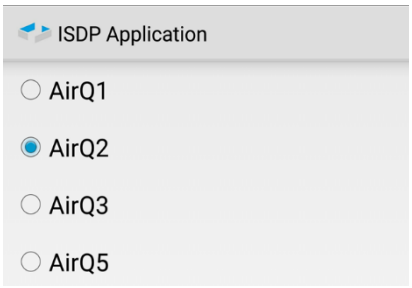

Abbildung 2-8 Liste zum Auswählen eines Arduinos

2.3 Motivation zur Weiterführung der ISDP-APP

Die Ziele dieser Arbeit zum Weiterführen der ISDP-App gliedern sich in zwei Teilkategorien. Zum einen wird das bisherige Modell zum Speichern der Daten neu bewertet und angepasst. Über eine SQLite-Datenbank Integration innerhalb der App sollen erhaltene Messwertdaten sinnvoll in dieser SQLite-Struktur gespeichert werden. Dieses Verfahren soll das bisher genutzte Speicherverhalten ersetzen.

Weiterhin wird die Notwendigkeit über eine physikalische Erreichbarkeit zwischen App und Messgerät überarbeitet. Jeder Nutzer der ISDP-App soll in Zukunft die Möglichkeit haben sich Langzeitdaten über eine externe Datenbank abzurufen. Dafür wird der JDBC-Treiber genutzt, welcher bei Einbinden in einer Java-Anwendung die Möglichkeit eröffnet mit externen Datenbanken zu kommunizieren. Dadurch kann z.B. bei bestehender Datenverbindung des Smartphones an einem beliebigen Ort Daten von der Datenbank heruntergeladen werden und in die intern zu erstellende SQLite-Datenbank gespeichert werden.

Daher wird in dieser Hausarbeit eine Prototyp-App entwickelt, welche die Funktionen zur Datenanbindung und zur Speicherverwaltung der benötigen Sensordaten beinhaltet. Das bedeutet konkret, dass die App sowohl eine interne SQLite-Datenbankanbindung, sowie eine JDBC-Treiberanbindung beinhaltet. Eine Implementierung in die bestehende ISDP-App erfolgt in dem Zeitbereich dieser Hausarbeit nicht, da dies den Zeitrahmen dieses Projektes überschreiten würde. Ein weiterer problematischer Umstand ist, dass die aktuellen Messgerät-Prototypen nicht verwendbar sind. Die Hardware wird aktuell im Laborbereich der Messtechnik an der Hochschule Osnabrück überarbeitet und sind in keinem einsatzfähigen Zustand. Daher wird eine prototypische neue App erstellt, die auf die Parameter-Strukturen und das Datenformat der ISDP-App angepasst sind. Somit ist nach erfolgreichem Abschluss dieser Arbeit nur noch die Integration der Komponenten in die ISDP-App notwendig.

3 Konzept

Nachfolgend wird das Konzept rund um die Erstellung der prototypischen Applikation vorgestellt. Zuerst erfolgt eine Einordnung der genutzten ISDP-Datenstrukturen in ein EER-Modell. Anschließend werden neue Konzepte zur Datenverwaltung vorgestellt und zusammenfassend eine resultierende Konzeptlösung präsentiert.

3.1 EER-Modell

Zur Ermittlung der Datenstruktur für die Speicherverwaltung innerhalb der App und in der externen Datenbank wird ein EER-Modell erstellt. Das erstellte EER-Modell ist in Abbildung 3-1 zu sehen.

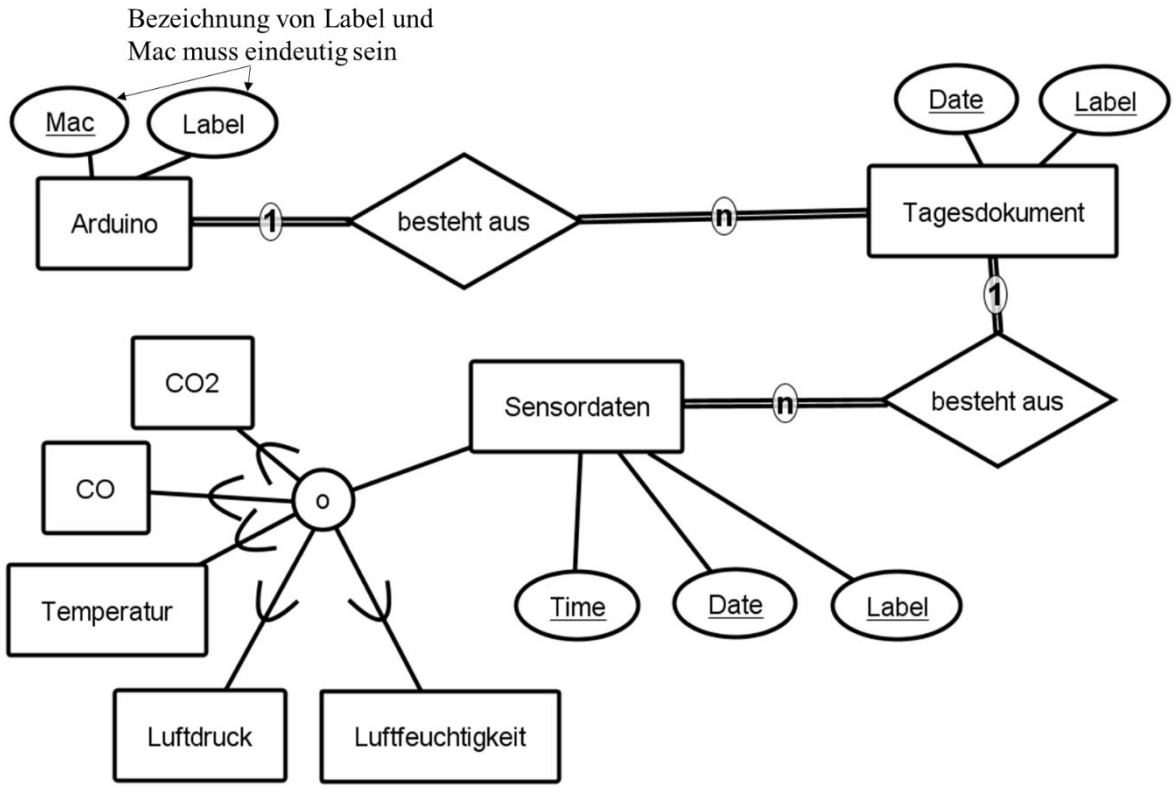

Abbildung 3-1 EER-Modell zur die Strukturierung der Daten aus ISDP

Die Entität Arduino enthält den Namen als Label und die Mac-Adresse des Messgeräts als Attribute. Diese beiden Attribute müssen in ihrer Bezeichnung eindeutig sein. Für das Abspeichern und Verwalten von Tagesdaten gibt es die Entität Tagesdokument mit den Attributen Date und Label, die in Kombination eindeutig sein müssen. Dabei besteht eine Beziehung zu einem Arduino, denn ohne einem Arduino ist es nicht möglich, Tagesdokumente zu erstellen.

Das Attribut Label ist ein Fremdschlüssel, dass sich auf das Label der Arduinodaten bezieht. Zusätzlich gibt es die Entität Sensordaten, in der Sensorwerte und ein Zeitstempel hinterlegt werden. Dieser Zeitstempel ist durch die Kombination aus Label, Date und Time eindeutig definiert. Auch hier gilt, ohne einen Arduino und einem hinterlegten Tagesdokument kann es keine Sensordaten geben. Daher besteht eine Beziehung zwischen den Sensordaten und den Tagesdokumenten. Dabei kann ein Tagesdokument eine beliebige Anzahl von Sensordaten besitzen. Die Attribute Label und Date sind als Fremdschlüssel den gleichnamigen Attributen aus dem Tagesdokument zugeordnet.

Aus dem ERR-Modell wird das relationales Schema für die Arduinodaten, der Tagedokumente und der Sensordaten für die Hinterlegung in die Speicherverwaltung innerhalb der Prototyp-App und für die externe Datenbank ermittelt. Die überlappenden Spezifikation der einzelnen Sensortypen bei Entität „Sensordaten“ werden durch Restrukturierung in Attribute von den Sensordaten umgewandelt. Im folgenden wird das relationale Schema der einzelnen Entitäten mit deren Beziehungen und beispielhaften Werten in Abbildung 3-2 dargestellt.

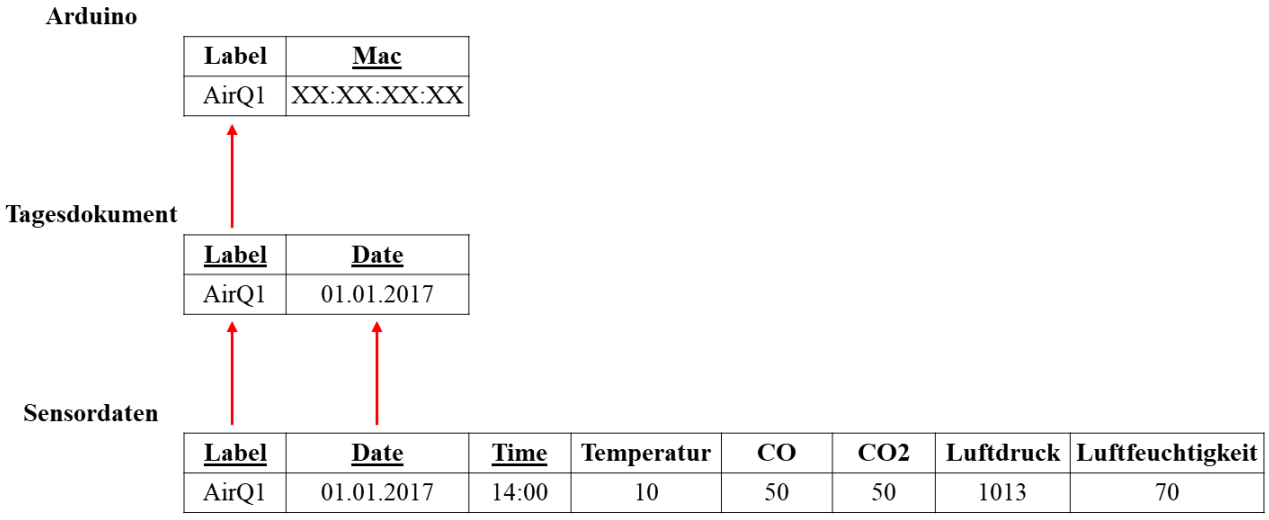

Abbildung 3-2 Relationales Schema der Arduinodaten, des Tagesdokumentes und der Sensordaten mit deren Abhängigkeiten

3.2 Datenverwaltung

Nachfolgend werden die Konzepte zum Speichern von Daten innerhalb der Applikation und außerhalb der Applikation erläutert. Dabei werden die Umstellung der internen Datenspeicherung auf SQLite und das Verfahren zum Nutzen des JDBC-Treibers vorgestellt.

