Technische Universität Braunschweig

Institut für Nachrichtentechnik

X-Map

Android-App zur qualitativen Erfassung von Mobilfunknetzen

Software-Entwicklungspraktikum (SEP)
Sommersemester 2013

Systemspezifikation

Auftraggeber
Technische Universität Braunschweig
Institut für Nachrichtentechnik
Prof. Dr.-Ing. Thomas Kürner
Schleinitzstraße 22
38106 Braunschweig

Betreuer: Dennis M. Rose

Auftragnehmer:

Name	E-Mail-Adresse
Sofia Ananieva	s.ananieva@tu-braunschweig.de
Andreas Bauerfeld	a.bauerfeld@tu-braunschweig.de
Ferhat Çinar	f.cinar@tu-braunschweig.de
Andreas Hecker	a.hecker@tu-braunschweig.de
Julia Kreyßig	julia@kreyssig.com
Timo Schwarz	t.schwarz@tu-braunschweig.de
Julian Troegel	j.troegel@tu-braunschweig.de
Deniz Yurtseven	d.yurtseven@tu-braunschweig.de

Braunschweig, 26. Juni 2013

Versionsübersicht

Version	Datum	Autor	Status	Kommentar
0.0.1	01.05.2013	Timo Schwarz	abgenommen	Kapitel 2, 5 angepasst
0.0.3	09.05.2013	Julian Troegel	abgenommen	Kapiteln 2 erweitert
0.0.4	10.05.2013	Timo Schwarz	abgenommen	Kapiteln 1.1, 2, 3 erweitert
0.0.7	11.05.2013	Julian Troegel	abgenommen	Kapiteln 2, 3 überarbeitet
0.0.9	11.05.2013	Ferhat Cinar, Deniz Yurtseven	abgenommen	Kapitel 2 erweitert
0.1.0	11.05.2013	Julian Troegel	abgenommen	Kapitel 5 & Glossar erweitert
0.1.1	11.05.2013	Timo Schwarz	abgenommen	Kapitel 1 erweitert
0.1.2	11.05.2013	Sofia Ananieva	abgenommen	Kapitel 3 erweitert
0.1.3	11.05.2013	Ferhat Cinar	abgenommen	Kapitel 1.1 erweitert
0.1.4	11.05.2013	Andreas Bauerfeld	abgenommen	Kapitel 2 erweitert
0.1.5	12.05.2013	Andreas Hecker, Sofia Ananieva	abgenommen	Kapitel 2 erweitert
0.1.6	12.05.2013	Andreas Bauerfeld	abgenommen	Kapitel 2 erweitert
0.1.7	12.05.2013	Andreas Hecker	abgenommen	Kapitel 2 erweitert
0.1.8	12.05.2013	Julian Troegel	abgenommen	Kapitel 2 und 3 erweitert
0.1.9	12.05.2013	Sofia Ananieva	abgenommen	Kapitel 2 und 3 erweitert
0.1.10	12.05.2013	Julia Kreyßig	abgenommen	Kapitel 2 erweitert
0.1.11	12.05.2013	Ferhat Cinar	abgenommen	Kapitel 1.1 überarbeitet
0.1.12	12.05.2013	Deniz Yurtseven	abgenommen	Kapitel 1.1 erweitert
0.1.13	12.05.2013	Julia Kreyßig, Timo Schwarz	abgenommen	Kapitel 3.1 erweitert
0.1.14	12.05.2013	Sofia Ananieva	abgenommen	Kapitel 3.1.2 erweitert
0.1.15	12.05.2013	Andreas Hecker	abgenommen	Kapitel 5 erweitert
0.1.16	13.05.2013	Julian Troegel	abgenommen	Kapitel 1, 3, 5 erweitert
0.2.0	14.05.2013	Julia Kreyßig, Timo Schwarz	abgenommen	Kapitel 1 erweitert
0.2.1	14.05.2013	Julian Troegel	abgenommen	Kapitel 2 erweitert. NFA eingefügt
0.2.2	14.05.2013	Andreas Bauerfeld	abgenommen	Kapitel 3 erweitert. Backend
0.2.3	14.05.2013	Sofia Ananieva	abgenommen	Kapitel 3.3.2 erweitert
0.2.4	14.05.2013	Andreas Bauerfeld	abgenommen	Schnittstelle eingefügt
0.2.5	14.05.2013	Andreas Hecker	abgenommen	Schnittstelle eingefügt

0.2.6	14.05.2013	Andreas Bauerfeld, Andreas Hecker, Julian Troegel, Sofia Ananieva	abgenommen	Kapitel 5 erweitert
0.2.7	15.05.2013	JJulia Kreyßig, Timo Schwarz	abgenommen	Kapitel 3.2 erweitert
0.2.8	15.05.2013	Julian Troegel	abgenommen	Kapitel 4 erstellt, Dokument formatiert
1.0.0	15.05.2013	Julian Troegel, Timo Schwarz	abgenommen	Abgabeversion Systementwurf 1
1.1.0	15.06.2013	Timo Schwarz	abgenommen	Kapitel 6.2, 7 erweitert
1.1.1	16.06.2013	Sofia Ananieva	abgenommen	Kapitel 5 erweitert
1.1.2	16.06.2013	Timo Schwarz	abgenommen	Kapitel 6.2 erweitert
1.1.4	18.06.2013	Julian Troegel	abgenommen	Kapitel 5.4 und 6.1 erweitert
1.1.5	19.06.2013	Andreas Hecker	abgenommen	Kapitel 8 überarbeitet
1.1.6	19.06.2013	Julian Troegel	abgenommen	Kapitel 2, 3, 5, 6 überarbeitet
1.1.7	20.06.2013	Timo Schwarz	abgenommen	Kapitel 3, 5, 6 erweitert
1.1.8	21.06.2013	Sofia Ananieva, Andreas Hecker	abgenommen	Kapitel 5 erweitert
1.1.9	21.06.2013	Timo Schwarz	abgenommen	Kapitel 5 erweitert
1.2.0	22.06.2013	Julian Troegel	abgenommen	Kapitel 3 und 5 erweitert
1.2.1	23.06.2013	Julia Kreyßig	abgenommen	Kapitel 5 erweitert
1.2.2	23.06.2013	Andreas Hecker	abgenommen	Kapitel 5 erweitert
1.2.3	23.06.2013	Julia Kreyßig	abgenommen	Kapitel 5 erweitert
1.2.4	23.06.2013	Sofia Ananieva	abgenommen	Kapitel 5 erweitert
1.2.5	23.06.2013	Andreas Hecker	abgenommen	Kapitel 5 erweitert
1.2.6	24.06.2013	Timo Schwarz	abgenommen	Kapitel 5 erweitert
1.2.7	24.06.2013	Sofia Ananieva	abgenommen	Kapitel 5 erweitert
1.2.8	25.06.2013	Andreas Hecker	abgenommen	Kapitel 5 erweitert
1.2.9	25.06.2013	Sofia Ananieva	abgenommen	Kapitel 2, 3 Diagramme korrigiert
1.3.0	25.06.2013	Timo Schwarz	abgenommen	Kapitel 7 erweitert
1.3.1	26.06.2013	Andreas Hecker, Sofia Ananieva	abgenommen	Kapitel 5 erweitert
1.3.2	26.06.2013	Ferhat Cinar, Deniz Yurtseven	abgenommen	Kapitel 1.1 überarbeitet

1.3.3	26.06.2013	Ferhat Cinar	abgenommen	Kapitel 5.10 Anmerkung hinzugefügt
1.3.4	26.06.2013	Timo Schwarz	abgenommen	Kapitel 1 erweitert
1.3.5	26.06.2013	Deniz Yurtseven	abgenommen	Kapitel 5 erweitert
1.3.6	26.06.2013	Andreas Bauerfeld	abgenommen	Kapitel 5 erweitert
1.4.0	26.06.2013	Julian Troegel	abgenommen	Kapitel 2, 4, 5 erweitert & Rechtschreibfehler behoben
2.0.0	26.06.2013	Timo Schwarz, Julian Troegel	abgenommen	Abgabeversion Systementwurf 2

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

X-Map ist ein verteiltes System zur Aufzeichnung, Speicherung und Auswertung von Messdaten zum Mobilfunknetz. Es setzt sich prinzipiell zusammen aus

- ... einer Android-Applikation, welche auf Mobilgeräten, auf denen sie installiert ist, Messdaten sammelt.
- ... einem WebService, der für Benutzerverwaltung und Messdatenspeicherung verantwortlich ist.
- ... einer Web-Anwendung, welche Möglichkeiten zur Visualisierung und Auswertung der Messdaten bereitstellt.

Kapitel 2 dieses Dokuments befasst sich mit der Erläuterung der im Pflichtenheft festgelegten Produktfunktionen und deren geplanter Umsetzung.
Kapitel 3 beschreibt, zu welcher Architektur die Analyse der Funktionen in Kapitel 2 geführt hat.
Kapitel 4 stellt mithilfe eines UML-Deployment-Diagramms dar, in welcher Weise die oben beschriebenen Strukturen verteilt sind.
Kapitel 5 geht in detaillierter Weise auf die Implementierung der einzelnen Komponenten, welche in Kapitel 3 gefunden wurden, ein.
Kapitel 6 beschreibt die Datenbanken, die sowohl App als auch WebService zugrunde liegen.
Kapitel 7 stellt in kurzer Weise da, wie ein zu verwendender Server zu konfigurieren ist.
In Kapitel 8 wird darüber Buch geführt, welche Kriterien bereits auf welche Weise umgesetzt wurden und welche noch zu bearbeiten sind.
Der Detailgrad der einzelnen Ausführungen in den jeweiligen Unterkapiteln ist dabei durch den aktuellen Stand der Entwicklung beschränkt.

1.1 Projektdetails

Die in Kapitel 1 angeschnittenen Grundkomponenten werden in der Folge durch Aktivitätsdiagramme bzw. Statecharts detailliert. Als Grundlage dienen die geplanten Normalfälle der Benutzung der einzelnen Komponenten.

1.1.1 Android Applikation

Es wird ein typischer Ablauf der Android Applikation beschrieben.

Abbildung 1.1 stellt einen typischen Arbeitsablauf für einen Benutzer dar, welcher Messungen auf der Android Applikation durchführen lässt.
Zunächst muss der Benutzer die App auf seinem Mobilgerät starten. Dabei wird angenommen, dass der Benutzer die App vorher schon installiert hat. Daraufhin kann der Benutzer verschiedene Benutzereingaben durchführen. Typische Benutzereingaben sind dabei der Start der Visualisierungen, die Einstellungen bearbeiten, sich anmelden oder registrieren, Übertragungen an den WebService durchführen lassen und weitere. Diese Benutzereingaben werden hier nicht weiter spezifiziert. Der Benutzer kann aber auch, ohne Messungen durchzuführen, die App beenden. Entweder ist der Ablauf dann beendet oder der Benutzer startet die App neu.
Sobald der Benutzer die Messungen starten lässt, wird innerhalb des Mobilgerätes ein Alarm gesetzt, der die periodischen Zeitpunkte für einen Receiver setzt. Der Alarm bindet sich dabei in das Android-System und kann damit auch ausgeführt werden, wenn die App nicht aktiv ist oder das Mobilgerät im Standby ist.
Der gestartete Receiver besitzt dabei eine sehr kurze Lebensdauer, da dieser vom Android-SDK bereitgestellt wird und somit vom ressourcensparenden Betriebssystem nach sehr kurzer Zeit beendet wird. Deshalb startet dieser einen Service, der länger überleben kann und die Messungen durchführt und anschließend die Daten in der Datenbank speichert. Danach beendet der Service sich selbst. Zum nächsten vom Alarm gesetzten Zeitpunkt wird der Receiver wieder gestartet und der Ablauf wiederholt sich.
Parallel zu diesem sich immer wiederholen Ablauf, kann der Benutzer verschiedene weitere Eingaben machen. Zum einen sind die oben beschriebenen Benutzereingaben möglich und zum anderen kann der Benutzer auch die App beenden. Dadurch, dass der Alarm sich in Android-System gebunden hat, läuft der Vorgang der Messung weiter ab. Der Benutzer muss dann die App wieder starten, um weitere Aktionen durchführen zu können. Andererseits kann der Benutzer die Messungen auch stoppen. Dadurch wird der Alarm beendet und aus dem Android-System geschrieben.
Nun können weitere Aktionen durchgeführt werden, wenn der Nutzer dieses möchte.