3.2.1 Speichern von Daten innerhalb der Applikation

Im Rahmen dieser Arbeit wird das bisherige Speichermodell, wie in Kapitel 2.3 erwähnt, überarbeitet. Das Speichern von Daten erfolgt derzeit in Form von verschiedenen Textdokumenten, wie bereits in Kapitel 2.2 erläutert. Dadurch ergibt sich unter Umständen ein hoher Verwaltungsaufwand. Einmal gespeicherte Werte müssen immer komplett ausgelesen werden, um gesuchte Einträge zu lokalisieren und zu ändern. Wenn nun z.B. Dateien über Monate hinweg gespeichert werden, ergibt sich eine nicht mehr überschaubare Übersicht der vorhandenen Daten. Werden zudem nachträglich Änderungen hinzugefügt, sind diese zudem an der richtigen Stelle einzufügen, da diese innerhalb der App Zeile für Zeile ausgelesen werden. Somit wird für die Datenverwaltung auf ein Datenbank-Modell umgestiegen. Aufgrund der einfach zugänglichen Nutzbarkeit von SQLite, wird diese als Speichermodell für die interne Datenspeicherlogik genutzt. Als fester Bestandteil der Laufzeitumgebung von Android sind eine Vielzahl an vorgefertigten Funktionen zur Kommunikation zwischen Anwendung und SQLite-Datenbank vorhanden.

Allerdings ergibt sich durch die Implementierung von einer SQLite-DB-Anbindung ein erheblicher Mehraufwand in der Entwicklungsdauer. SQLite-Librarys stellen zwar Funktionen zur Interaktion mit der SQLite-DB selbst zur Verfügung, diese müssen allerdings innerhalb der App erst als nutzbare Funktionen implementiert werden. Über die genaue Herangehensweise und Problematik wird konkret in Kapitel 4.1 eingegangen.

3.2.2 Speichern von Daten außerhalb der Applikation

Das Speichern außerhalb der App wird, wie in der Motivation erwähnt, über den Hochschul-Oracle-Server realisiert. Diese wurde bereits im Praktikum zu diesem Prüfungsfach ausgiebig genutzt, wodurch zusätzliche Einarbeitungszeiten in andere Datenbank-Programme entfallen. Zudem besteht ein Zugriff auf die Datenbank über eine hochschulinterne Benutzerkennung, wodurch das eventuelle kostenpflichtige Beziehen von anderen Datenbank-Strukturen entfällt.

Die Anbindung an den Oracle-Server der Hochschule erfolgt direkt durch die prototypische App. Allerdings muss für den Zugriff auf eine externe Datenbank der entsprechende Treiber für die externe Datenbank implementiert werden. [3] Dafür wird die Java Database Connectivity (JDBC) eingesetzt. JDBC ist eine Standard-Schnittstelle zwischen einem Java-Programm und einer Datenbank eines beliebigen Herstellers. [3] Für den Zugriff auf die externe Datenbank

werden SQL-Befehle genutzt. Innerhalb der externen Datenbank werden die SQL-Befehle verarbeitet und entsprechen Aktion, z.B. das Erstellen von Tabellen oder Abfragen von Inhalten aus Tabellen, durchgeführt. Die Abfrageergebnisse werden der App durch die JDBC-Schnittstelle bei Implementierung der benötigten Funktionen mitgeteilt. Hier gilt das gleiche Prinzip wie schon bei SQLite. Der JDBC-Treiber stellt die benötigten Funktionen, wie z.B. Verbindungsbefehle, das Senden oder Abholen von Daten, bereit – diese müssen allerdings innerhalb der Applikation implementiert und sinnvoll eingebunden werden.

3.3 Resultierende Konzeptlösung

In der Abbildung 3-3 ist das resultierende Konzept der zu entwickelnden App dargestellt. Im linken Teil der Abbildung ist der bereits bestehende Teil aus der ISDP-App zu sehen. In dieser Hausarbeit wird der Teil aus dem unteren Bereich der Abbildung, also das Implementieren einer internen SQLite-Datenbank und der rechte Teil, also das Implementieren einer JDBC-Treiberanbindung an Oracle bearbeitet.

Abbildung 3-3 Resultierendes Konzept der Applikation

Sensordaten werden in einem Arduino-Board aufgenommen und mittels Bluetooth an ein Smartphone übermittelt. Unterhalb des Smartphones kennzeichnet die Verbindung zu SQLite die interne Kommunikation mit der SQLite-Datenbank. Ankommende Daten werden in zu er-

stellende Tabellen-Strukturen abgelegt oder aktualisiert und sind von der App abruf- und darstellbar, so wie bereits bei der Einführung in die Thematik erklärt wurde. Diese Daten sollen nun über die JDBC-Schnittstelle an den Oracle-Server übermittelt werden können. Diese Verbindung umfasst sowohl das Senden als auch das Empfangen von Daten zwischen der App und Oracle. Dafür werden ebenfalls Funktionen zum Eintragen, Aktualisieren, Löschen und Suchen von Einträgen in Oracle innerhalb der App implementiert.

Durch dieses Konzept ist das Projekt prinzipiell ohne größere Einschränkungen skalierbar. Weitere App-Benutzer könnten so z.B. Sensordaten über eine Bluetooth-Verbindung mit anderen Geräten abrufen und diese mit der Oracle-Datenbank synchronisieren, z.B. über WLAN. Wie bereits erwähnt, werden ISDP-App und prototypische App nicht im Rahmen dieser Hausarbeit ineinander integriert, sodass das obige Konzeptbild nicht die finale App-Version repräsentiert.

Im nachfolgenden Kapitel wird nun konkret auf die Implementierung der SQLite- und JDBC-Schnittstellen eingegangen.

4 Implementierung

Innerhalb dieses Kapitels werden die vorgestellten Konzeptlösungen mittels Android Studio Entwicklungsumgebung in der Programmiersprache Java implementiert. Das Resultat ist dabei eine graphische Bedienoberfläche mit Zugriff auf interne und externe Datenbanken.

Zuerst wird dabei auf das Implementieren einer Datenbank mit der SQLite-Standardbibliothek eingegangen. Es werden exemplarisch der einzelne Aufbau und Funktionen der jeweiligen Klasse beschrieben. Anschließend wird thematisiert, wie der JDBC-Treiber in die Android Studio Entwicklungsumgebung eingebunden und die bereitgestellten Funktionalitäten innerhalb der Applikation genutzt werden. Abschließend werden die erstellten Applikationsoberflächen präsentiert und der Funktionsumfang abgegrenzt. Abgeschlossen wird das Kapitel mit einer kurzen Übersicht über das genutzte Test-Verfahren während der Entwicklung.

Die Übersicht über prinzipielle Abhängigkeiten zwischen den einzelnen Klassen ist in der Abbildung 4-1 auf der nächsten Seite dargestellt. Dabei wird zusätzlich zwischen Frontend und Backend unterschieden.

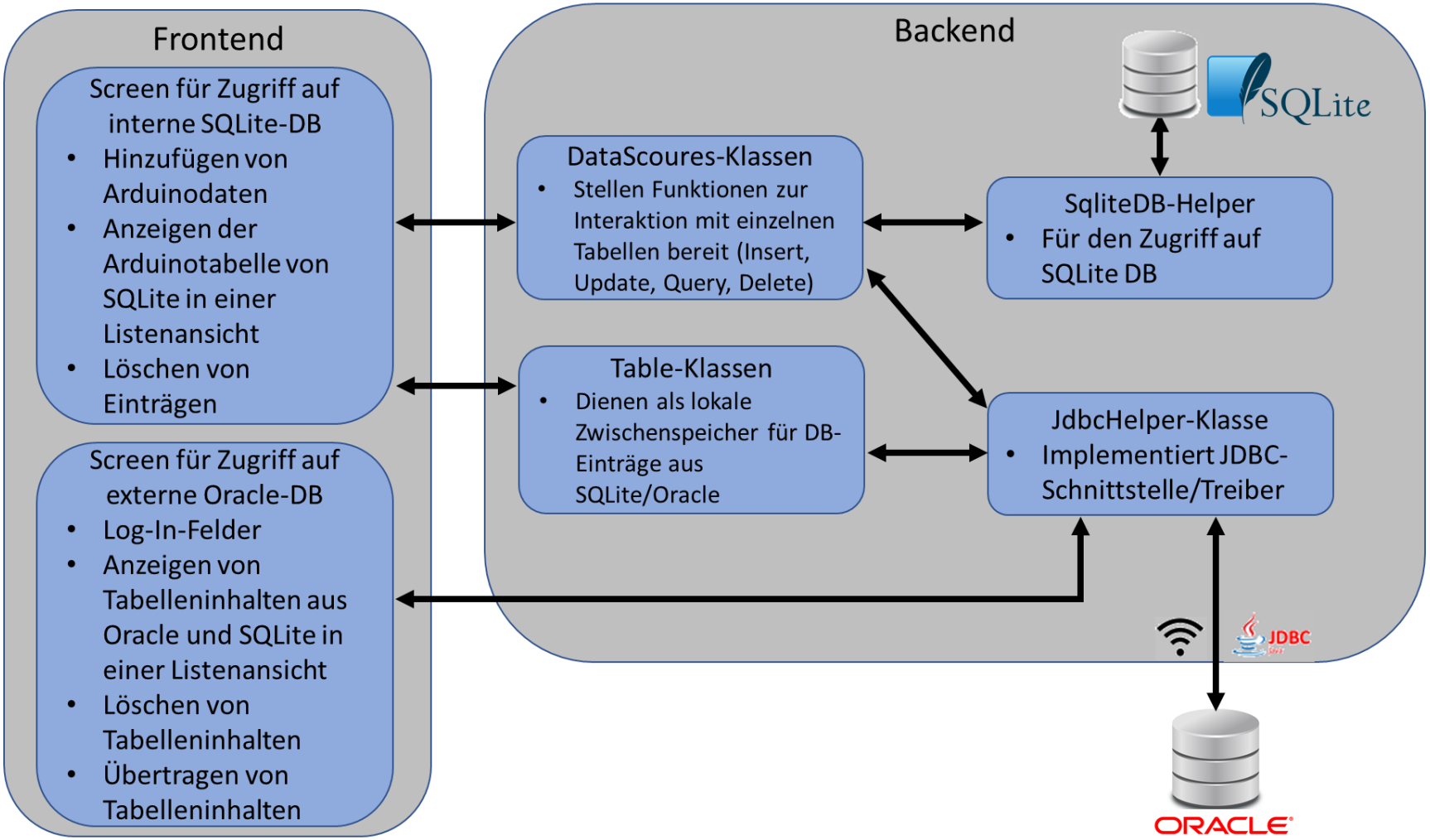

Abbildung 4-1 Frontend und Backend der Prototyp-App

4.1 SQLite-Implementierung

Die Nutzung von SQLite als Standard-Bibliothek für eine Datenbank-Struktur gilt in Java als komfortabel und hat eine große Nutzer-Community. Daher wird sie für die Implementierung einer internen Datenbank-Struktur genutzt. Durch SQLite werden Funktionen bereitgestellt, womit das Erstellen, Löschen, Aktualisieren, Suchen und einigen weiteren Funktionen von Datenbanken und Tabellenstrukturen innerhalb einer Java-Applikation ermöglicht wird.

Die SQLite-Struktur dieser Applikation besteht grundlegend aus drei Bereichen. Diese werden konzeptionell in der Abbildung 4-2 dargestellt:

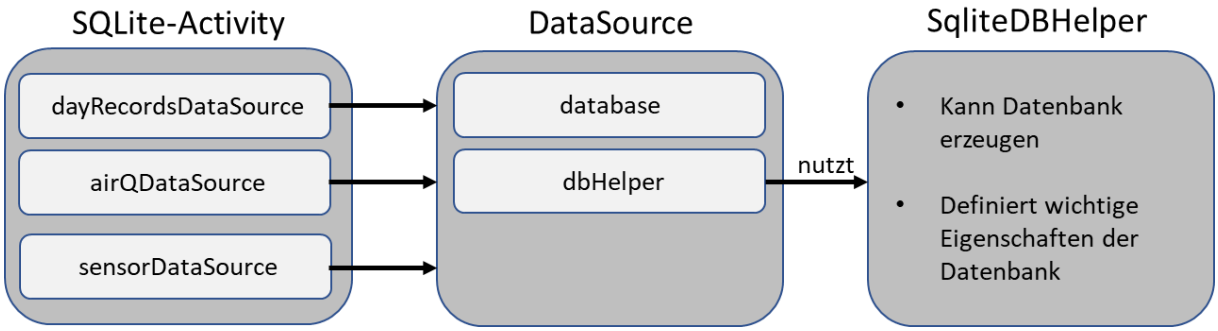

Abbildung 4-2 Konzeptioneller Aufbau der Klassen für die SQLite-Implementierung

Der SQLiteDBHelper ist dabei eine Hilfsklasse, die zum Erstellen von Datenbanken genutzt wird. Weiterhin werden in ihr Funktionen implementiert, die während der Laufzeit genutzt werden können/müssen. Die DataSource-Klassen sind Helferklassen für den Zugriff auf Tabellen innerhalb der SQLite-Datenbank. Funktionen können hierbei das Löschen, Einfügen, Suchen oder Aktualisieren von Tabelleneinträgen sein. In Activities (Benutzeroberflächen von Android) werden diese Helfer-Klassen dann implementiert, um so Zugriff auf Datenbank-Inhalte zu erhalten. Nachfolgend werden der SQLiteDBHelper und DataSource-Klassen exemplarisch erläutert. Angefangen wird dafür mit Hilfsklassen zum Zwischenspeichern von Einträgen aus Datenbanken.

4.1.1 Table-Klassen

Bevor eine eigene Datenbank-Algorithmen entworfen werden können, bedarf es zuerst der Überlegung wie Daten aus dieser Datenbank gelesen oder über Schnittstellen geschrieben werden sollen. Beispielhaft ist z.B. das Auslesen von Tabelleninhalten nur Zeile für Zeile möglich. Der Entwickler kann nun wählen, ob er diese Zeilen in einen String-Builder einfügt oder eine andere Struktur in Erwägung zieht. Im Rahmen dieser Arbeit werden statt der Verwendung von

einzelnen Strings Klassen erstellt, die dem Inhalt jeder Tabellenstruktur entsprechen. Diese werden allgemein Table-Klassen genannt. Dafür sei auf das relationale Schema aus Abbildung 3-2, sowie auf das EER-Modell verwiesen. Durch diese Klassen können Tabelleninhalte aus den Datenbanken zum Beispiel in Listen zwischengespeichert werden. So wird für einen Arduino-Eintrag ein Objekt erstellt, das als Attribute Name und Mac-Adresse enthält, so wie in der nachfolgenden Abbildung dargestellt ist.

```
public class AirQ {
    private String sMac;
    private String sLabel;
    private long id;
    public AirQ(String sMac, String sLabel, long id) {
        this.sMac = sMac;
        this.sLabel = sLabel;
        this.id = id;
    }
    public String getMac() { return sMac; }
    public String getLabel() { return sLabel; }
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }
    @Override
    public String toString() {
        String output = sLabel + "  " + sMac;

        return output;
    }
```

Abbildung 4-3 Aufbau einer Table-Klasse im Android Studio Projekt

Die Attribute aller Table-Klassen sind zusätzlich in der nachfolgenden Abbildung dargestellt und entsprechen, wie bereits erwähnt, der Struktur des EER-Modells.

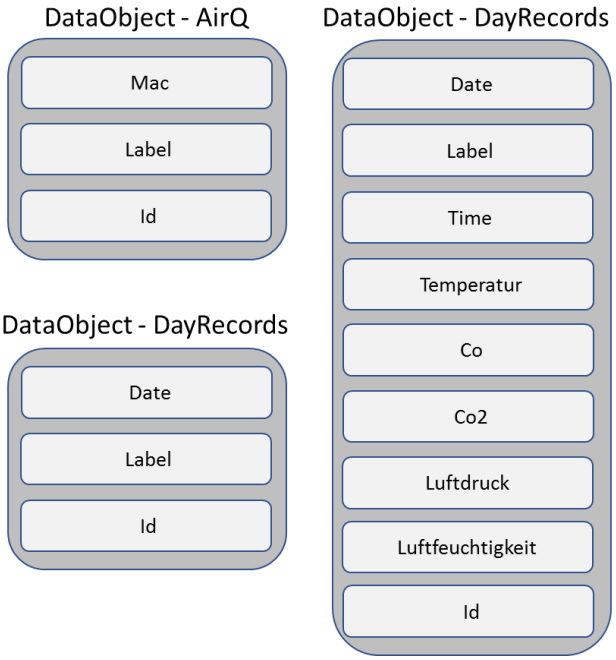

Abbildung 4-4 Table Klassen innerhalb der Applikation

Die Id ist dabei die Zeilenummer in der SQLite-DB. Diese wird automatisch während der Laufzeit generiert und hat vorerst keine weitere Bedeutung.

4.1.2 SQLiteHelper-Klasse

Die SQLite-Helferklasse wird genutzt, um die Datenbank-Verbindung einrichten und verwalten zu können. Mit ihr werden die Datenbank, sowie Tabellenstrukturen vorgegeben und erstellt.

```
// Die onCreate-Methode wird nur aufgerufen, falls die Datenbank noch nicht existiert
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        Log.d(LOG_TAG, "Die Tabelle wird mit SQL-Befehl: " + SQL_CREATE + " angelegt.");
        db.execSQL(SQL_CREATE);
        Log.d(LOG_TAG, "Die Tabelle wird mit SQL-Befehl: " + SQL_CREATE1 + " angelegt.");
        db.execSQL(SQL_CREATE1);
        Log.d(LOG_TAG, "Die Tabelle wird mit SQL-Befehl: " + SQL_CREATE2 + " angelegt.");
        db.execSQL(SQL_CREATE2);
    }
    catch (Exception ex) {
        Log.e(LOG_TAG, "Fehler beim Anlegen der Tabelle: " + ex.getMessage());
```

Abbildung 4-5 Funktion zum Erstellen von Tabellen-Strukturen

Die onCreate-Funktion aus Abbildung 4-5 erstellt Tabellenstruktur innerhalb der Datenbank. Dazu werden vordefinierte Strings, die der Tabellenstruktur aus dem relationalen Schema entsprechen mit einem execute-SQL-Befehle an die Datenbank gesendet. Die Strings entsprechen klassischen SQL-Statements zur Erstellung von Tabellen. Diese sind in Anhang B Abbildung zum SQ-Lite Helfer dargestellt. Die Tabellenstruktur wird auch in der externen Datenbank verwendet werden. Innerhalb dieser SQLite-Strukturen sind Fremdschlüssel allerdings standardmäßig deaktiviert. Abbildung 4-6 zeigt den Programmausschnitt zum Aktivieren von Fremdschlüsselbeziehungen:

```
@Override
public void onOpen(SQLiteDatabase db) {
    if (!db.isReadOnly()) {
        super.onOpen(db);
        // Enable foreign key constraints
        db.execSQL("PRAGMA foreign_keys=ON");
    }
}
```

Abbildung 4-6 Aktivierung von Fremdschlüsseln innerhalb der Datenbank

Beim Öffnen einer Datenbank für Schreibvorgänge muss zusätzlich jedes Mal das SQL-Statement aus der Abbildung abgesetzt werden, um alle Fremdschlüssel zu aktivieren, da diese Einstellung nach Schließen der SQLite-Datenbank-Struktur jedes Mal deaktiviert wird.

4.1.3 DataSources-Klassen

Die Data-Sources-Hilfsklassen dienen zur Verwaltung von Tabellen-Einträgen. Somit wird für Arduinos, Tagesdaten und Sensordaten jeweils eine DataSource-Helferklasse genutzt, wie im vorherigen Unterkapitel erwähnt.

Anhand der Helfer-Klasse zur Verwaltung der Arduino-Tabellendaten werden die Funktionalitäten exemplarisch erläutert. Diese unterscheiden sich zwischen den Tabellen größtenteils nur durch Übergabeparameter oder genutzt Variablen. Für jede Funktion muss eine try/catch Struktur eingehalten werden, da Datenbank-Zugriffsfehler ansonsten zum Absturz des Programms führen.

Die Anbindung an die SQLite-Datenbank und damit die Nutzbarkeit der Datenbank-Funktionalität erfolgt in jeder DataSource-Helferklasse gleich gemäß folgender Abbildung:

```
private SQLiteDatabase database;
public void open() {
    Log.d(LOG_TAG, "Eine Referenz auf die Datenbank wird jetzt angefragt.");
    database = dbHelper.getWritableDatabase();
    Log.d(LOG_TAG, "Datenbank-Referenz erhalten. Pfad zur Datenbank: " + database.getPath());
}
```

Abbildung 4-7 Anbindung der DataSource-Klassen an die SQLite-Datenbank

Ein vom SQLiteDatabase erstelltes Objekt wird über den open Befehl der jeweiligen DataSource aufgerufen. Dieser open-Befehl muss später innerhalb der Applikation genutzt werden, bevor ein Datenbank-Zugriff möglich ist. Innerhalb des open-Befehls wird über das dbHelper-Objekt vom Typ der SQLiteHelper-Klasse der Datenbank-Pfad hinterlegt. Dadurch erhält das database-Objekt sämtliche von SQLite implementierten Funktionen zur Interaktion mit der Datenbank und die darin befindlichen Tabellen.