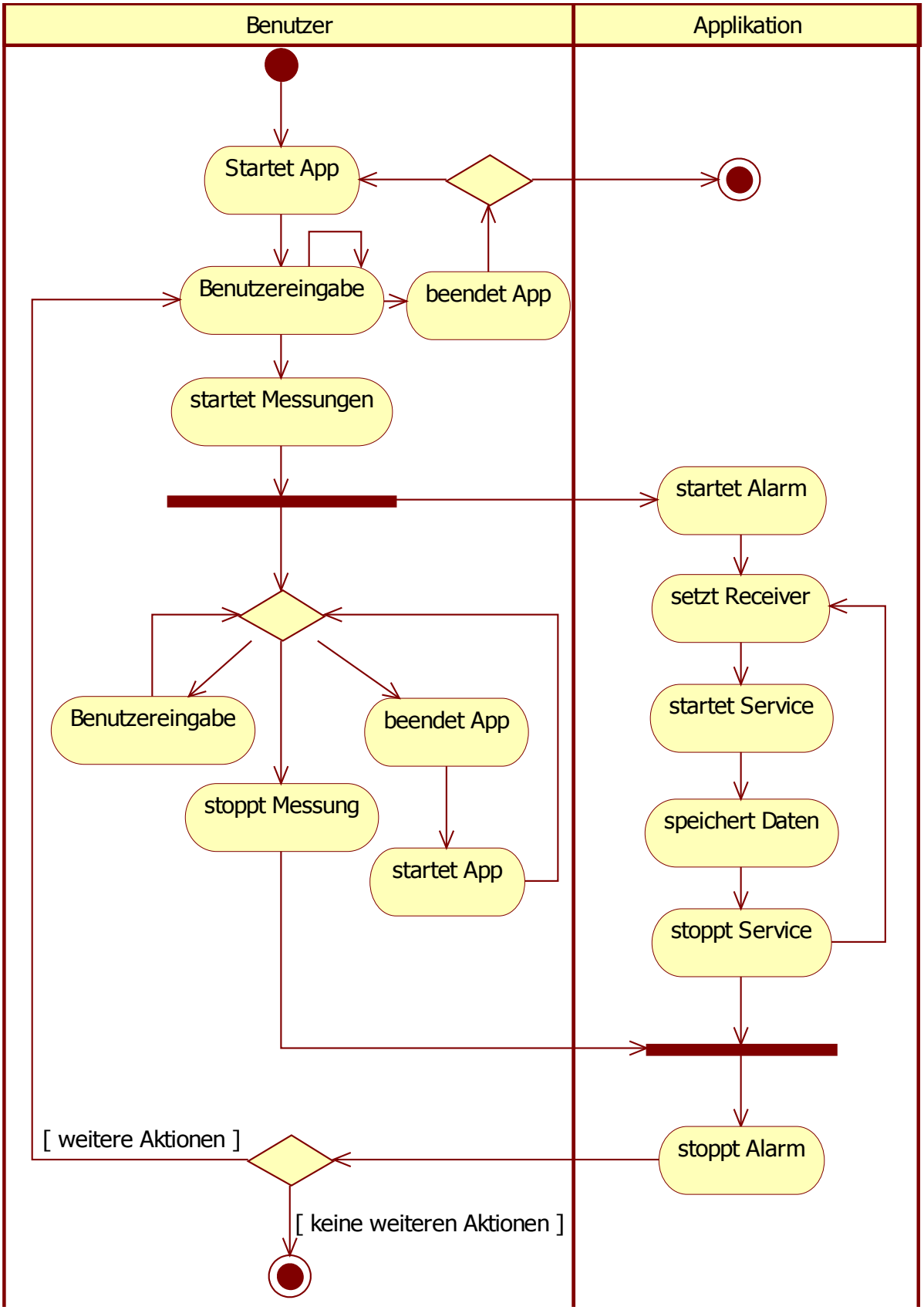

Abbildung 1.1: Aktivitätsdiagramm zur Benutzung der Android Applikation

1.1.2 WebService

Es wird ein typischer Ablauf des WebServices beschrieben.

Abbildung 1.2 beschreibt den typischen Ablauf, wenn ein Nutzer die App installiert hat und nun das Übertragen von Daten an den WebService ermöglichen möchte.
Hierfür startet der Nutzer zunächst die Registrierungsprozedur in der App und gibt seine Daten ein. Diese überträgt die App dann zusammen mit Informationen über das verwendete Gerät an den WebService, welcher den Datensatz in die Datenbank einträgt. (Sind die Nutzerdaten bereits vorhanden, erhält die App hierüber Rückmeldung.) Ist die Registrierung erfolgt, meldet sich die App automatisch mit den neuen Nutzerdaten an, woraufhin sie vom WebService eine SessionID erhält.
Die App verpackt nun die Messdaten in das vorgegebene Format und übermittelt sie an den WebService, welcher sie an die Datenbank überträgt. Nun kann sie die App vom WebService abmelden, woraufhin die SessionID gelöscht wird.

1.1.3 Web Applikation

Es wird ein typischer Ablauf der Web Applikation beschrieben.

Abbildung 1.3 stellt einen typischen Arbeitsablauf für einen Forscher dar, welcher sich die von X-Map gesammelten Messdaten im Browser ansehen möchte.
Zunächst muss die Webseite der X-Map WebApp aufgerufen werden. Daraufhin muss sich der Benutzer anmelden, wobei hier vorausgesetzt wird, dass er bereits registriert ist. Ist er das nicht, ist die hier nicht abgebildete Registrierfunktion zu nutzen. Nach der Verifizierung der eingegebenen Daten wird dem Nutzer zunächst ein Willkommensbildschirm angezeigt, von welchem aus er die gewünschte Darstellungsart wählen kann. Diese wird daraufhin (zunächst ohne das Einblenden von eigentlichen Messdaten) aufgerufen. In der Folge kann der Nutzer entsprechende Filter auswählen, sodass er nur die Daten angezeigt bekommt, die er haben möchte. Schließlich werden die gefilterten Daten vom WebService an die Applikation zurückgeschickt, welche sie nun darstellen kann.

Abbildung 1.4 stellt die möglichen Zustände dar, welche die Web Applikation besitzen kann. Auf der„Startseite“ befindet man sich im Startzustand. Von dort aus kann man über die Menüleiste zur „Anmelden“-Seite, zur „Info“Seite oder zur Seite „Kontakt“ wechseln, auf der sich nähere Informationen zu den Entwicklern befinden. Falls der Anwender noch kein Benutzerkonto besitzt, gelangt man von der „Anmelden“-Seite zur Registrierung. Wurde über diese erfolgreich ein Benutzerkonto erstellt, wird man vom Zustand „Registrieren“-zur „Anmelde“-Seite zurückgeleitet. Falls sich ein Fehlversuch beim Login oder der Registrierung ereignet, bleibt der Zustand in den jeweiligen Seiten erhalten. Kommt es zu einem erfolgreichen Login, befindet sich die WebApp im Oberzustand „eingeloggt“ und dem Unterzustand „Startseite“. Auch im eingeloggten Zustand