CreateAirDataSource zum Einspeichern eines neuen Arduino-Eintrags

In der Abbildung 4-8 wird die implementierte CreateAirDataSource Funktion zum Hinzufügen eines Arduino-Tabelleneintrags erläutert:

```
public boolean createAirQDataSource(String sMac, String sLabel) {
    try{
        ContentValues values = new ContentValues();
        values.put(IsdpDbHelper.COLUMN_MAC, sMac);
        values.put(IsdpDbHelper.COLUMN_LABEL, sLabel);
        database.insertOrThrow(IsdpDbHelper.TABLE_AIRQ_LIST, null, values);
        return true;
    }
    catch(SQLiteException sq)
    {
        Log.d(LOG_TAG, sq.getLocalizedMessage());
        return false;
    }
}
```

Abbildung 4-8 Funktion zum Einfügen eines neuen Tabelleneintrags im AirQ Table

Als Übergabeparameter werden die einzutragende MAC-Adresse und der einzutragende Name übergeben. Die ContentValues-Klasse wird als Speicherstruktur genutzt, die direkt in einen SQLite-Befehl übergeben werden kann. Der put-Befehl hinterlegt eine Kombination aus Spaltennamen und zugehörigem Wert. So z.B. wie oben abgebildet den Spaltennamen für die Mac-Adresse und die übergebene MAC-Adresse selbst. Diese werden über das database-Objekt in

die angegebene Tabelle übergeben. Der insertOrThrow-Befehl fügt den Eintrag nur hinzu, sofern dieser nicht vorhanden ist – ansonsten wird der Eintrag verworfen. Würde hier lediglich der insert-Befehl genutzt werden, stürzt das Programm in jedem Fall ab, wenn eine Unique-Constrait-Verletzung beim Eintrage-Versuch vorliegt. Sollte eine Fehlermeldung durch ein catch aufgenommen werden, erfolgt das Ausgeben über den LogCat-Monitor (auf diesen wird im Test-Kapitel genauer eingegangen).

UpdateAirQDataSource Funktion zum Aktualisieren von Einträgen

Die Funktion zum Aktualisieren einzelner Eintragsdaten wird nachfolgend abgebildet:

```
public boolean updateAirQDataSource(String sMac, String sLabel) {
    try {
        ContentValues values = new ContentValues();
        values.put(IsdpDbHelper.COLUMN_MAC, sMac);
        int iResponse = database.update(IsdpDbHelper.TABLE_AIRQ_LIST, values,
                IsdpDbHelper.COLUMN_LABEL + " ='" + sLabel + "'", null);
        return iResponse > 0 ? true : false ;
```

Abbildung 4-9 Funktion zum Aktualisieren von Einträgen

Neu ist hierbei die Update-Funktion des database-Objekts. Hierbei werden dessen Übergabe-Parameter in ein passendes SQL-Statement umgewandelt. Vereinfacht können die Übergabe-gabe-Parameter der Reihe nach wie folgt beschrieben werden: Update in der Tabelle „Table_AirQ_List mit den Werten „values“, wo der Spaltenname dem Übergabeparameter „sLabel“ entspricht. Klappt dieser Vorgang wird mit einem true quittiert. Ansonsten erfolgt die Rückgabe eines false-boolean.

DeleteAirQ Funktion zum Löschen von Arduino-Einträgen

Zum Löschen von Einträgen in der Arduino-Tabelle wird die nachfolgende Funktion genutzt:

```
public boolean deleteAirQ(String label) {
    try{
        database.delete(IsdpDbHelper.TABLE_AIRQ_LIST,
                IsdpDbHelper.COLUMN_LABEL + " ='" + label +"'",
                null);
        Log.d(LOG_TAG, "Eintrag gelöscht! Arduino: " + label);
        return true;
```

Abbildung 4-10 Funktion zum Löschen von Tabellen-Einträgen

Der delete-Befehl ähnelt dem Befehl zum Aktualisieren von Einträgen, hier wird jedoch auf einzuspeichernde Parameter verzichtet, da der Eintrag lediglich gelöscht werden soll.

SearchForAirQ Funktion zum Finden von Einträgen

Die Such-Funktion besitzt die Übergabeparameter Spaltenname und Spaltenwert. Mit einem query-Befehl wird der Inhalt der Arduinotabelle in der übergebenen Spalte nach dem übergebenen Spaltenwert gesucht, s. Abbildung 4-11.

```
public boolean searchForAirQ(String sColumnName, String sValue) {
    try {
        Cursor cursor = database.query(IsdpDbHelper.TABLE_AIRQ_LIST,
                columns, sColumnName + " ='" + sValue + "'", null, null, null, null);
        return cursor.moveToFirst() ? true : false;
```

Abbildung 4-11 Suchfunktion zum Finden eines oder mehrerer Einträge

Wird ein Eintrag über die query-Funktion lokalisiert, quittiert die Funktion dies mit einem Rückgabewert vom Typ Cursor. Werden Einträge zu der gesetzten Bedingung gefunden, erfolgt das Speichern dieser Informationen in dem Cursor Objekt. Dabei spielt es keine Rolle ob ein Eintrag oder beliebig viele hinterlegt werden.

CursorToAirQ zum Speichern von Zeileneinträgen der AirQ-Tabelle

Zur Beschreibung der darauffolgenden Funktion muss zunächst eine Cursor-Hilfs-Funktion beschrieben werden. Diese wird nachfolgend dargestellt:

```
private AirQ cursorToAirQ(Cursor cursor) {
    int idIndex = cursor.getColumnIndex(IsdpDbHelper.COLUMN_ID);
    int idMac = cursor.getColumnIndex(IsdpDbHelper.COLUMN_MAC);
    int idLabel = cursor.getColumnIndex(IsdpDbHelper.COLUMN_LABEL);
    String sMac = cursor.getString(idMac);
    String sLabel = cursor.getString(idLabel);
    long id = cursor.getLong(idIndex);
    AirQ airQ = new AirQ(sMac, sLabel, id);
    return airQ;
}
```

Abbildung 4-12 Erstellen eines Table-Objekteintrags mithilfe der Cursor-Funktion

In dieser Cursor-Funktion wird ein Cursor übergeben. Geeignet dafür ist z.B. der Cursor einer Suchfunktion. Ein Cursor wird grundsätzlich in SQLite dafür genutzt, Zeilen nach und nach abzuarbeiten. Der Cursor beinhaltet dabei die eingetragenen Spaltendaten einer jeweiligen Zeile. Der Cursor durchläuft die Tabelle Zeile für Zeile und filtert die entsprechenden Werte zu den entsprechenden Spalten. Diese Wert werden in einem AirQ-Objekt gespeichert.

GetAllAirQs Funktion zur Rückgabe einer Liste mit vorhandenen Einträgen

Mit der in Abbildung 4-13 dargestellten getAllAirQs-Funktion wird eine Liste mit allen Einträgen des Arduino-Tables zurückgegeben. Dabei wird wie bereits beschrieben, zuerst ein Cursor zugeordnet, der alle Einträge zum jeweiligen Suchbefehl beinhaltet.

```
public List<AirQ> getAllAirQs() {
    List<AirQ> AirQList = new ArrayList<>();
    Cursor cursor = database.query(IsdpDbHelper.TABLE_AIRQ_LIST,
            columns, null, null, null, null, null);
    cursor.moveToFirst();
    AirQ airQ;
    while(!cursor.isAfterLast()) {
        airQ = cursorToAirQ(cursor);
        AirQList.add(airQ);
        Log.d(LOG_TAG, "ID: " + airQ.getId() + ", Inhalt: " + airQ.toString());
        cursor.moveToNext();
    }
    cursor.close();
    return AirQList;
}
```

Abbildung 4-13 Funktion zum Erhalt aller Einträge einer Tabelle

Dieser wird an die erste Position gesetzt und anschließend iterativ durchlaufen. Innerhalb jedes Iterationsschritts wird die vorher erklärte CursorToAirQ Funktion aufgerufen und ein Objekt mit den Informationen des Zeilen-Eintrags zurückgegeben. Dieses Objekt wird einer Liste vom selben Typ übergeben und nach Iterationsschluss als Rückgabewert übergeben.

4.2 JDBC-Implementierung

Nachfolgend werden die notwendigen Schritte zur erfolgreichen Implementierung des JDBC-Treibers vorgestellt. Das umfasst zum einen das Einbinden des JDBC-Treibers in Oracle selbst, sowie das Erstellen einer JDBC-Schnittstelle zum Kommunizieren zwischen interner SQLite-Datenbank und externer Oracle-Datenbank.

4.2.1 Einbinden des JDBC-Treibers

Der Zugriff auf die externe Datenbank in Form des Oracle-Servers erfolgt über einen JDBC-Treiber. Im Rahmen der Projektarbeit wurde Version 6 des Treibers von dem betreuenden Dozenten zur Verfügung gestellt. Um diesen Treiber zu nutzen, wurde dieser über das Einstellungsmenü von Android Studio als .jar-Datei zuerst in das Projekt eingebunden. Dadurch ergibt sich folgende Abbildung bei den Abhängigkeiten der App-Struktur:

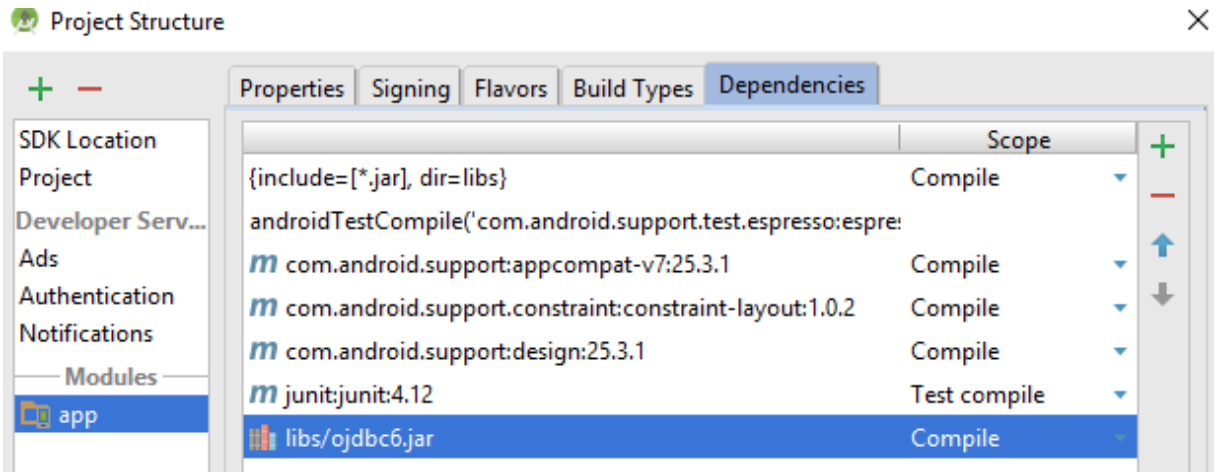

Abbildung 4-14 Einbinden des JDBC-Treibers Version 6 unter Android Studio

Dabei ist darauf zu achten, dass im Gradle-Builder eine Abhängigkeit für diese Bibliothek erstellt wurde. Gradle ist ein auf Java basiertes Build-Management Automatisierungstool, welches als Tool zum Builden von Frameworks genutzt wird. Somit können Plugins, wie zum Beispiel die JDBC-Bibliothek in ein Android Studio Projekt eingebunden werden. Im Gradle-Builder für die Applikation lautet der genaue Gradle-Befehl zum Einbinden der JDBC-Treiber-Schnittstelle: `compile files('libs/ojdbc6.jar')`

4.2.2 JDBC-Schnittstelle

In der App wird die JDBC-Schnittstelle in Form einer sogenannten Helfer-Klasse implementiert, nachfolgenden JDBC-Helper genannt. Diese Klasse bildet die Schnittstelle zu den beschriebenen DataSource-Klassen der internen SQLite-DB und zu der externen Oracle-DB. Für

den Zugriff auf die SQLite-Datenbank werden demzufolge die implementierten Funktionen der DataSource-Klassen verwendet.

Der JDBC-Helper nutzt gemäß Konzeptlösung den JDBC-Treiber, um eine Verbindung zu der Oracle-Datenbank herzustellen. Abbildung 4-15 zeigt den Konzeptlösungsbereich, in dem dieser aktiv genutzt wird. Dazu wird auch auf die Funktionen der DataSource-Helferklassen aus dem vorherigen Unterkapitel zugegriffen.

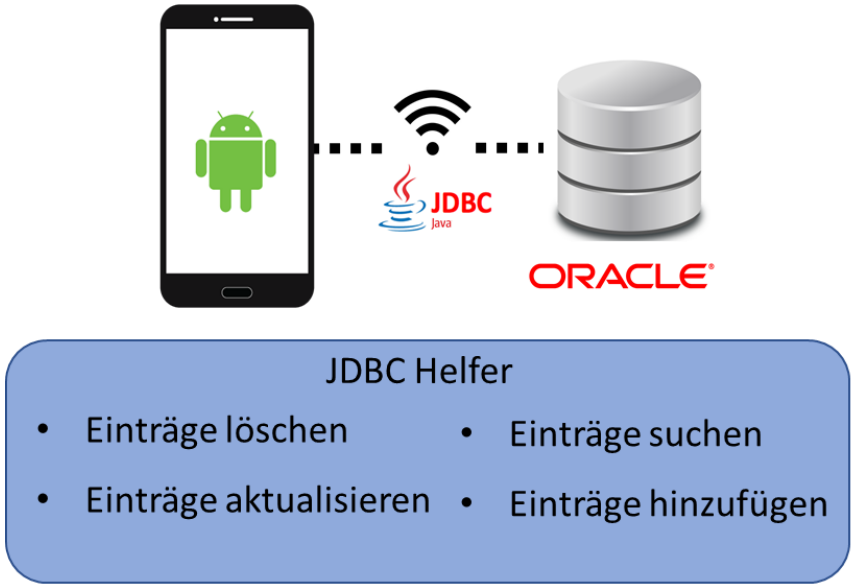

Abbildung 4-15 JDBC-Helfer für die Komponente zur Oracle-Verbindung

Dabei stellt der JDBC-Helper prinzipiell die gleichen Funktionen bereit (Löschen, Suchen, Aktualisieren und Hinzufügen von Einträgen), die schon in den DataSource-Klassen implementiert wurden – in diesem Fall allerdings für die Kommunikation mit der Oracle-Datenbank. Nachfolgend werden einzelne Funktionsausschnitte abgebildet und erklärt, um den grundsätzlichen Aufbau der JDBC-Helfer Klasse nachvollziehbar zu machen. Dabei wird sich darauf bezogen, wie die Behandlung der AirQ-Tabelleneinträge aus Oracle erfolgt, da die prinzipielle Syntax für die anderen zwei Tabellen größtenteils identisch ist und den Dokumentationsumfang meist nur quantitativ erweitern würde.

Connect Funktion zum Verbinden mit einer Datenbank

In der nachfolgenden Abbildung wird die implementierte Connect Funktion erläutert:

```
public boolean connect(String sdriver, String sserver, String susername, String spasswort) {
    try {
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitNetwork().build();
        StrictMode.setThreadPolicy(policy);
        // Oracle JDBC driver laden
        Class.forName(sdriver);
        // Verbindung zur HS-Datenbank
        conn = DriverManager.getConnection(sserver, susername, spasswort);
        statement = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
        Log.d(LOG_TAG, "Verbindung zum Server wird hergestellt.");
        return true;
    } catch (SQLException es) {
        problem(es);
        return false;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return false;
    }
}
```

Abbildung 4-16 Connect Funktion zum Herstellen einer Verbindung zwischen App und Oracle-DB

Übergabeparameter sind hierbei der Treiber, in diesem Fall der Oracle-Treiber, der Servername der Hochschule Osnabrück für den externen Zugriff, sowie die Benutzererkennung. Die Klasse StrictMode gibt das Thread-Handling vor und wurde von der Vorlage übernommen. Das Objekt conn ist vom Typ Connection und beinhaltet die Verbindungsinformationen der angelegten Session über die DriverManager.getConnection Funktion. Das Objekt statement vom Typ Statement wird über die createStatement Funktion mit parametrisierten SQL-Einstellungen gespeist. Jede Art von Kommunikation über die JDBC-Schnittstelle benötigt ein sogenanntes try/catch Handling, damit eventuelle Fehler aufgefangen werden die sonst zum Absturz des Programms führen würden. Wurde die Verbindung hergestellt, ist eine LOG-Meldung im LogCat Fenster sichtbar.

InsertAirQOracle Funktion zum Einfügen eines AirQ-Eintrages in Oracle

In der nachfolgenden Abbildung wird die implementierte InsertAirQOracle Funktion erläutert:

```
public void insertAirQOracle(String slabel, String smac){
    if (conn == null) {
        Log.d(LOG_TAG,"keine Verbindung vorhanden");
        return;
    }
    try {
        String sInsert =  "INSERT INTO " + IsdpDbHelper.TABLE_AIRQ_LIST
                + " (" + IsdpDbHelper.COLUMN_LABEL + ", " + IsdpDbHelper.COLUMN_MAC + ")"
                + " Values " + "('"+ slabel+ "', '"+ smac + "')";
        Statement stmt = conn.createStatement();
        Log.d(LOG_TAG, sInsert); // zur Info
        stmt.executeUpdate(sInsert);
        conn.commit();
        Log.d(LOG_TAG, "Daten werden in "+ IsdpDbHelper.TABLE_AIRQ_LIST +"eingefuegt");
    }
    catch (SQLException e) {
        problem(e);
    }
}
```

Abbildung 4-17 Funktion zum Einfügen eines Eintrags in den AirQ-Table der Oracle-DB

Sofern eine Verbindung vorhanden ist, wird ein SQL-Statement generiert, welcher der Oracle-Syntax entspricht und als Inhalt die während der Laufzeit statisch angelegten Spaltennamen des vorhandenen AirQ-Tables abruft und die Übergabeparameter der einzutragenden Werte enthält (exemplarisch hier für den AirQ Eintrag Namen und Mac-Adresse).

Abgespeichert als String, wird dieser über das Statement Objekt mit der JDBC-Funktion executeUpdate an Oracle weitergeleitet. Über die commit Funktion wird die Änderung anschließend in der Oracle-Datenbank festgeschrieben.

SearchAirQOracle Funktion zum Suchen eines Eintrages im AirQ-Oracle Table

In Abbildung 4-18 wird ein Ausschnitt aus der implementierten SearchAirQOracle Funktion erläutert. Da der Code rund um die try/catch Syntax ähnlich zu der Insert-Funktion aufgebaut ist, wird nur der nicht behandelte Teil vorgestellt. Neben dem Absetzen eines SQL-Befehls gibt es in Form der Klasse ResultSet eine entscheidende Neuerung. Das Resultset Objekt rs speichert nun den Rückgabewert der executeQuery Funktion, welcher die geforderten Einträge aus dem SQL-String beinhaltet. Ist ein Eintrag vorhanden, wird der Wert returnValue auf true gesetzt und am Ende der Funktion zurückgegeben.

```
try {
    Statement stmt = conn.createStatement();
    String squery = "Select "+IsdpDbHelper.COLUMN_LABEL + " from "
            +IsdpDbHelper.TABLE_AIRQ_LIST
            + " where " + IsdpDbHelper.COLUMN_LABEL + "='"+svalue +"'";
    ResultSet rs = stmt.executeQuery(squery);
    //Ergebnisausgabe
    while (rs.next()) {
            Log.d(LOG_TAG, rs.getString(IsdpDbHelper.COLUMN_LABEL) + " ");

        if (svalue.contentEquals(rs.getString(IsdpDbHelper.COLUMN_LABEL))) {
            Log.d(LOG_TAG, svalue);
            returnValue=true;
        }
    }
```

Abbildung 4-18 Code-Ausschnitt zum Suchen eines Eintrages in Oracle

Auf die Funktion zum Löschen von Einträgen wird an dieser Stelle nicht näher eingegangen, es sei nur so viel gesagt, dass der Statement Befehl nun: `stmt.executeUpdate(sDelete);` lautet. Wobei der String sDelete wie schon beim Einspeichern und Suchen einen SQL-Befehl für Oracle beinhaltet.

Für alle weiteren Funktionen zum Löschen, Eintragen oder Suchen von den verschiedenen Tabellen (AirQs, Tagesdaten, Sensordaten) sei auf den beigefügten Projektcode verwiesen, da sich diese meist nur marginal in der Syntax und größtenteils in der Anzahl und den Namen der Übergabeparameter unterscheiden.