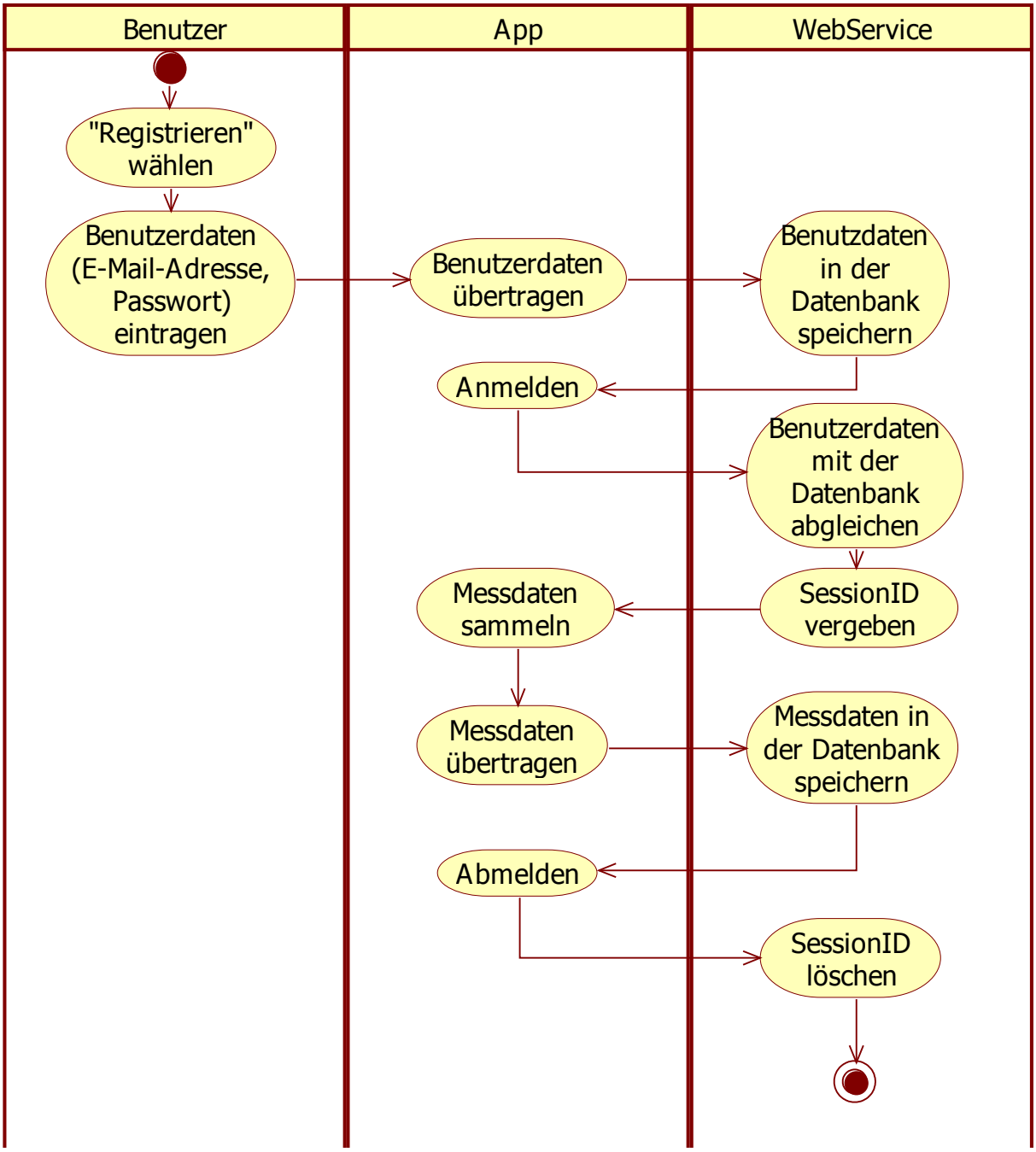

Abbildung 1.2: Aktivitätsdiagramm zur Interaktion zwischen App und WebService

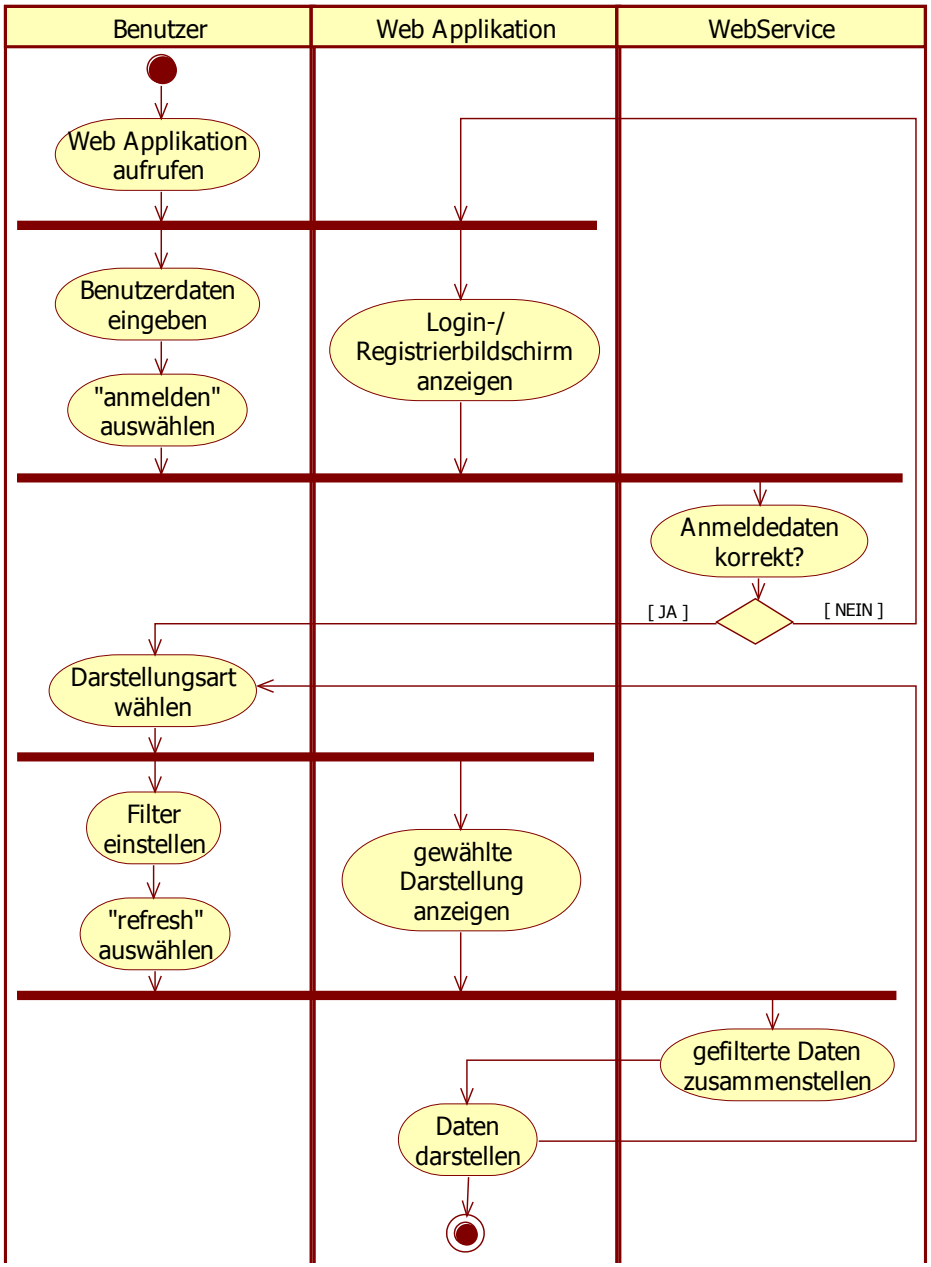

Abbildung 1.3: Aktivitätsdiagramm zur Benutzung der Web Applikation

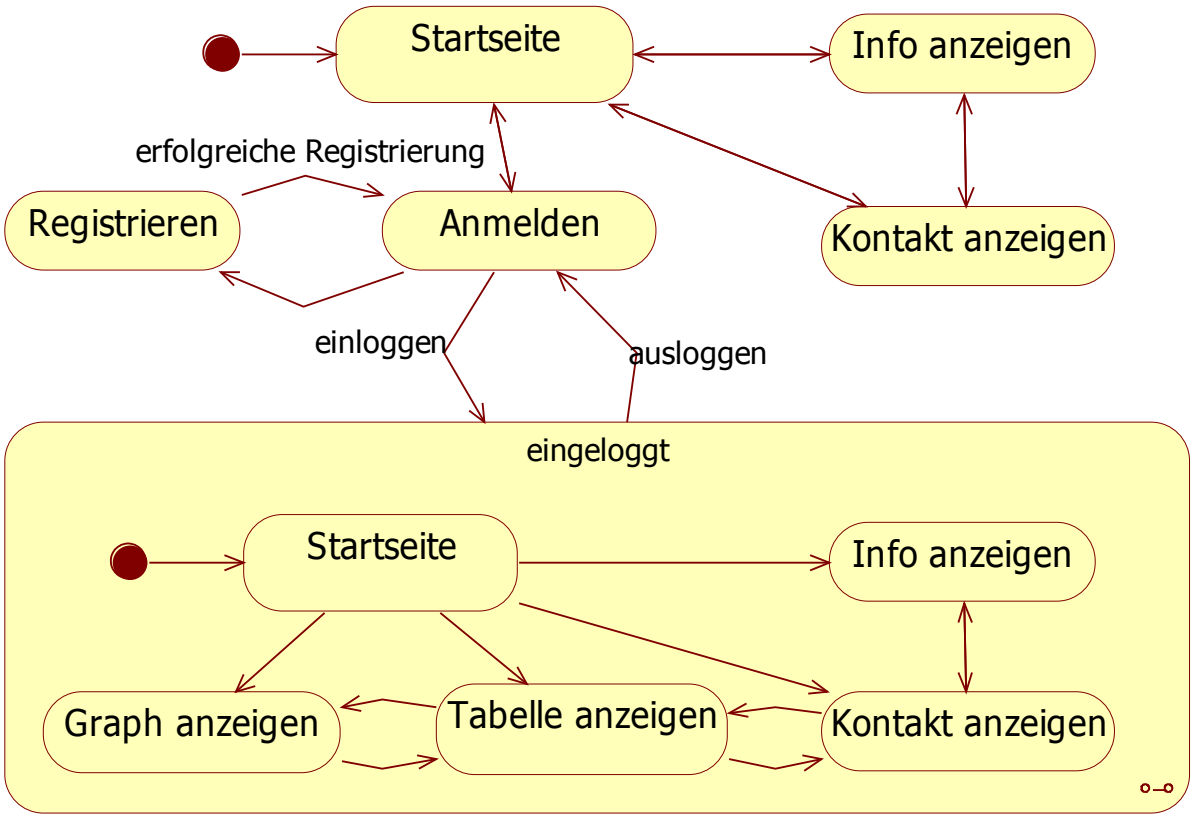

Abbildung 1.4: Statechart zur Web Applikation

kann der Benutzer zu den Seiten „Kontakt“und „Info“ wechseln, jedoch ohne den Oberzustand zu verlassen. Hinzukommt aber die Möglichkeit die Seiten „Graph“ und „Tabelle “ aufzurufen, auf denen die gesammelten Messwerte mit Hilfe des Webservices in Form einer Karte und einer Tabelle visualisiert werden. Falls sich der Benutzer ausloggt, befindet man sich wieder auf der Startseite. Die Aktion „ausloggen“ kann jederzeit ausgeführt werden.

2 Analyse der Produktfunktionen

In den folgenden Unterkapiteln werden die im Pflichtenheft gefundenen Produktfunktionen analysiert und die Art und Weise ihrer geplanten Lösung mithilfe von Sequenz-Diagrammen konkretisiert.

2.1 App

Die folgenden Funktionalitäten $\langle F10 \rangle$ bis $\langle F80 \rangle$ beziehen sich hauptsächlich auf die Android-Applikation.

2.1.1 Analyse von Funktionalität $\langle F10 \rangle$: Daten messen

Auf dem Mobilgerät werden die Daten des Mobilfunknetzes gemessen und in Relation zur Position des Gerätes gesetzt.

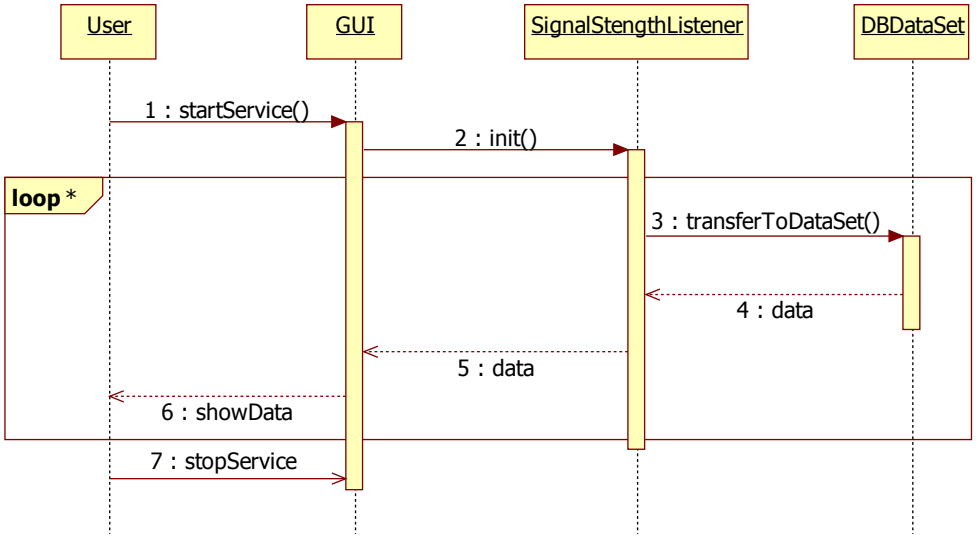

Abbildung 2.1: Sequenzdiagramm zum Messen von Daten

Der User startet die App (1). Nach dem Start wird der SignalStrengthListener initialisiert (2). Der SignalStrengthListener beinhaltet alle Methoden zum Messen der gewünschten Daten. Bei

einer Änderung der Signalstärke wird automatisch eine neue Messung der Daten vorgenommen. Die gemessenen Daten werden in einen temporären Datensatz DBDataSet geschrieben (3). Anschließend wird der Datensatz an das User Interface weitergeleitet (4)+(5) und die Daten ausgegeben (6).

2.1.2 Analyse von Funktionalität $\langle F20 \rangle$: Visualisierung der Android-Applikation

Die auf dem Mobilgerät gespeicherten Messdaten werden durch einen Graphen, eine Tabelle und GoogleMaps visualisiert.

Abbildung 2.2 zeigt die zur Visualisierung nötigen Schritte. Um seine Messdaten zu sehen, kann der Nutzer auf den Visualisierungsbutton klicken. Anschließend kann er zwischen bestimmten Visualisierungstypen wählen: Einem Graph, einer Tabelle oder GoogleMaps. Durch einen Lesezugriff der GUI auf die unterliegende Datenbankschicht, wird das gewünschte Visualisierungsobjekt mitsamt den Messdaten von der GUI erstellt und dem Nutzer angezeigt. Sobald das Visualisierungsobjekt nicht mehr benötigt wird, wird es vom Android-System zerstört. Alternativ wird der Nutzer über eine Fehlermeldung benachrichtigt, falls aus Fehlergründen keine Visualisierungen angezeigt werden kann.

In Abbildung 2.3 lauscht die Singleton Backend Klasse SignalStrenghListener, ob sich die Signalstärke ändert. Ist dies der Fall, werden die neuen Daten gemessen. XYChart, DBTable und GoogleMaps sind Listener der Klasse SignalStrengthListener. Sie implementieren ein Interface, dessen Methode aufgerufen wird, um den neuen Datensatz entsprechend zu visualisieren.