LoadToOracle Funktion zum Übertragen aller Einträge der SQLite-DB zur Oracle-DB

Zur korrekten Übertragung der Daten ist eine bestimmte Sendereihenfolge einzuhalten. Zuerst muss immer die Tabelle der Arduinodaten aktualisiert werden, danach die Tabelle der Tagesdokumente und zuletzt die Tabelle mit den Sensordaten. Der Ablauf ist dabei für alle Tabellen grundsätzlich gleich, weswegen hier lediglich die AirQ-Tabellen-Funktionen abgebildet werden.

In der Abbildung 4-19 wird die implementierte loadToOracle Funktion erläutert:

```
public void loadToOracle() {


    List<AirQ> AirQList = airQDataSource.getAllAirQs();
    Iterator<AirQ> airQIterator = AirQList.iterator();
    int iCount = 0;
    while(airQIterator.hasNext())
    {
        if (!searchAirQOracle(AirQList.get(iCount).getLabel()))
            insertAirQOracle(AirQList.get(iCount).getLabel(), AirQList.get(iCount).getMac());
        airQIterator.next();
        iCount++;
    }
```

Abbildung 4-19 Teilausschnitt 1 der Funktion zum Übertragen der SQLite Einträge in die Oracle-DB

Die getAllAirQs Funktion wurde bereits DataSource-Helferklassen erwähnt und gibt eine Liste mit allen vorhandenen AirQ-Objekten zurück. Diese Liste wird im Anschluss mit einem Iterator verknüpft und iterativ bis zum Endpunkt durchlaufen. Dabei wird bei jedem Iterationsschritt geprüft, ob ein AirQ-Eintrag in der Oracle-DB vorhanden ist. Ist dies nicht der Fall, so wird ein Eintrag mit den entsprechenden Informationen aus dem AirQ-Objekt angelegt.

Im Anschluss werden die Tagesdaten kopiert. Hierbei gibt es einen gezielten Unterschied zum AirQ-Ablauf aus der vorherigen Abbildung in der IF-Bedingung im Iterationsverlauf, welcher in Abbildung 4-20 dargestellt wird (die unsaubere Formatierung dient zur besseren Lesbarkeit in der Abbildung).

```
if (!searchDayOracle(DayRecordsListApp.get(iCount).getLabel(),
        DayRecordsListApp.get(iCount).getDate()))
{
    insertDayRecordsToOracle(DayRecordsListApp.get(iCount).getLabel(),
            DayRecordsListApp.get(iCount).getDate());
    DayRecordsForOracle.add
            (new DayRecords(DayRecordsListApp.get(iCount).getDate(),
                    DayRecordsListApp.get(iCount).getLabel(),0));

}
```

Abbildung 4-20 Teilausschnitt 2 der Funktion zum Übertragen der SQLite Einträge in die Oracle-DB

Falls ein Tagesdateneintrag noch nicht vorhanden ist in Oracle-DB, wird eine zusätzliche Liste DayRecordsForOracle mit einem neuen Tagesdokument-Objekt gespeist. Diese Liste wird nachfolgend beim Einfügen neuer Sensordateneinträge in der Oracle-DB abgearbeitet. Dadurch muss nicht jeder einzelne Sensordaten-Wert wie bei den AirQ- und Tagesdateneinträgen iterativ durchlaufen und gesucht werden. Es werden anhand der DayRecordsForOracle Liste direkt die

Sensordaten-Einträge aus den vorher nicht bekannten Tagesdaten-Einträgen übermittelt. Für den 3. Teilausschnitt sei auf den Programmiercode verwiesen, da sich dieser von der Syntax ähnlich zu den ersten beiden Teilausschnitten verhält.

Funktionen zum Einspeichern, Löschen und Aktualisieren von SQLite-Einträgen

Die JDBC-Schnittstelle nutzt die bereitgestellten Funktionen aus den DataSource-Helfer Klassen und stellt damit ein Interface zum Zugriff auf die hinterlegte SQLite-Datenbank für den Anwender dar. Die Funktionen der DataSource-Helfer Klassen wurden bereits in Kapitel 4.1 erläutert und werden anhand der nachfolgenden Abbildung einmalig exemplarisch vorgestellt:

```
public void insertIntoAirQTable(String sMac, String sLabel) {

    if (!airQDataSource.createAirQDataSource(sMac, sLabel)) {
        Log.d(LOG_TAG, "Eintrag in AirQ mit Tag: " + sMac
                + " und Name: " + sLabel + " konnte nicht erstellt werden.");
        return;
    }

}
```

Abbildung 4-21 Funktion zum Hinzufügen eines AirQ Eintrages in die SQLite-DB

Innerhalb der genutzten Bedingung wird geprüft, ob ein AirQ-Eintrag in der internen SQLite-Datenbank vorhanden ist. Ist dies der Fall, so wird eine entsprechende LOG-Meldung im LogCat Monitor ausgegeben.

Nach diesem Verfahren sind alle weiteren Funktionen (Eintragen, Löschen, Aktualisieren, Suchen) für sämtliche Tabellen (AirQs, Tagesdaten und Sensordaten) aufgebaut.

4.3 Benutzeroberflächen in der Applikation

Damit der Benutzer einer Applikation Zugriff auf die dahinterliegenden Funktionen erhält, muss ein graphische Bedienoberfläche implementiert werden. Dies erfolgt bei Android Studio in Form von Activity-Klassen. Activites geben dem Entwickler die Möglichkeit eine graphische Bedienoberfläche zu implementieren, um die Funktionen des Backend-Bereichs graphisch zu nutzen. Nachfolgend werden zwei graphische Bedienoberflächen zur Interaktion mit der prototypischen App präsentiert. Die hinter den Activities liegenden Funktionen entsprechen denen der JDBC-Helferklasse bzw. der DataSource-Helferklassen.

4.3.1 Benutzeroberflächen für den JDBC-Treiber

In der App sind zwei gekoppelte Activites zur Nutzung des JDBC-Treibers vorhanden. Diese Activities nutzen graphische Bedienelemente hinter denen Funktionen der JDBC-Helferklasse liegen. Die Funktionalität der einzelnen Interaktionselemente für den Benutzer mit der App werden im Folgenden anhand der Bildschirme dieser Activities erläutert.

In der Abbildung 4-22 ist der Screen zur Anmeldung zum Oracle-Server abgebildet. In diesem Screen wird zur Anmeldung an die Oracle Datenbank der Username, z.B. ospikulik und das entsprechende Passwort eingegeben. Zur Eingabe der Benutzerdaten wird eine Tastatur aufgerufen. Mit dem Button „Log IN“ wird der Benutzer im Oracle-Server anmeldet, sofern die Nutzerkennung gültig ist.

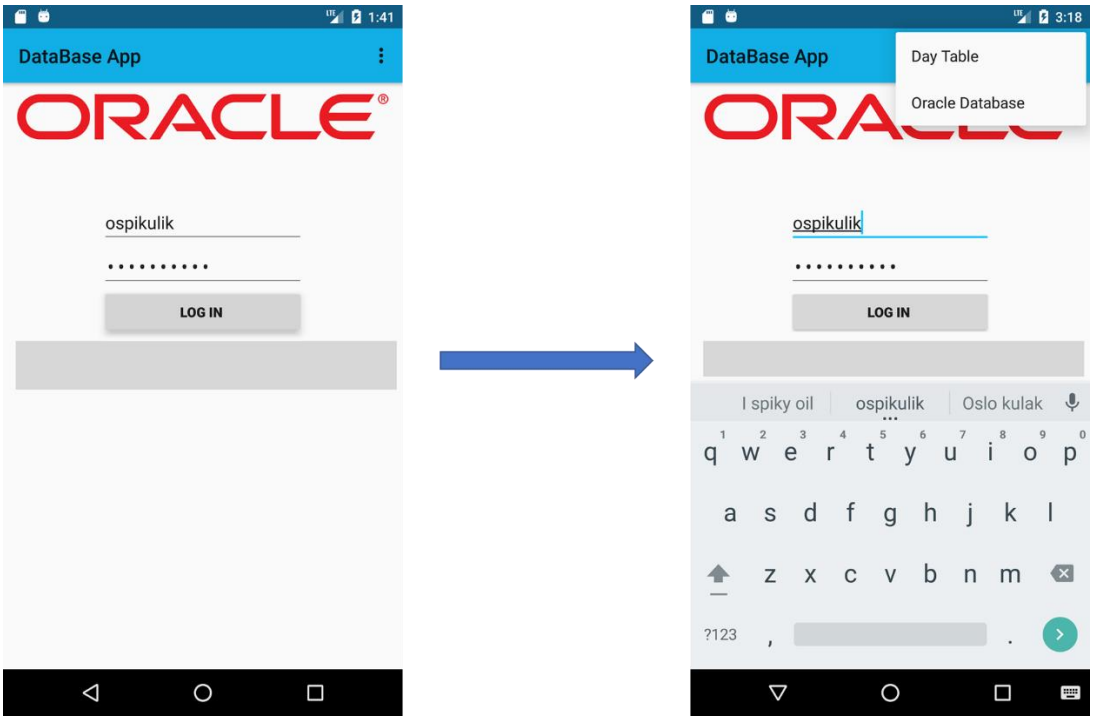

Abbildung 4-22 Screen zur Anmeldung an den Oracle-Server

Das graue Textfeld unter dem Log IN Button beinhaltet eine LOG-Meldung über den Versuch zur Herstellung einer Verbindung zum Oracle-Server. Mit dem Optionsmenü im rechten oberen Bereich, gekennzeichnet durch drei waagerecht angeordnete Punkte, kann zwischen zwei Activities gewechselt werden. Dabei stellt „Oracle Database“ den oben dargestellten Bildschirm dar, der Bildschirm DayTable wird im nachfolgenden Unterkapitel erläutert.

Gelingt die Authentifizierung mit dem Oracle-Server, gelangt der Benutzer zur zweiten gekoppelten Activity, in welchem verschiedenen Interaktionsmöglichkeiten zur SQLite- und Oracle-

Datenbank bestehen. In diesem Screen sind im oberen Bereich vier Buttons implementiert. Mit dem Button SQLite bzw. Oracle werden Aktionen in der jeweiligen Datenbank durchgeführt. Dazu wird beim Button des entsprechenden Buttons ein Auswahlmenü geöffnet, das in der Abbildung 4-23 in dem rechten Teil für den Oracle-Button dargestellt ist.