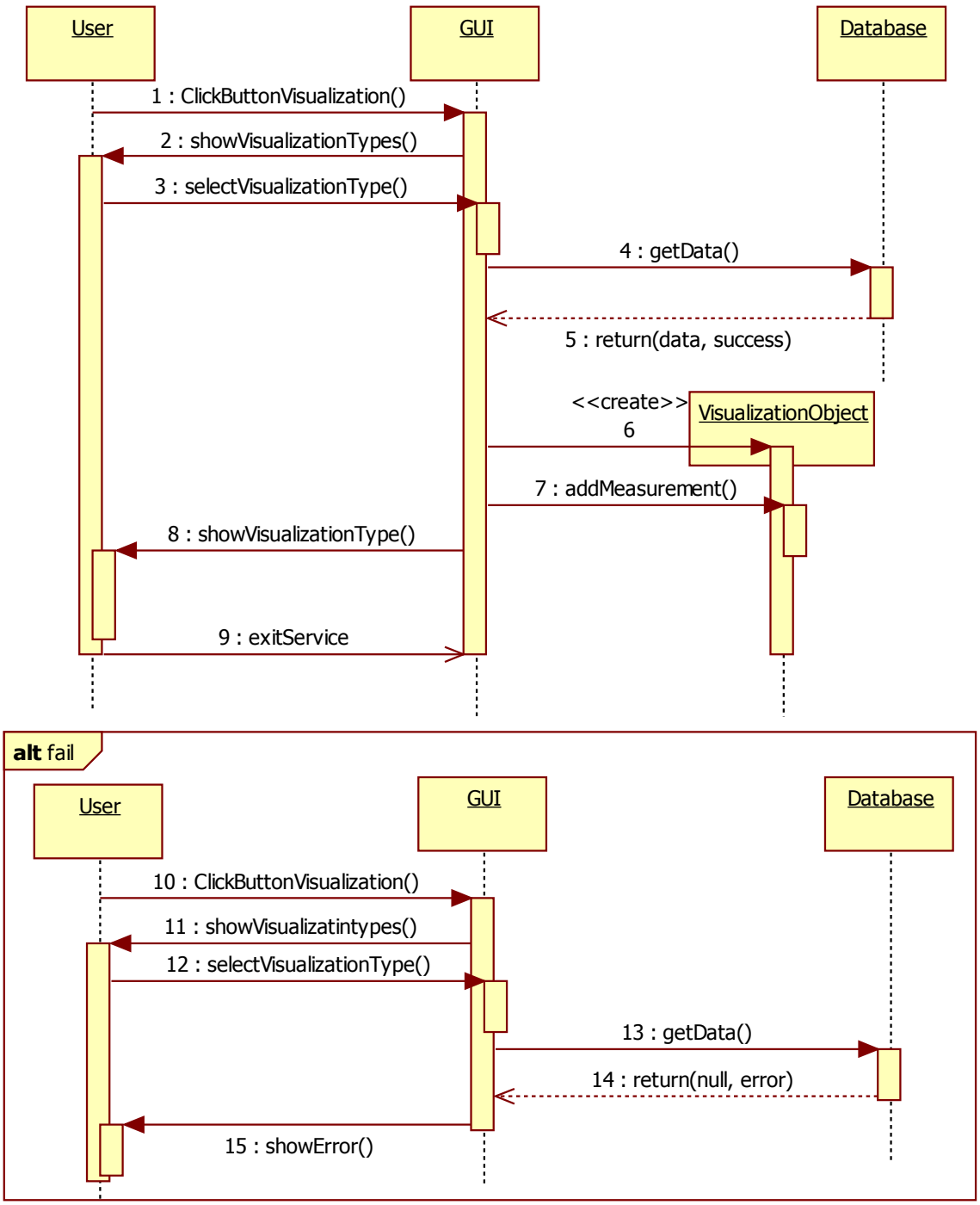

Abbildung 2.2: Sequenzdiagramm zur Visualisierung lokal gespeicherter Daten

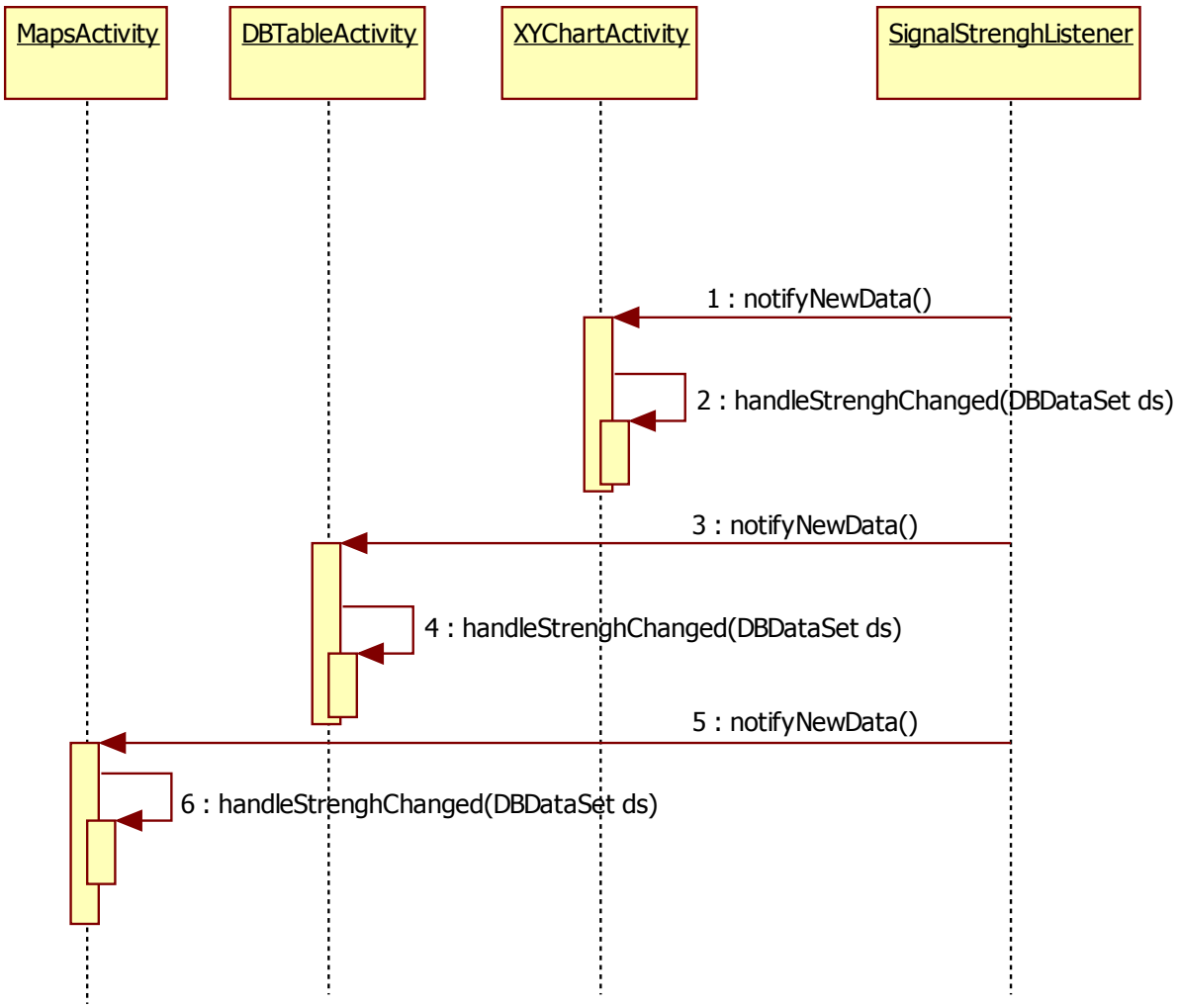

Abbildung 2.3: Sequenzdiagramm zur Kommunikation zwischen neuen Daten und GUI Activities

2.1.3 Analyse von Funktionalität $\langle F30 \rangle$: Daten für WebService

Das Mobilgerät macht sich am Server bekannt und kündigt so das Senden von Daten an. Daraufhin werden zur Verfügung stehende Datensätze an den Server übermittelt. Abbildung 2.4 veranschaulicht dies.

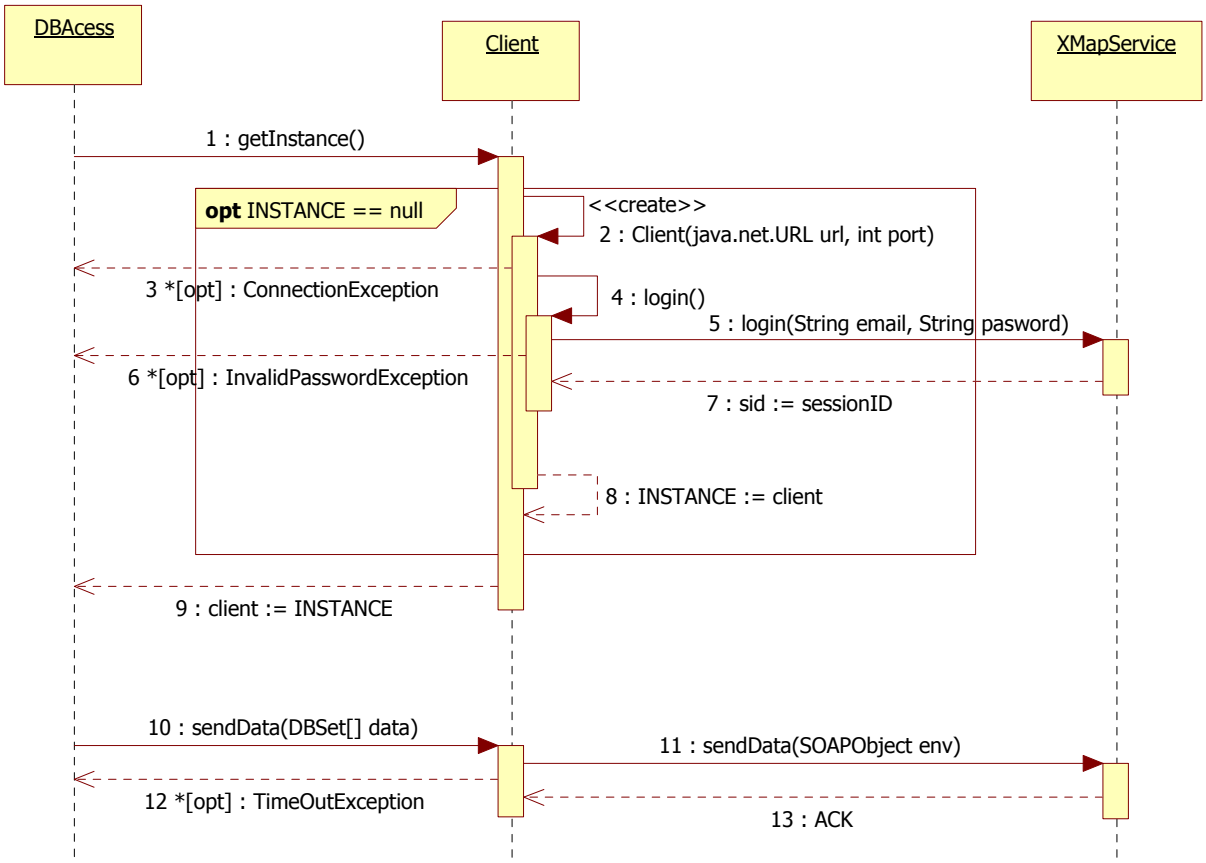

Abbildung 2.4: Sequenzdiagramm zum Anmelden und Übertragen von Daten

Client ist eine Klasse des Design Patterns 'Singleton', welche den Datenaustausch über SOAP implementiert. DBAcess braucht eine Instanz von Client zur Übertragung (1). Ist momentan kein Client-Objekt vorhanden wird dieses erzeugt und mit dem Server verbunden (2). Sollte es Verbindungsfehler geben wird eine ConnectionException zurückgegeben (3). Danach wird das Login ausgeführt (4)+(5). Sollte das Passwort falsch sein und der Server keine Antwort geben wird eine InvalidPasswordException zurückgegeben (6). Läuft alles gut bekommt der Client eine SessionID von X-MapService zurück, welche er für die künftige Verwendung speichert (7). Die Client-Klasse gibt die Referenz auf das Objekt an DBAcess (9). Mit der Referenz kann DBAcess nun sendData aufrufen welche als Parameter ein Array aus Datensätzen enthält (10). Der Client bereitet diese Daten auf und sendet sie an den X-MapService (11), dieser bestätigt den Empfang (13). Kommt es zu Fehlern wird eine TimeOutException erzeugt (12).

2.1.4 Analyse von Funktionalität $\langle F40 \rangle$: Daten löschen

Alte bzw. nicht mehr benötigte Datensätze werden vom Mobilgerät unwiederbringlich gelöscht. Dazu gibt es drei Möglichkeiten, wie es zu einer Datenlöschung kommt.

- Beim Einfügen von Daten. (Abbildung 2.5)
- Bei einer Einstellungsänderung der maximalen Speichergröße. (Abbildung 2.6)
- Der Benutzer kann auch alle Daten in einem von ihm gewählten Zeitraum löschen (Abbildung 2.7).

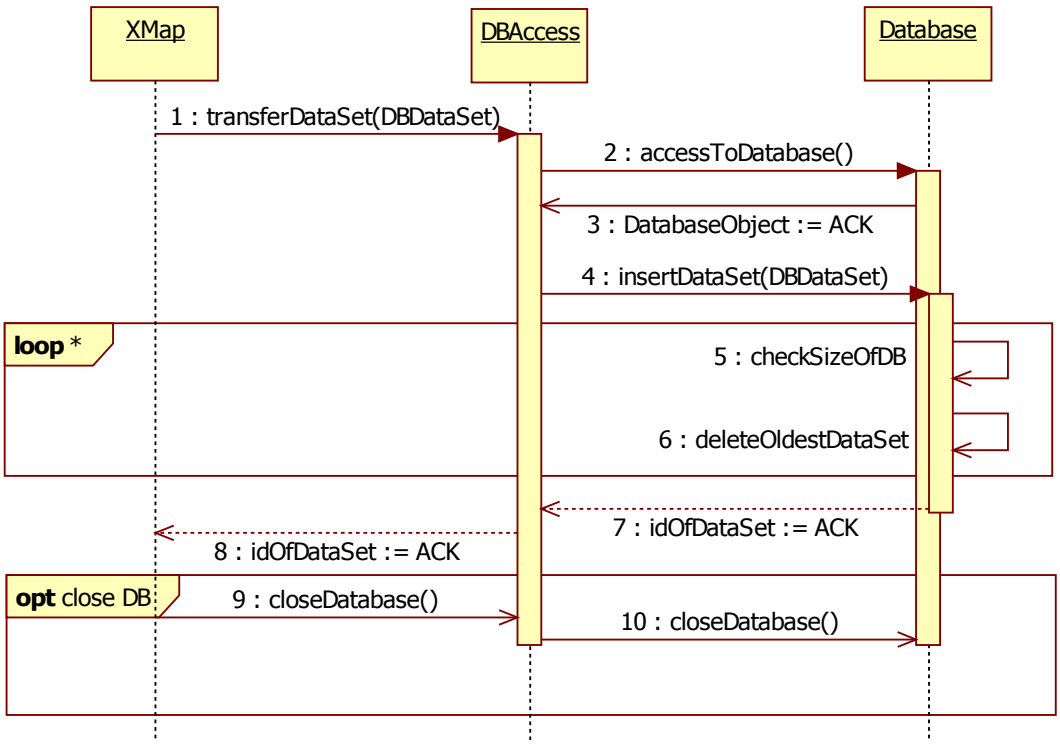

Abbildung 2.5: Sequenzdiagramm zum Löschen von Daten beim Einfügen von Daten

Die X-Map-App möchte einen Datensatz in der Datenbank speichern, wodurch die maximale Speichergröße überschritten wird.
Dazu wird dieser Datensatz an den Datenbankzugriffspunkt gesendet. (1) Danach greift dieser Zugriffspunkt auf die Datenbank zu und erhält ein Datenbankobjekt auf dem dieser Zugriffspunkt arbeitet. (2)+(3) Über dieses wird nun der Datensatz in der Datenbank eingefügt und gespeichert. (4)
In der Folge prüft die Datenbank, ob nun die maximale Speichergröße der Datenbank überschritten wurde. (5) Wenn dies der Fall ist, wird der älteste Datensatz gelöscht. (6)

Dabei werden die Schritte (5) und (6) in einer Schleife solange wiederholt, bis die aktuelle Speichergröße wieder unter dem Maximum liegt.
Danach wird von der Datenbank über dem Datenbankzugriff (7) an die X-Map-App (8) die in der Datenbank gespeicherte "id" des Datensatzes zurückgegeben.
Optional kann danach noch die Datenbank geschlossen werden, falls kein weiter Zugriff benötigt wird. (9)-(10)

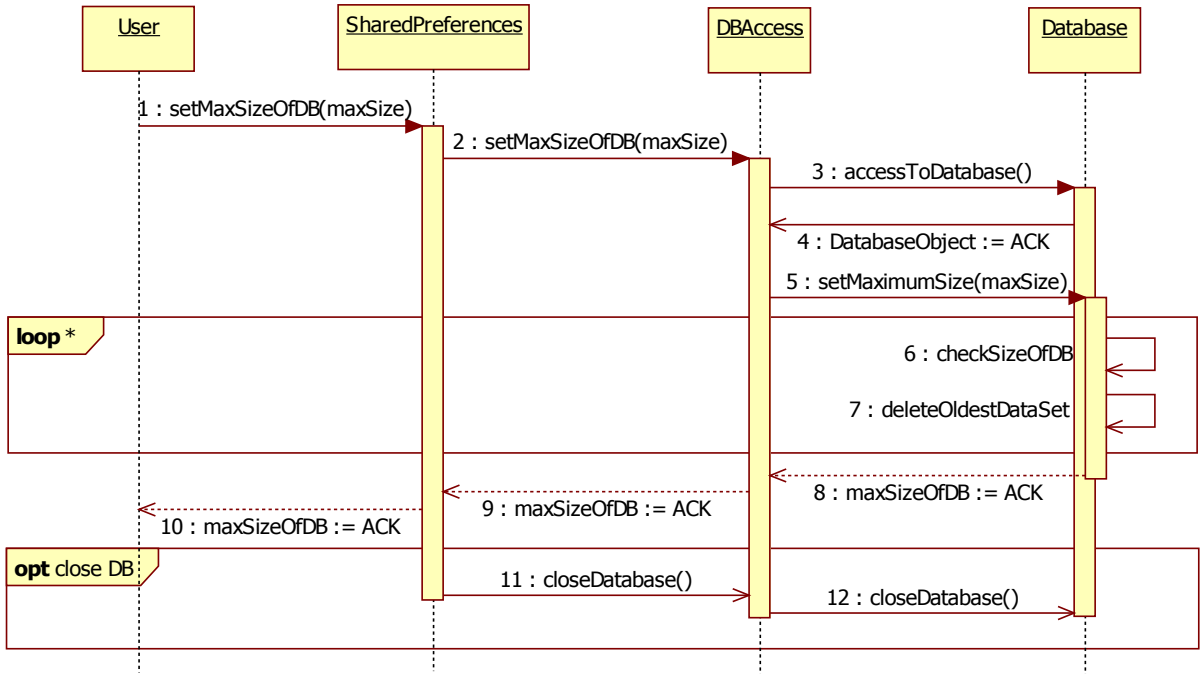

Abbildung 2.6: Sequenzdiagramm zum Löschen von Daten bei einer Einstellungsänderung