Durch einen Klick auf den ersten Menüpunkt werden die Datenbank-Einträge von Oracle in

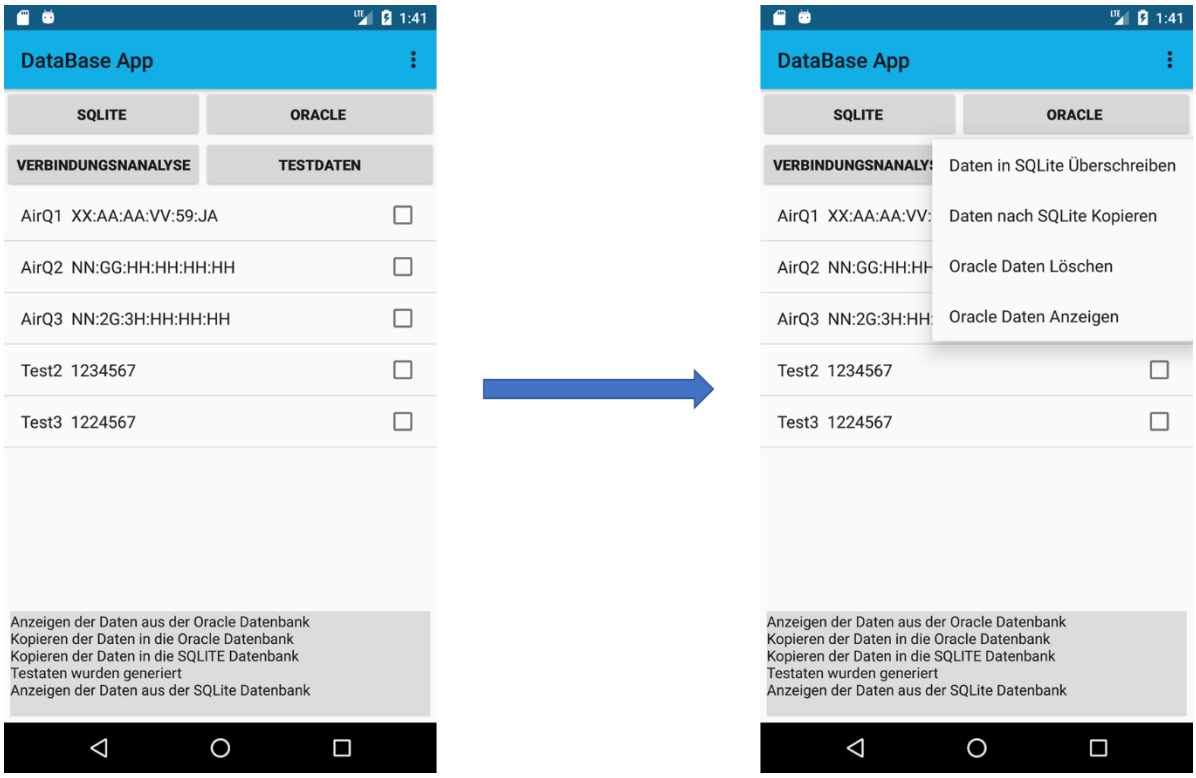

Abbildung 4-23 Screen zur Datenübertragung zwischen den Datenbanken

SQLite überschrieben. Dies bedeutet das die Daten in der SQLite-Datenbank gelöscht werden und die Daten von Oracle in SQLite übernommen werden. Beim Drücken von „Daten nach SQLite kopieren“ werden die noch nicht vorhandenen Daten von Oracle in SQLite kopiert. Mit dem dritten Menüpunkt können alle Daten in der Oracle-Datenbank gelöscht werden. Die Tabelle der Arduinodaten wird durch das Drücken des vierten Menüpunktes in einer Listenform angezeigt, die mittig im Screen angeordnet ist. In der Listenansicht können Arduino-Einträge gelöscht werden. Dazu drückt der Benutzer länger auf das Quadrat hinten den zu löschenden Arduinos. Oben in der Benutzerleiste öffnet sich ein Menü. Durch Klicken des Papierkorbs werden Arduinos gelöscht. Befindet sich der Nutzer in der Listenansicht für SQLite werden nur Arduinos in der internen Datenbank mit den referenzierten Daten gelöscht. Wird ein Arduino in Oracle gelöscht, wird dieser auch in SQLite entfernt. Analog sind die Auswahlmenüfunkti-

onen beim SQLite Button hinterlegt. Mit dem Button Verbindungsanalyse werden Informationen zu der Verbindung zum Oracle-Server im Textfeld am unteren Ende des Screens dargestellt. Zum Erstellen von Testdaten kann der Button Testdaten genutzt werden. Dabei werden Testdaten in SQLite und Oracle erzeugt. Der Nutzer kann die durchgeführten Aktionen im Textfeld am unteren Ende des Screens nachvollziehen.

4.3.2 Klasse für Zugriff auf SQLite

Für den Zugriff auf SQLite ist die Activity DayTable in Abbildung 4-24 implementiert worden und dient lediglich dazu Einträge in die AirQ Tabelle zu schreiben oder zu löschen. Dieser Screen beinhaltet im rechten oberen Bereich wieder das Optionsmenü zum Wechseln zwischen den Screens. Des Weiteren sind im oberen Bereich zwei Eingabefelder zur Eingabe von einem Label und einer MAC-Adresse eines Arduinos vorhanden. Dadurch können neue Arduinos hinzugefügt werden. Das Hinzufügen eines Arduinos erfolgt mit dem „Plus-Button“. In der darunterliegenden Listenansicht werden alle vorhandenen Arduinos mit ihrer MAC-Adresse in SQLite dargestellt. Auch in dieser Listenansicht können Arduinos durch das längere Drücken auf das entsprechende Quadrat gelöscht werden. Dies ist im rechten Teil der Abbildung 4-24 dargestellt. Durch das Drücken des Papierkorbs in der oberen Leiste werden die ausgewählten

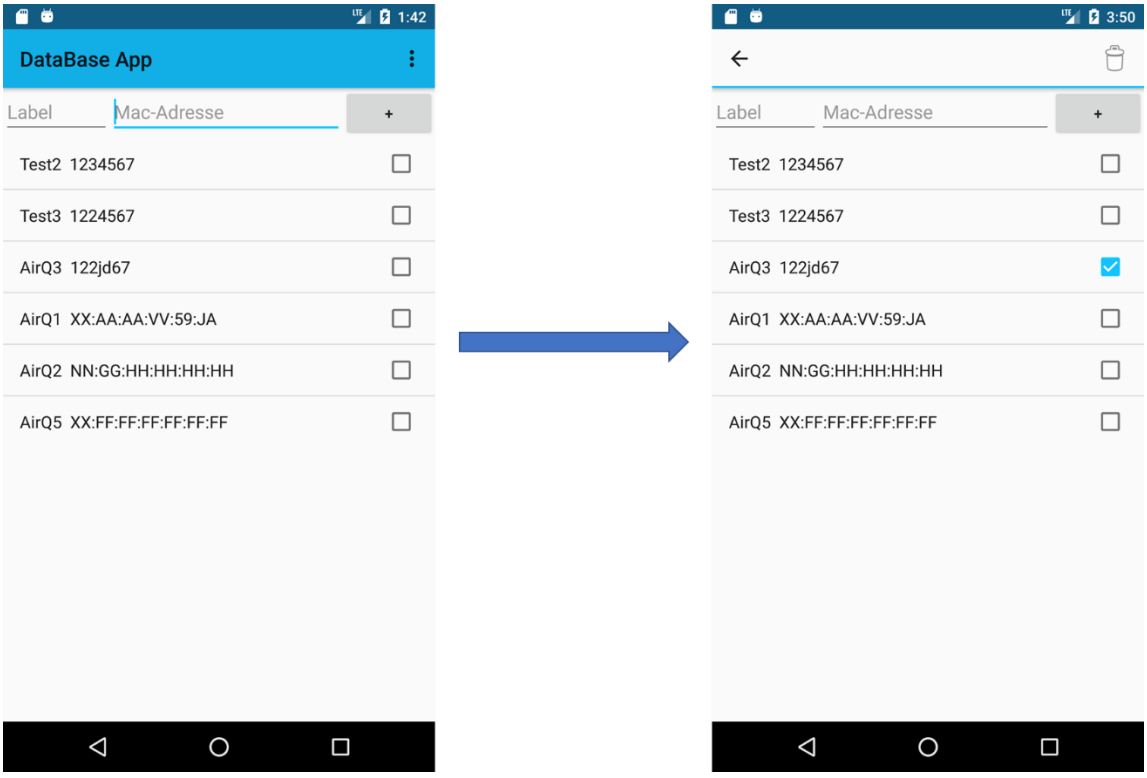

Abbildung 4-24 Screen für die Verwaltung der Arduinodaten in SQLite

Arduinos gelöscht und darunterliegende verknüpfte Tabelleneinträge (Tagesdaten und Sensordaten) mit passendem Fremdschlüssel beim Löschen kaskadiert.

4.4 Testen der App

Zum Überprüfen der Funktionalität dieser App sind besagte Logs genutzt worden, die im Debug-Fenster von Android Studio über den LogCat-Monitor ausgegeben werden können. Dafür musste die App im Debug-Modus laufen. Durch die Logs werden beim Testen der App verschiedene Zustände der App überprüft. Dazu wurden Logs in jeder Klasse an wichtigen Schlüsselstellen des Programmcodes eingebunden (zu sehen in den Code-Ausschnitten in Kapitel 4.2 und 4.3). Durch sogenannte Intent-Filter im LogCat Monitor können gezielt einzelne Klassen deklariert werden, deren Funktionsaufrufe nachvollzogen werden sollen. Sind dort Logs eingebunden, werden diese bei Auswahl des Intent-Filters eingeblendet. Dadurch weiß der Nutzer, ob die Interaktion mit der internen bzw. der externen Datenbank funktioniert hat. In der Abbildung 4-25 ist ein Teil-Ausschnitt aus dem Log-Monitor aus der JdbcHelper-Klasse abgebildet. In diesem Ausschnitt ist zu erkennen, dass Informationen beim Hinzufügen von Tabelleninhalten, beim Suchen von Tabelleninhalten und bei der Abfrage von Tabelleninhalten in den Logs dargestellt werden.