Der Benutzer stellt in den Einstellungen die maximale Größe der Datenbank ein. Die neue maximale Speichergröße ist nun kleiner als die aktuelle Datenbankgröße. (1)
Nun wird die X-Map-App über das SharedPreferences-Interface an den Datenbankzugriffspunkt die maximale Speichergröße weiterleiten (2), wodurch dieser Zugriffspunkt den Zugriff auf die Datenbank anfordert und ein Datenbankobjekt für die Weiterverarbeitung erhält. (3)+(4) Über dieses wird die Datenbank Speichergröße neu gesetzt. (5)
Hierfür wird in einer Schleife geprüft, ob die aktuelle Größe der Datenbank noch größer als die maximal zulässige Größe ist. (6) Ist dies der Fall, wird der älteste Datensatz gelöscht. (7) Die Schleife über (6) und (7) wird verlassen, sobald die aktuelle Speichergröße kleiner oder gleich der maximal eingestellten ist. Danach wird dem Benutzer die neu gesetzte maximale Datenbankgröße zurückgegeben. (8)-(10)
Optional kann danach noch die Datenbank geschlossen werden, falls kein weiter Zugriff benötigt wird. (11)+(12)

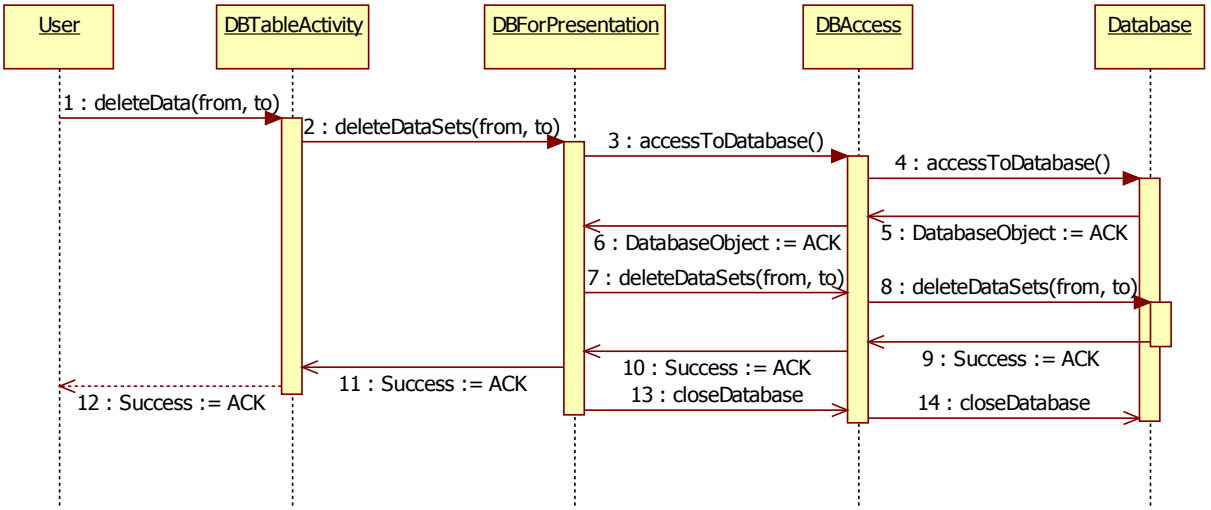

Abbildung 2.7: Sequenzdiagramm zum Löschen von Daten innerhalb eines Zeitraums

Der Benutzer wählt in der Tabellenvisualisierung einen Zeitraum, in dem er die Daten löschen will. Dabei wählt er einen Startzeitpunkt und einen Endzeitpunkt. (1)
Die Tabelle leitet diese Daten an DBForPresentation weiter (2), wo sich als erstes ein Datenbankzugriff geholt wird (3)-(6). Sobald der Datenbankzugriff vorhanden ist, werden alle Datensätze die sich zwischen dem Startzeitpunkt und Endzeitpunkt befinden innerhalb der Datenbank gelöscht. (7)+(8)
Danach wird der Benutzer über den Erfolg der Löschung informiert (9)-(12) und DBForPresentation schließt die Datenbank wieder, da kein weiterer Zugriff mehr benötigt wird. (13)+(14)

2.1.5 Analyse von Funktionalität $\langle F50 \rangle$: Daten lokal speichern

Gemessene Daten werden in einer Datenbank gespeichert.

Der Benutzer startet einen Alarm, im Hintergrund (1). Der Alarm schreibt sich dadurch ins Android-System und wird damit auch gestartet, wenn die App nicht aktiv ist oder das Mobilgerät sich im Standby-Modus befindet. Dieser Alarm ruft sich in bestimmten Zeitabständen selbst als Receiver auf (2). Dieser hat nur eine sehr kurze Lebensdauer, da er für sehr kurze Aktionen vorgesehen ist und vom Android-System selbst kurz nach Aufruf beendet wird. Deshalb startet dieser Receiver einen Service, bzw. kreiert diesen, falls er noch nicht existiert. (3)
Der Service startet den Listener (4), der für das Auslesen des Handynetzes und der GPS-Daten fungiert. Dabei wird über DBAccess ein Zugriff auf die Datenbank ermöglicht (5)+(6) und erhält hierfür ein Datenbankobjekt. (7) Die gesamte Kommunikation zwischen dem Service und der Datenbank erfolgt nun über die Schnittstelle des Datenbankzugriffs. Wenn der Listener Daten gesammelt hat, werden diese in der Datenbank gespeichert (8)+(9) und dem SaveService wird in der Folge durch die “id“ des gespeicherten Datensatzes, über den Erfolg des Speicherns

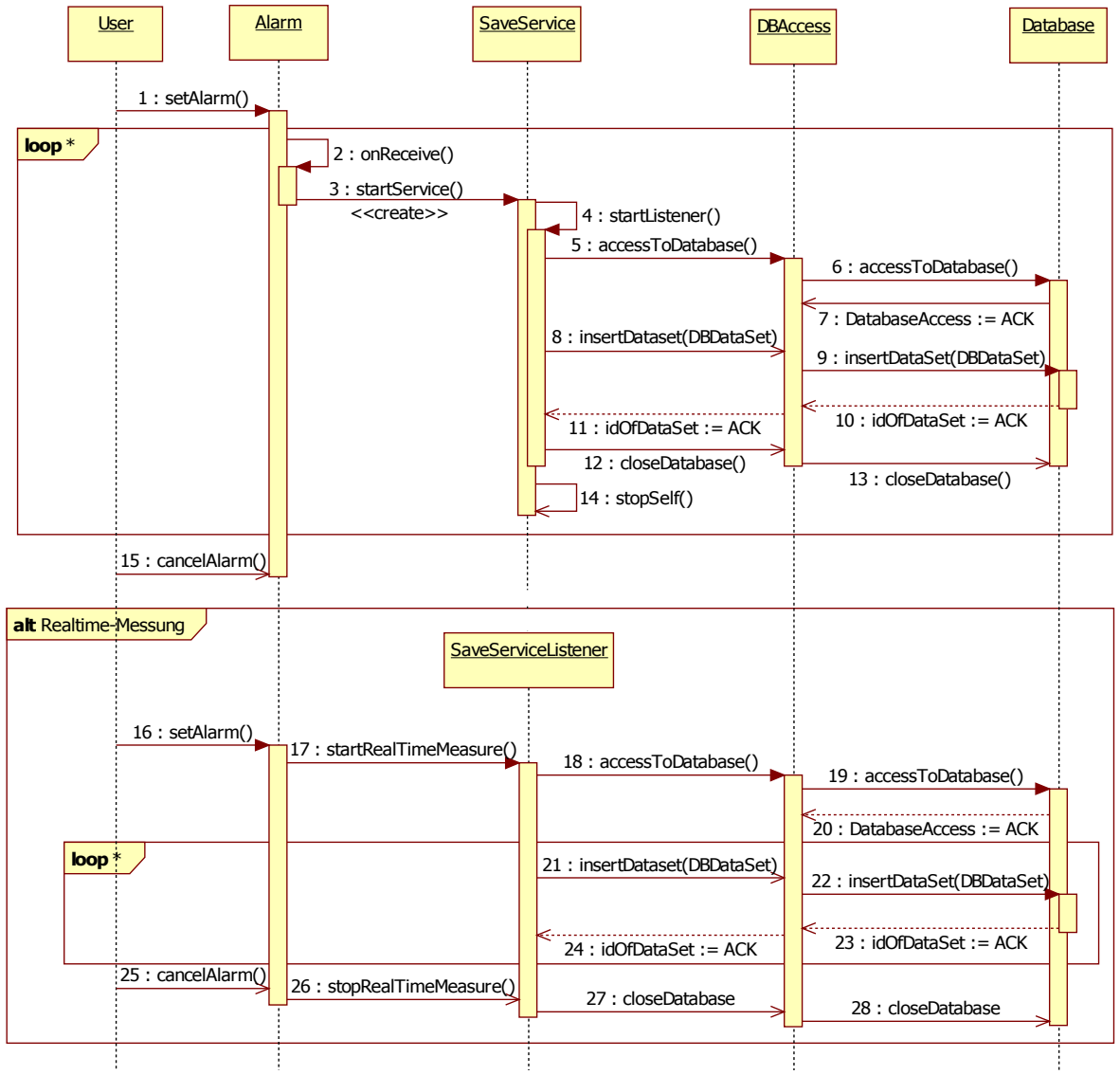

Abbildung 2.8: Sequenzdiagramm zum lokalen Speichern der Daten

informiert. (10)+(11)
Danach hat der Service seine Arbeit beendet, schließt die Datenbank (12)+(13) und stoppt sich selbst. (14)
Dabei werden die Schritte (2)-(14) immer periodisch in einer Schleife wiederholt bis der Benutzer den Alarm beendet. (15)

Alternativ kann der Benutzer auch die Messung in Realzeit durchführen lassen. Dann wird beim Start des Alarms (16) direkt der SaveServiceListener aufgerufen (17), der die Messung durchführt. Der SaveService wird hierbei umgangen.
Als erstes holt sich der SaveServiceListener den Datenbankzugriff (18) - (20), um Daten speichern zu können.
Danach wird bei Veränderung der Signalstärke der aktuelle Datensatz in der Datenbank gespeichert. (21)-(24) Die Signalstärkeänderung soll durch die Schleife, der Schritte (21)-(24), verdeutlicht werden.
Sobald der Benutzer die Messung beendet (25) wird der SaveServiceListener gestoppt (26) und die Datenbank wird geschlossen. (27)-(28)

Zwischen den Schritten (9) und (10) bzw. (22) und (23) kann optional, die in Abbildung 2.5 beschriebene Schleife mit den dortigen Schritten (5)+(6), zum Löschen von Datensätzen, eintreten. (Wird hier nicht näher spezifiziert.)

2.1.6 Analyse von Funktionalität $\langle F60 \rangle$: Einstellungen für lokal zu speichernde Daten

Einstellungen für lokal zu speichernde Daten werden vom Nutzer festgelegt.

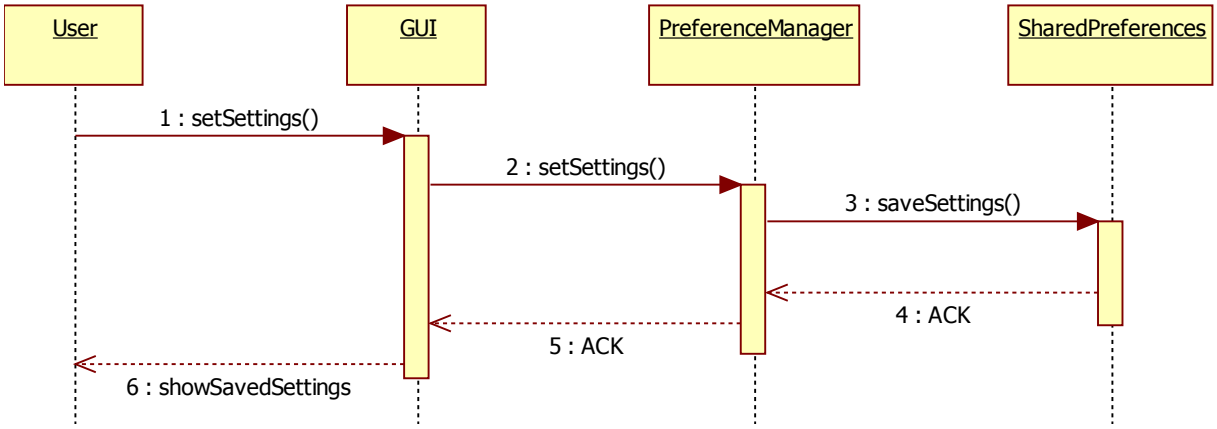

Abbildung 2.9: Sequenzdiagramm für Einstellungen lokal zu speichernder Daten

Der Nutzer wählt zu speichernde Parameter in der App aus:

- Messdaten, die lokal gespeichert und an den WebService übertragen werden (inkl. Datenschutzeinstellungen bzgl. Gerätebezug (ehemalig $\langle F70 \rangle$)
- Die Häufigkeit der Messdatenübertragung
- Maximale Speichergröße der Datenbank

Sind die gewünschten Parameter ausgewählt, werden sie von der GUI an den PreferenceManager übergeben. Dies ist eine von Android zur Verfügung gestellte Klasse zur Verwaltung von Nutzereinstellungen. Der PreferenceManager speichert die Einstellungen mit Hilfe des Interfaces SharedPreferences in einer XML-Datei. SharedPreferences stellt einen Mechanismus für Speicherung von und Zugriff auf Einstellungen bereit.

2.1.7 Analyse von Funktionalität $\langle F80 \rangle$: Registrierung eines Benutzers

Ein Benutzer gibt auf dem Mobilgerät seine E-Mail-Adresse und ein Passwort ein. Diese Daten werden dann an den Server übermittelt, welcher die Daten mit der bestehenden Benutzdatenbank abgleicht und entsprechend die Daten einträgt, oder nicht.

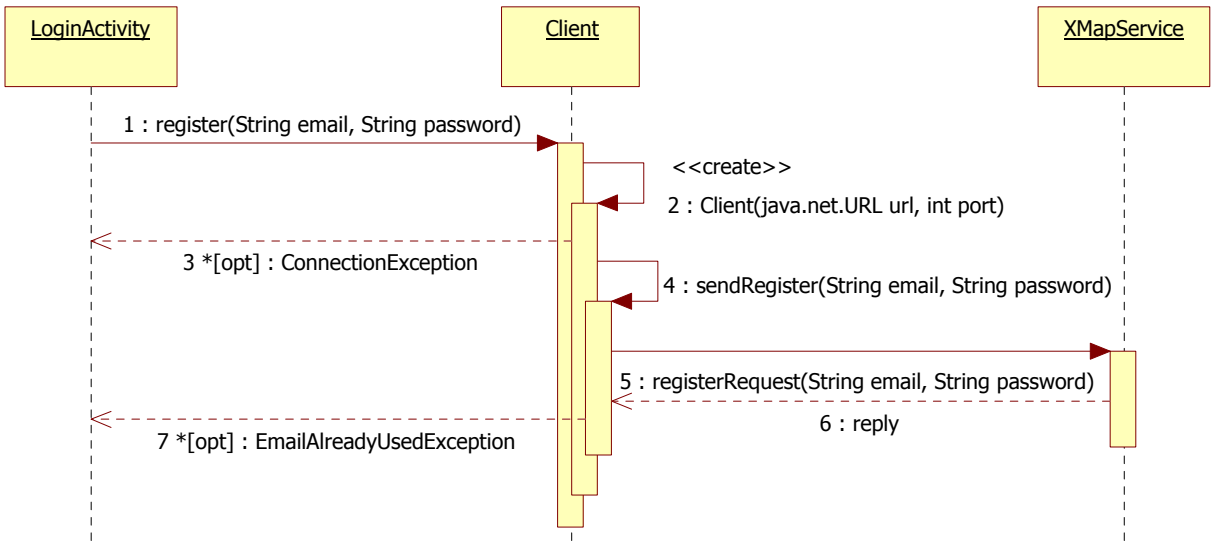

Abbildung 2.10: Sequenzdiagramm zum Registrieren eines Benutzers

Die statische Methode register wird mit Email-Adresse und Passwort als Parameter aufgerufen (1). Danach wird der Client-Konstruktor gestartet (2). Dabei kann wie beim Login eine ConnectionsException ausgelöst (3) werden, wenn es Verbindungsprobleme geben sollte. Intern wird die private Methode sendRegister aktiviert (4), welche dann ein RegisterRequest (5) an den Server sendet. Das Ergebnis des Servers (6) kann entweder eine Erfolgsnachricht sein, oder ein Fehler. Im letzteren Fall wird eine EmailAlreadyUsedException (7) erstellt. Entstehen keine Exceptions war die Registrierung erfolgreich.

2.2 Server

Die folgenden Funktionalitäten $\langle F90 \rangle$ bis $\langle F130 \rangle$ sind dem Server zugeordnet.

2.2.1 Analyse von Funktionalität $\langle F90 \rangle$: Anmeldung

Ein Benutzer gibt seine Anmeldedaten auf dem Mobilgerät ein. Diese Daten werden an den Server übermittelt, mit der Benutzerdatenbank abgeglichen, woraufhin entweder die Anmeldung zurückgewiesen oder das Mobilgerät am Server bekanntgemacht wird.

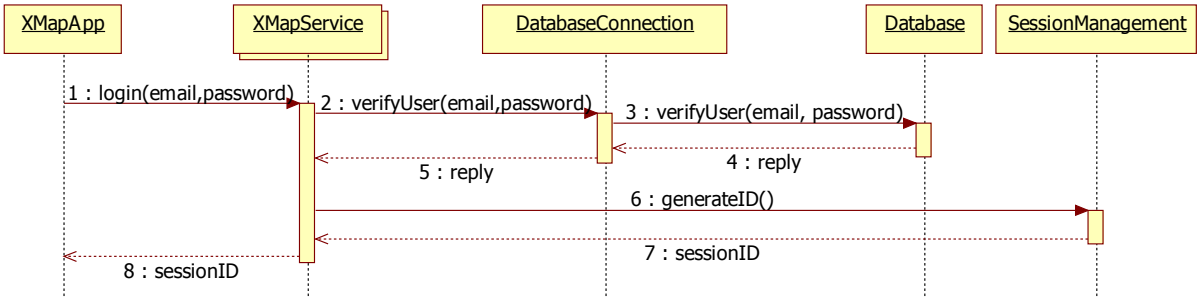

Abbildung 2.11: Sequenzdiagramm zum Anmeldevorgang

Die App ruft über einen Fernaufruf auf dem (mutlithreaded) WebService die Login-Prozedur mit E-Mail-Adresse und Passwort als Parameter auf. (1) Der Service fragt nun über die Datenbankverbindungsschicht bei der Datenbank an, ob eingegebene Werte korrekt sind, woraufhin eine entsprechende Antwort erfolgt. (2)-(5) Ist der Login korrekt, wird mithilfe des SessionManagements eine SessionID generiert und der App als Antwort übergeben. (6)-(8) Ist der Login inkorrekt wird keine ID generiert und eine negative SessionID zurückgeliefert.

2.2.2 Analyse von Funktionalität $\langle F100 \rangle$: Registrierung eines neuen Mobilgeräts

Ein Benutzer meldet sich mit einem bisher unbekannten Mobilgerät an. Das Mobilgerät wird, sofern im Datenschutz erlaubt, dem Nutzer zugeordnet und kann von nun an Daten übermitteln.

[A] := manufacturer, device, phoneType, networkOperator, networkOperatorName, countryCode

Die App ruft über einen Fernaufruf auf dem (multithreaded) WebService die Geräte-Registrierungsprozedur mit ihrer aktuellen sessionID sowie den Parametern aus [A] auf. (1)
Es wird überprüft, ob die SessionID gültig ist. (2)+(3)
Im Erfolgsfall wird die DatabaseConnection mit der Registrierung des Gerätes betraut. (4)-(8)

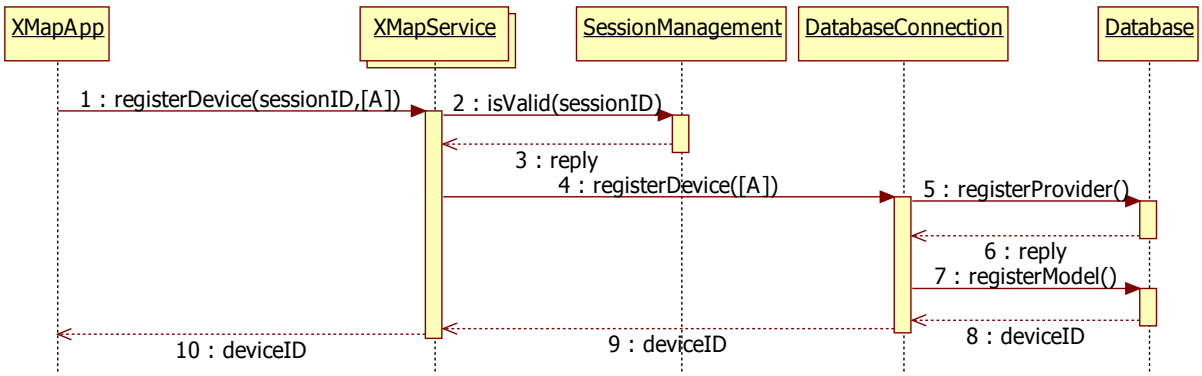

Abbildung 2.12: Sequenzdiagramm zum Geräte-Registrierungsvorgang

Die zurückgegebene deviceID wird dem Gerät dann übergeben. (9)+(10)
Das Gerät ist nun registriert.

2.2.3 Analyse von Funktionalität $\langle F110 \rangle$: Speicherung der Daten

Ein neu erhaltener Messdatensatz wird bei gültiger Session-ID in der Datenbank gespeichert.

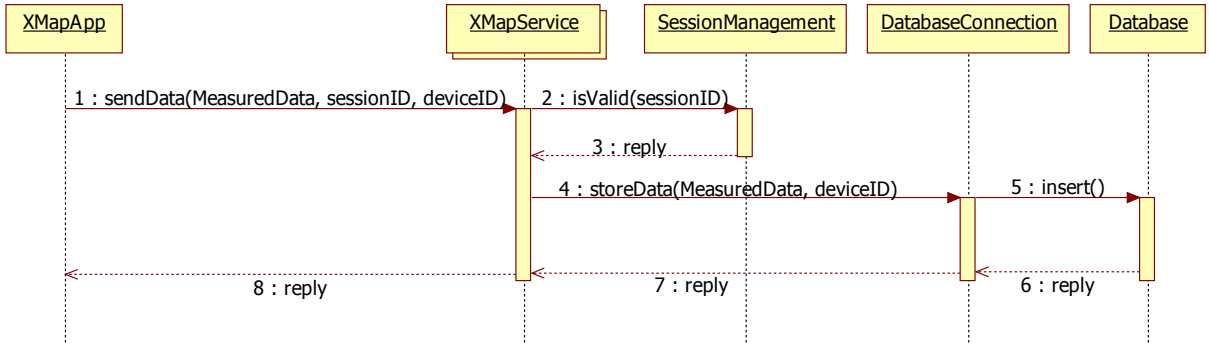

Abbildung 2.13: Sequenzdiagramm zur Speicherung der Daten

Die App sendet einen neuen Messdatensatz mit der aktuellen SessionID und der DeviceID an den WebService. Dieser überprüft, ob die SessionID gültig ist und speichert, wenn ja, die neuen Messdaten mit der DeviceID in der Datenbank.

2.2.4 Analyse von Funktionalität $\langle F120 \rangle$: Auslesen der Daten

Zu Auswertungszwecken werden über den WebService Datensätze aus der Datenbank ausgelesen.

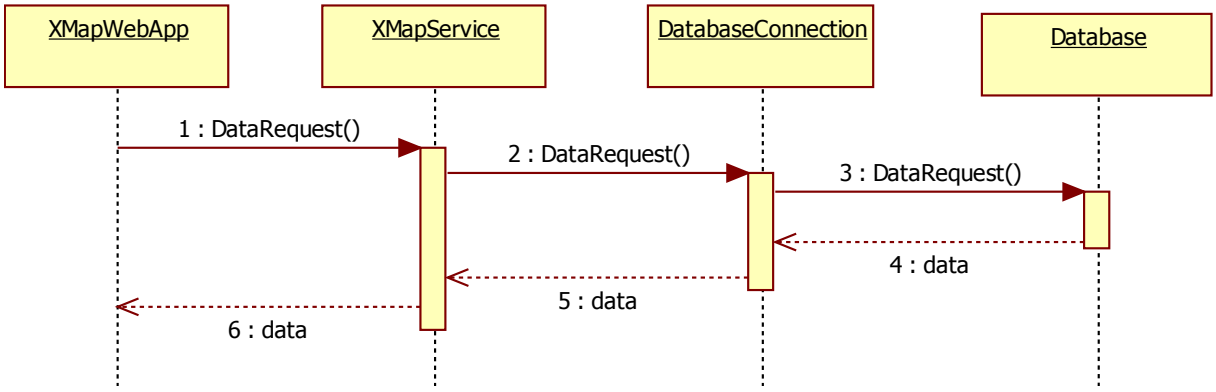

Abbildung 2.14: Sequenzdiagramm zum Auslesen der Daten

Die Web-Applikation sendet eine Datenanfrage über den WebService an die Datenbank. Diese gibt die benötigten Daten auf selben Wege zurück an die WebApp.

2.2.5 Analyse von Funktionalität $\langle F130 \rangle$: Visualisierung der Daten

Die Web-Applikation benutzt den WebService um Daten aus der Datenbank zu erhalten, welche daraufhin visualisiert werden können.