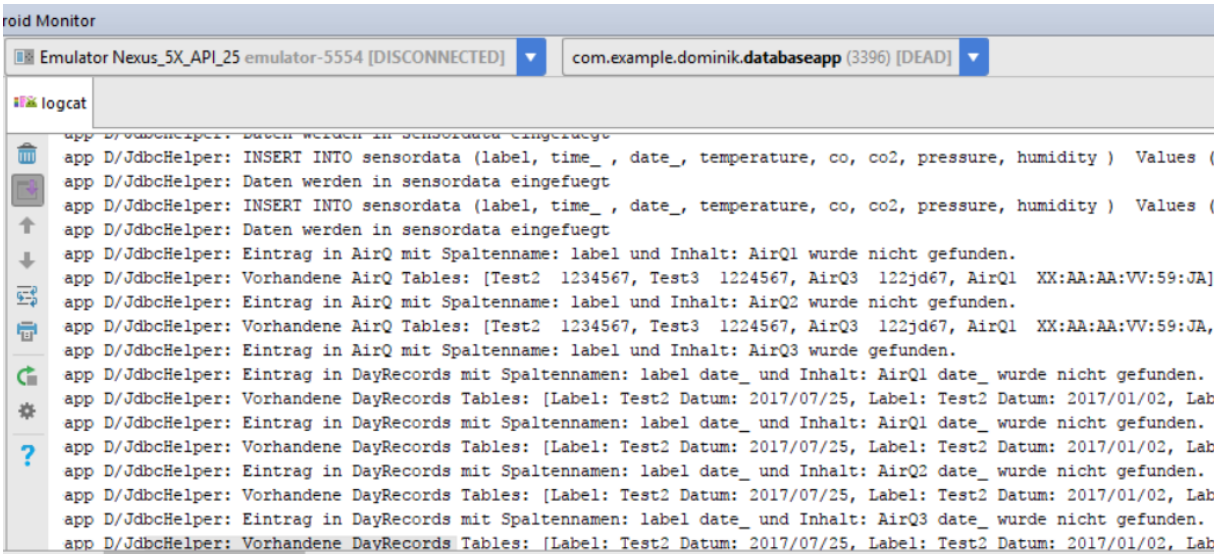

Abbildung 4-25 Ausschnitt aus dem Log-Monitor in der JdbcHelper-Klasse

Nach diesem Verfahren wurden sämtliche Klassen überprüft, wobei es lediglich kleine Programmierfehler zu beanstanden gab. Diese wurden ausgebessert, sodass die Applikation wie geplant lauffähig ist. Während der Testphase ist zusätzlich ein Problem mit Oracle-Statements aufgefallen, dass die Oracle-DB nicht ohne weiteres referenzierte Tabelleninhalte aktualisieren

(kaskadieren) kann. Dieses Problem wird umgangen indem zuerst der Fremdschlüssel deaktiviert wird, dann jede einzelne Tabelle händisch aktualisiert wird und nach Abschluss der Fremdschlüssel wieder aktiviert wird. Dieser Update-Mechanismus wurde daher aus der Implementierung entfernt und kann für einen späteren Zeitpunkt wieder vorgesehen werden. Nach einiger Recherche zeigt sich, dass dafür oft Prozeduren oder Trigger genutzt werden, diese wurden allerdings aufgrund des benötigten zusätzlichen Recherche-Aufwands ebenfalls nicht implementiert.

5 Zusammenfassung

Aus der Zusammenarbeit zwischen der Universität Helsinki und der Hochschule Osnabrück ist innerhalb eines Projektteams ein System zur Überwachung von Luftqualitätswerten entstanden. Dieses System besteht aus einem Arduino mitsamt Sensoren zur Aufnahme von Daten und einer Applikation (App), welche diese Daten für einen Benutzer visualisiert. Dafür wird eine proprietäre Bluetooth-Verbindung genutzt, um eine Kommunikationsverbindung zwischen App und Arduino herzustellen. Übertragbare Daten sind Informationen über den jeweiligen Arduino, aktuelle Sensorwerte oder nach Wahl Langzeitdaten, in Form von Tagesdaten.

Im Rahmen dieser Arbeit wird für die eben genannte App eine prototypische Erweiterung geschrieben. Diese beinhaltet zwei große Teilbereiche. Zum einen wird das bisherige Daten-Speicherungsmodell der App verworfen und mittels der Standardbibliothek SQLite durch ein internes Datenbank-Modell ersetzt. Auf Basis eines EER-Modells bzw. eines relationalen Schemas werden in der Konzeptphase Tabellenstrukturen definiert. Diese werden mit verschiedenen sogenannten Helfer-Klassen implementiert, welche die SQLite-Funktionalität einbinden und Funktionen wie das Eintragen, Löschen, Aktualisieren und Suchen von Einträgen für den Benutzer bereitstellen.

Der zweite große Teilbereich dieser Arbeit beschäftigt sich damit, dass Messdaten eines Arduinos ohne physische Präsenz verfügbar gemacht werden. Grundgedanke ist dabei, dass einmal empfangene Daten in einer externen Datenbank, in diesem Fall konkret am Beispiel des Hochschul-Oracle-Servers, abgelegt werden. Dafür wird ein sogenannter JDBC-Treiber in der Applikation implementiert, welcher eine Schnittstelle zur Verfügung stellt, um mit Datenbanken-Strukturen kommunizieren zu können. Mithilfe dieser Schnittstelle wird ein Interface für den Benutzer geschrieben, welches es diesem ermöglicht, über eine Benutzerkennung Zugriff auf den Oracle-Server zu erhalten. Wie schon in der internen SQLite-Datenbankstruktur werden dafür Funktionen zum Eintragen, Löschen, Aktualisieren und Suchen von Einträgen für den Benutzer bereitgestellt.

Diese beiden Teilbereiche werden in Form von Activities implementiert. Activites sind Java-Klassen, die eine graphische Bedienoberfläche implementieren. So sind in der prototypischen z.B. Buttons vorhanden, welche Funktionen aus den Schnittstellen zu der SQLite-Datenbank oder der Oracle-Datenbank aufrufen.

6 Ausblick

Die prototypische Applikation wurde eine Verbindung zu der internen SQLite-Datenbank und der externen Oracle-Datenbank implementiert. Im nächsten Schritt muss die Einbindung in die Applikation des International-Sensor-Development-Projekts erfolgen. Sinnvoll ist hier das Einbinden in den bereits vorhandenen JSON-Parser, wo erhaltene Daten über die Bluetooth-Verbindung entschlüsselt werden. Im Anschluss ist ein Trigger oder eine Prozedur in der Oracle-Datenbank einzufügen, um Akutalisierungs-Statements kaskadieren zu können, da bisher Fremdschlüsselbeziehungen diese Update-Methodik unbrauchbar machen.

7 Literaturverzeichnis

[1] HS Osnabrück/Metropolia, ISDP-Dokumentation, Osnabrück/Helsinki, 2017.

[2] „Einführung in JSON," [Online]. Available: http://www.json.org/json-de.html. [Zugriff am 27. 07. 2017].

[3] D. Abts, Masterkurs Client/Server-Programmierung mit Java, Wiesbaden: Vieweg +TeubnerVerlag I Springer Fachmedien Wiesbaden GmbH, 2010.

Anhang A Inhalt der CD

Auf der beigefügten CD sind folgende Ordner und Dateien enthalten.

Tabelle A-1: Inhalt der CD

Ordnerverzeichnis	Dateien	Beschreibung
\Hausarbeit	Hausarbeit_Datenbanken.pdf	Die Hausarbeit im Portable Document Format (PDF)
	Hausarbeit_Datenbanken.docx	Die Hausarbeit im Microsoft Word Format
\Android_Studio_Projektordner	DatabaseApp	Das erstellte Android-Projekt in Ordnerform
	DatabaseApp.apk	Die erstellte App zum Installieren auf dem Smartphone im Android-Package-Format
\Literatur	JSON.html	Verwendete Internetseite im html-Format
	Masterkurs_Client_Server_Programmierung_mit_JAVA.pdf	Verwendete elektronische Quelle
	ISDP-Dokumentation.pdf	Dokumentation zum ISDP-Projekt

Anhang B Abbildung zum SQ-Lite Helfer

In der Abbildung B-7-1 wird der Programmcode zur Erstellung der Tabellen dargestellt. Dieser Programmcode entspricht der Datenstruktur der Tabellen in beiden Datenbanken. Lediglich die Datentypen und das Update müssen in der Oracle-DB angepasst werden.

```
private static final String LOG_TAG = IsdpDbHelper.class.getSimpleName();

public static final String DB_NAME = "AirQualityDB.db";
public static final int DB_VERSION = 1;
public static final String TABLE_AIRQ_LIST = "airQ_List";
public static final String COLUMN_ID = "_id";
public static final String COLUMN_MAC = "mac";
public static final String COLUMN_LABEL = "label";
// Table for available Day List
public static final String TABLE_DAY_LIST = "day_list";
// Table for the sensor data
public static final String TABLE_SENSORDATA = "sensordata";
public static final String COLUMN_TIME = "time_";
public static final String COLUMN_DATE = "date_";
public static final String COLUMN_TEMPERATURE = "temperature";
public static final String COLUMN_CO = "co";
public static final String COLUMN_CO2 = "co2";
public static final String COLUMN_PRESSURE = "pressure";
public static final String COLUMN_HUMIDITY = "humidity";

public static final String SQL_CREATE =
        "CREATE TABLE " + TABLE_AIRQ_LIST +
                "(" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COLUMN_MAC + " TEXT NOT NULL UNIQUE, " +
                COLUMN_LABEL + " TEXT NOT NULL UNIQUE);";

public static final String SQL_CREATE1 =
        "CREATE TABLE " + TABLE_DAY_LIST +
                "(" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COLUMN_DATE + " TEXT NOT NULL, " +
                COLUMN_LABEL + " TEXT NOT NULL, " +
                "UNIQUE ("+COLUMN_LABEL+","+COLUMN_DATE+"), "+
                "FOREIGN KEY ("+COLUMN_LABEL+") REFERENCES "
                +TABLE_AIRQ_LIST+"("+COLUMN_LABEL+") ON UPDATE CASCADE ON DELETE CASCADE);";

public static final String SQL_CREATE2 =
        "CREATE TABLE " + TABLE_SENSORDATA +
                "(" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COLUMN_TIME + " TEXT NOT NULL, " +
                COLUMN_DATE + " TEXT NOT NULL, " +
                COLUMN_TEMPERATURE + " INTEGER NOT NULL, " +
                COLUMN_CO + " INTEGER NOT NULL, " +
                COLUMN_CO2 + " INTEGER NOT NULL, " +
                COLUMN_PRESSURE + " INTEGER NOT NULL, " +
                COLUMN_HUMIDITY + " INTEGER NOT NULL, " +
                COLUMN_LABEL + " TEXT NOT NULL, " +
                "UNIQUE ("+COLUMN_TIME+","+COLUMN_DATE+","+COLUMN_LABEL+"), "+
                "FOREIGN KEY ("+COLUMN_LABEL+","+COLUMN_DATE+") REFERENCES "
                +TABLE_DAY_LIST+"("+COLUMN_LABEL+","+COLUMN_DATE+") ON UPDATE CASCADE ON DELETE CASCADE);";
```

Abbildung B-7-1 Programmcode zur Erstellung der Tabellen

Erklärung

Hiermit versichern wir, dass wir die Hausarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Datum: …….....

(Unterschrift)

Datum: …….....

(Unterschrift)