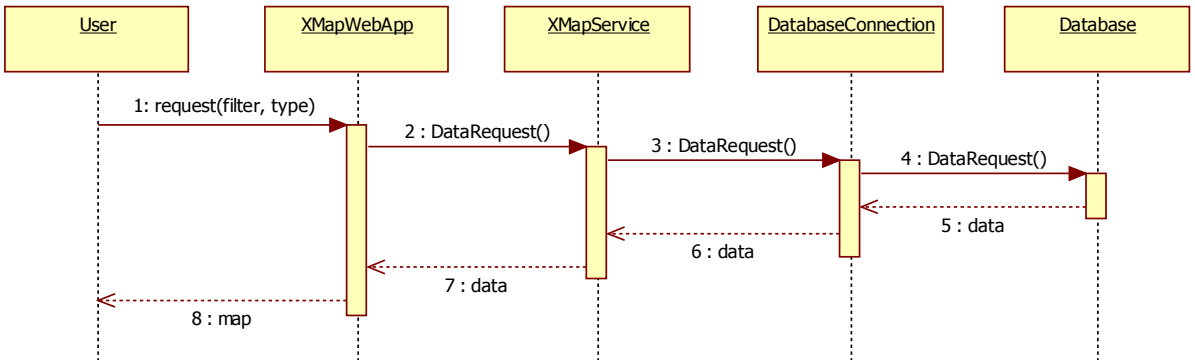

Abbildung 2.15: Sequenzdiagramm zum Visualisieren der Daten

Die Web-Applikation erhält Informationen vom Benutzer welche Daten er auf welche Art visualisiert bekommen möchte und sendet dementsprechend eine Datenanfrage raus. Sobald diese Daten bei der Web-Applikation eingetroffen sind werden sie dem Benutzer angezeigt.

2.3 Nicht-Funktionale Anforderungen

Dieser Abschnitt beschreibt die Nicht-Funktionale Anforderungen in Sequenzdiagrammen. Dabei werden $\langle Q10 \rangle$, $\langle Q20 \rangle$ und $\langle Q30 \rangle$ sowie $\langle Q40 \rangle$ und $\langle Q50 \rangle$ in jeweils einem Sequenzdiagramm zusammengefasst. $\langle Q60 \rangle$ besitzt ein eigenständiges Sequenzdiagramm.

2.3.1 Analyse der Nicht-Funktionalen Anforderungen $\langle Q10 \rangle$, $\langle Q20 \rangle$ und $\langle Q30 \rangle$

Der Teilabschnitt beschreibt mittels Sequenzdiagramm in Abbildung 2.16, wie der Benutzer auf eine zu lange Antwortzeit hingewiesen wird. Die hier benutzten Anforderungen sind:

- $\langle Q10 \rangle$ Die Antwortzeit darf für jegliche Benutzeranfrage nicht mehr als drei Sekunden betragen.
- $\langle Q20 \rangle$ Der Benutzer wird informiert, sobald die Antwortzeit länger als drei Sekunden ist.
- $\langle Q30 \rangle$ Eine Fortschrittsanzeige wird bei einer Verzögerung zur Verfügung gestellt.

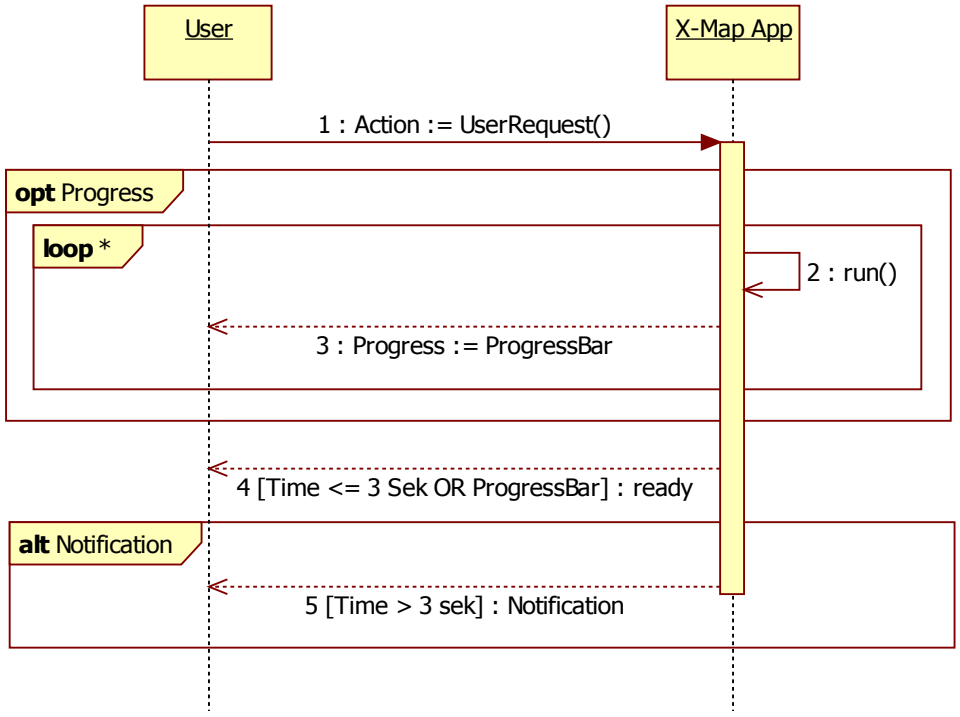

Abbildung 2.16: Sequenzdiagramm zu $\langle Q10 \rangle$, $\langle Q20 \rangle$ und $\langle Q30 \rangle$

Der Benutzer startet eine Benutzeranfrage an die X-Map App (1), welche eine längere Berechnung ausführen muss. Dazu kann die App dem Benutzer optional während sie arbeitet (2) eine Fortschrittsanzeige ausgeben (3), wenn schon bei Beginn der Berechnung bekannt ist, dass folgende Berechnung länger dauern kann. Welche Berechnung eine Fortschrittsanzeige, von Beginn

an erhält, wird von den Entwicklern durch gezieltes Testen bestimmt.
Am Ende der Berechnung arbeitet die App ohne weitere Information an den Benutzer weiter. Dazu hat die App entweder die Berechnung innerhalb von 3 Sekunden vollendet oder der Benutzer wurde mittels Fortschrittsanzeige über den Berechnungsfortschritt auch über die 3 Sekunden hinaus informiert. (4)
Wenn kein Fortschritt angezeigt wird und die Berechnung länger als 3 Sekunden benötigt, wird dem Benutzer alternativ nach 3 Sekunden eine Benachrichtigung angezeigt, dass die Berechnung noch nicht vollendet ist.

2.3.2 Analyse von den Nicht-Funktionalen Anforderung $\langle Q40 \rangle$ und $\langle Q50 \rangle$

Der Teilabschnitt beschreibt die Fehlerbehandlung durch den Benutzer. Dazu soll der Benutzer bei Fehlern möglichst nicht eingreifen müssen. Dies wird im Sequenzdiagramm auf Abbildung 2.17 dargestellt. Die hier benötigten Nicht-Funktionalen Anforderungen sind:

- $\langle Q40 \rangle$ Bei Aussetzen von Datenübertragung, Positionsbestimmung oder Messungen ist keine Interaktion des Benutzers nötig.
- $\langle Q50 \rangle$ Fehler bei Hintergrundfunktionen werden dem Anwender durch ein Fehlerprotokoll zugänglich gemacht.

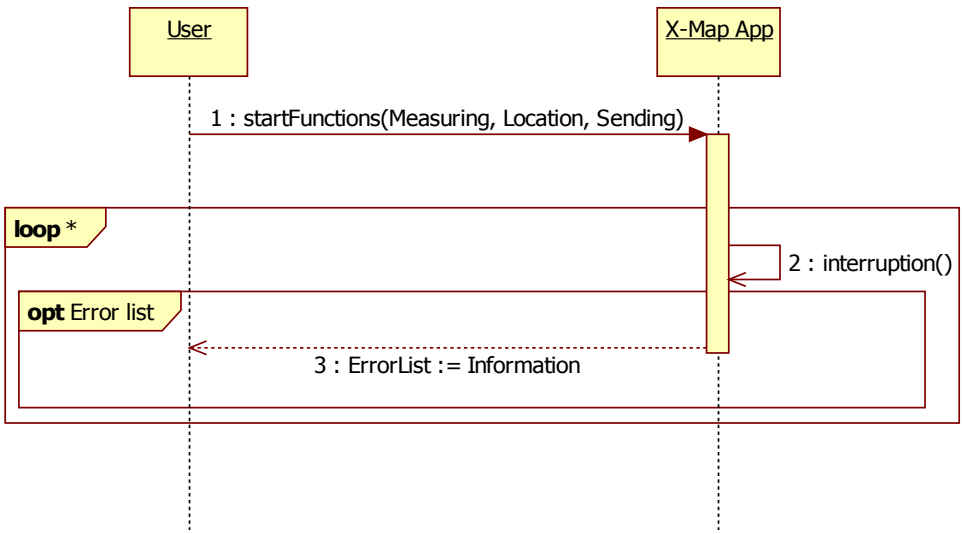

Abbildung 2.17: Sequenzdiagramm zu $\langle Q40 \rangle$ und $\langle Q50 \rangle$

Der Benutzer startet eine oder mehrere der Funktionen, die für die Datenübertragung, Positionsbestimmung und Messungen vorhanden sind. (1)
Während die Funktionen ihre Aufgaben erfüllen, kann es zu Aussetzern verschiedener Formen kommen. (2) Da diese Aussetzer mehrfach auftreten können, wird dieses in einer loop-Schleife

dargestellt.
Der Benutzer kann dabei optional ein Fehlerprotokoll einsehen, um sich über mögliche Aussetzer und Fehler zu informieren. (3) Das Fehlerprotokoll wird dabei als optional gekennzeichnet, da dieses ein Kann-Kriterium $\langle RC1 \rangle$ ist und somit noch nicht mit in die bisherige Planung einbezogen wurde.

2.3.3 Analyse von der Nicht-Funktionalen Anforderung $\langle Q60 \rangle$

In diesem Teilabschnitt wird die Spracheinstellung der App in einem Sequenzdiagramm in Abbildung 2.18 beschrieben. Die hier beschriebene Nicht-Funktionale Anforderung ist:

- $\langle Q60 \rangle$ Die Verkehrssprache ist Englisch und Deutsch.

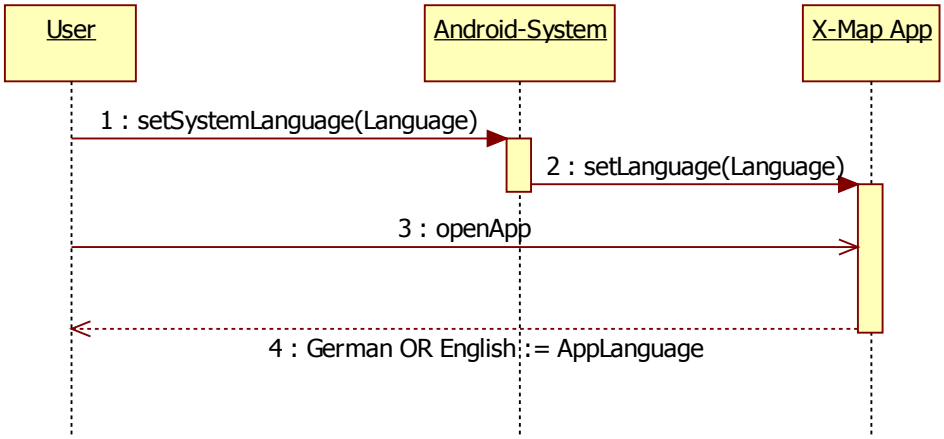

Abbildung 2.18: Sequenzdiagramm zu $\langle Q60 \rangle$

Die Sprachauswahl der App, wird über die Android-Systemsprache festgelegt. Demnach setzt der Benutzer die Sprache des Mobilgerätes auf eine von ihm gewünschte Sprache. (1)
Dadurch passt die App sich der eingestellten System-Sprache an, indem das Betriebssystem der App die eingestellte Sprache übermittelt. (2)
Danach kann der Benutzer die App öffnen (3), wodurch ihm die Sprache der App angezeigt wird. (4) Dazu wird bei deutscher Systemsprache, die App ebenfalls in Deutsch dargestellt. Bei allen anderen Systemsprachen, wird die App auf Englisch dargestellt.

3 Resultierende Softwarearchitektur

Durch eine vorhergehende Analysephase wird für die X-Map App sowohl client- als auch serverseitig eine 3-Schichten-Architektur verfolgt. Unterschieden wird dabei zwischen einer GUI- bzw. Service-Schicht, einer Fachkonzeptschicht und einer Datenhaltungsschicht.
Es folgt ein genauerer Einblick in dieses System.

3.1 Komponentenspezifikation

Im Folgenden wird die Komponentenstruktur von X-Map näher beschrieben. Das zentrale Element dieses Kapitels ist Abbildung 3.1, welche einen Überblick über die Komponenten des Systems liefert.

3.1.1 Client

In diesem Kapitel werden die Komponenten der Client-Oberkomponente besprochen:

Komponente $\langle C10 \rangle$: Client
Der Client beschreibt ein Mobilgerät. Der Client ermöglicht alle Operationen die notwendig sind. Dabei arbeitet der Client auf einer 3-Schichten-Architektur, um eine erweiterbare Architektur zu gewährleisten.
Der Client kommuniziert mit der Komponente Server ($\langle C20 \rangle$), um Daten an diesen zu übertragen.

Komponente $\langle C11 \rangle$: UserGUI
Die UserGUI dient zur direkten Kommunikation mit dem Nutzer. Die Auswertung und Anzeige der lokal gemessenen Daten kann durch eine Tabelle, einen Graphen oder eine Google Maps erfolgen. Bei der Google Maps wird die Empfangsqualität durch ein farbiges Overlay dargestellt.

Komponente $\langle C12 \rangle$: Backend
Das Backend ist eine zentrale Steuerungs- und Kontrolleinheit. Es übernimmt u.a. die Datenmessung, Speicherung sowie Kommunikation mit dem Server.

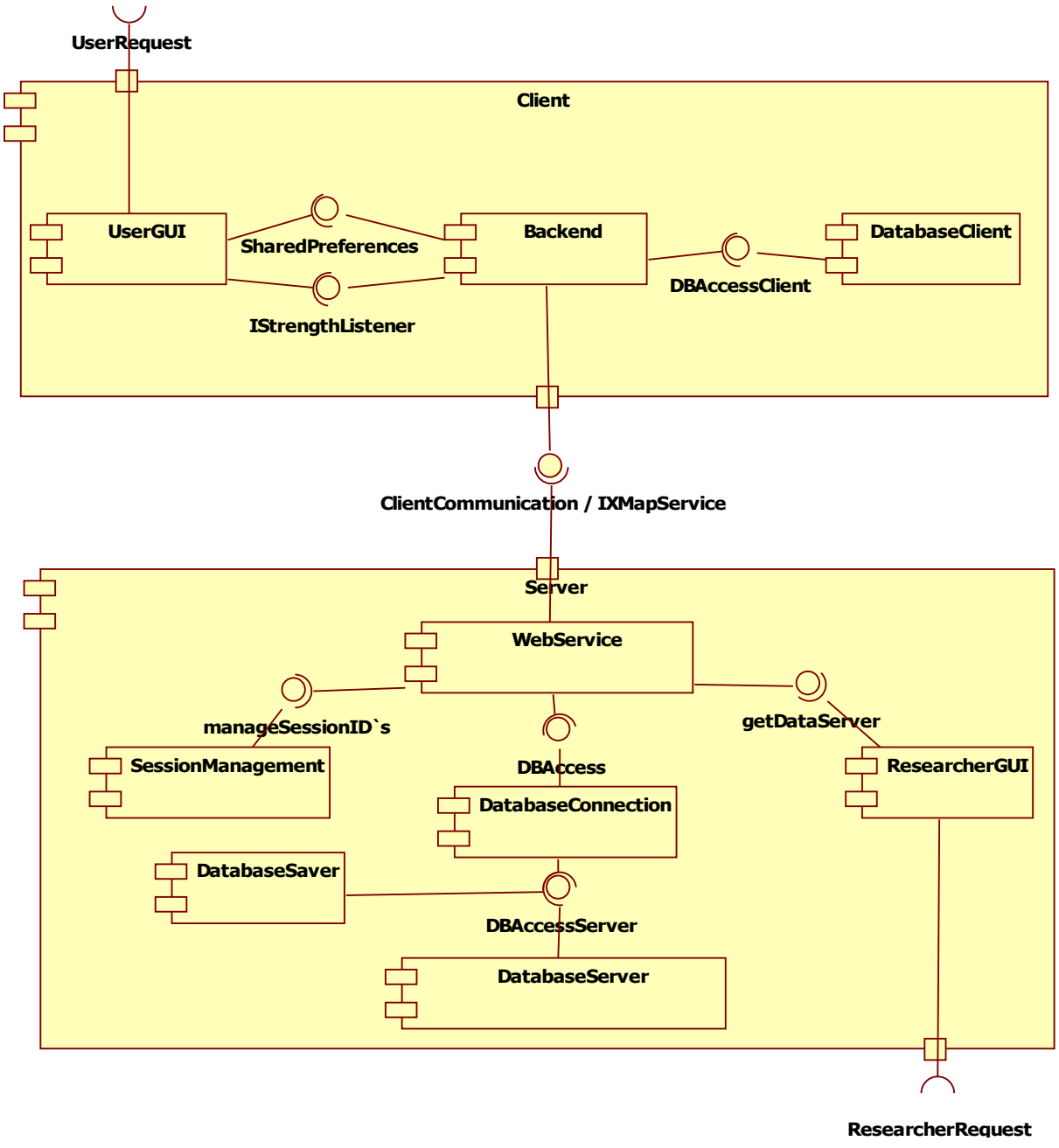

Abbildung 3.1: Komponentendiagramm

Komponente $\langle C13 \rangle$: DatabaseClient
Der DatabaseClient stellt die Datenbank des Clients dar. Es stellt auch Methoden zu Verfügung, um die Operationen auf der eigentlichen Datenbank zu vereinfachen.

3.1.2 Server

Dieses Kapitel enthält die Beschreibungen der Komponenten des Servers:

Komponente $\langle C20 \rangle$: Server
Der Server bietet die Basis für die Ausführung des WebServices und der WebApplikation.

Komponente $\langle C21 \rangle$: WebService
Der WebService stellt Methoden zum Registrieren und Anmelden von Usern, sowie zum Annehmen und Auslesen von Messdaten bereit.

Komponente $\langle C22 \rangle$: SessionManagement
Das SessionManagement ist ein statisches Objekt, das über alle Instanzen des WebService eine threadsichere Liste von SessionID's verwaltet. SessionID's werden bei der Anmeldung an Geräte übergeben und dienen dazu, User zu authentifizieren, ohne die Login-Daten mehrfach übertragen zu müssen.

Komponente $\langle C23 \rangle$: DatabaseConnection
Die DatabaseConnection stellt dem WebService einfache Methoden zur Verfügung, in welchen z.T. komplexe Operationen auf der eigentlichen Datenbank gekapselt sind.

Komponente $\langle C24 \rangle$: DatabaseServer
DatabaseServer symbolisiert die Datenbank, welche User- und Messdaten verwaltet und bereitstellt.

Komponente $\langle C25 \rangle$: ResearcherGUI
Die Komponente ResearcherGUI stellt die Web-Applikation dar, welche zur Anzeige und Auswertung der gemessenen Daten dient.

Komponente $\langle C26 \rangle$: DatabaseSaver
DatabaseSaver ist eine Anwendung, die auf der Datenbankverbindung aufsetzt und bei Ausführung die Einträge der Entität MeasuredData aus der Datenbank strukturiert in XML-Dateien ablegt.

3.2 Schnittstellenspezifikation

Die in Abbildung 3.1 abgebildeten Schnittstellen der einzelnen Komponenten und ihre zugehörigen Methoden werden in den folgenden Tabellen näher erläutert.

Schnittstelle $\langle I10 \rangle$: DBAccessClient

Operation	Beschreibung
accessToDatabase()	Wird von einer Klasse des Clients aufgerufen, die Zugriff auf die Datenbank benötigt. Die Datenbank erzeugt ein Datenbankobjekt, das für die Klasse in der Schnittstelle bereitgestellt wird.
insertDataset()	Es soll ein Datensatz in der Datenbank gespeichert werden. Dazu wird über die Schnittstelle der gewünschte Datensatz an die Datenbank weitergeleitet, welche diesen nun in sich aufnimmt.
getCursor()	Es müssen Datensätze aus der Datenbank ausgelesen werden. Die Datenbank liest die gewünschten Datensätze aus und speichert diese in einem Cursor. Dieser wird dann der Schnittstelle übergeben und an die Backend-Schicht weitergeleitet. Diese Methode gibt es in verschiedenen Ausführungen, die je nach geladener Visualisierung verschiedene Daten enthält.
closeDatabase()	Wird von einer Klasse des Clients aufgerufen. Die Klasse benötigt den Zugriff auf die Datenbank und muss explizit diese für sich schließen lassen.
deleteDataSets(from, to)	Löscht alle Daten, die sich innerhalb dem Zeitintervall, welches sich aus den Werten from und to bildet. Dazu wird über die Schnittstelle das gewünschte Zeitintervall an die Datenbank weitergeleitet, welche alle Datensätze die in dem Zeitintervall liegen gelöscht.

Schnittstelle $\langle I20 \rangle$: IStrengthListener

handleStrengthChanged()	GoogleMapsActivity, XYChartActivity sowie DBTableActivity sind Listener von der Singleton Backend Klasse StrengthListener. Dafür implementieren sie das Interface IStrengthListener mit der Methode handleStrengthChanged(). Sobald neue Nutzerdaten gemessen und in der Datenbank gespeichert werden, wird im Backend über alle, in einer Liste gesammelten, Listeners iteriert und handleStrengthChanged() aufgerufen. In HandleStrengthChanged() wird das Verhalten der einzelnen Visualisierungstypen bei neuen Daten implementiert. Auf diese Weise kann das Backend ohne direkte Abhängigkeit das Front-end (GUI) notifizieren.

Schnittstelle $\langle I30 \rangle$: ClientCommunication

getInstance()	Gibt ein übertragungsbereites Objekt der Singleton-Klasse Client zurück.
sendData()	Sendet ein Array aus DBDataSet Objekten an den Server.
registerUser()	Registriert einen Nutzer am WebService
registerDevice()	Registriert ein Gerät am WebSservice

Schnittstelle $\langle I40 \rangle$: SharedPreferences

Operation	Beschreibung
getDefaultSharedPreferences()	Wird von einer Klasse des Clients aufgerufen, die Lesezugriff auf die gespeicherten Einstellungen benötigt.
edit()	Wird von einer Klasse des Clients aufgerufen, die Schreibzugriff auf die gespeicherten Einstellungen benötigt.
commit()	Wird aufgerufen, um veränderte Wertepaare zu speichern.
onSharedPreferenceChanged()	Listener eines Wertepaares. Wird von einer Klasse des Clients implementiert, die bei Änderung des Wertepaares benachrichtigt werden muss.

Schnittstelle $\langle I50 \rangle$: IXMapService

Operation	Beschreibung
sendData()	Wird vom Mobilgerät aufgerufen, um Daten zur Speicherung an die Datenbank zu übertragen.
endSession()	Mobilgerät übergibt dem WebService seine SessionID, welche daraufhin durch das SessionManagement entfernt wird.
registerUser()	Mobilgerät überträgt E-Mail-Adresse, Passwort und Telefon-Spezifika an den WebService, sodass ein neuer User erstellt und das Gerät der Gerätedatenbank hinzugefügt werden kann.
login()	Mobilgerät überträgt Login-Daten des Nutzers zur Authentifizierung an den WebService, woraufhin eine SessionID generiert und zurückgegeben wird.

Hinweis: Die Interfaces ClientCommunication $\langle I30 \rangle$ und IXMapService $\langle I50 \rangle$ sind zwei Sichtweisen auf die gleiche Verbindung. $\langle I30 \rangle$ ist hierbei die Sicht des Clients, $\langle I50 \rangle$ die des Servers.

Schnittstelle $\langle I60 \rangle$: manageSessionID's

Operation	Beschreibung
isValidID()	Sucht unter den SessionIDs nach der angefragten SessionID und überprüft gegebenenfalls den zugehörigen Zeitstempel. Gibt einen Boolean-Wert mit der Antwort zurück.
blockingDeleteID() und nonBlockingDeleteID()	Beide Methoden löschen die übergebene SessionID. Die blockierende Methode kehrt hierbei erst zurück, wenn die Löschung erfolgt ist. Die nicht-blockierende kehrt sofort zurück, ungeachtet ob das Löschen bereits erfolgt ist.
generateID()	Generiert eine neue SessionID, trägt sie in die entsprechende Datenstruktur ein und gibt sie daraufhin zurück.

Schnittstelle $\langle I70 \rangle$: DBAccess

Operation	Beschreibung
insertData()	Erhält einen Messdatensatz und speichert diesen in der Datenbank. Gibt zurück, ob das Speichern geklappt hat oder nicht.
userInDB()	Überprüft, ob sich die übergebenen Benutzerdaten, insbesondere die E-Mail-Adresse, schon in der Userdatenbank befindet.
storeNewUser()	Speichert die übergebenen Benutzerdaten mit einer User-ID in der Datenbank.
getData()	Liest die gewünschten Daten aus der Datenbank aus

Schnittstelle $\langle I80 \rangle$: DBAccessServer

Es werden parametrisierte Anfragen verwendet, um mit SQL-Statements auf die Daten, die in der Datenbank gespeichert sind, zuzugreifen.

Schnittstelle $\langle I90 \rangle$: getDataServer

Die Methoden zur Datenabfrage für die Web-Applikation (im Diagramm „ResearcherGUI“) konnten noch nicht entworfen und implementiert werden. Angegeben sind daher nur die aktuell wahrscheinlichen Methoden:

Operation	Beschreibung
registerResearcher()	Methode zur Registrierung eines neuen Benutzers mit Forschungszugriff.
loginResearcher()	Methode zum Login eines bestehenden Benutzers mit Forschungszugriff.
getFilters()	Gibt die aktuell zur Verfügung stehenden Möglichkeiten zum Filtern von Messergebnissen zurück.
getOverlayData() und getTableData()	Liefert jeweils angepasste Datensätze für die Ausgabe über ein Kartenoverlay bzw. über eine Tabellendarstellung.

3.3 Protokolle für die Benutzung der Komponenten

In diesem Abschnitt wird die korrekte Verwendung der einzelnen Komponenten anhand von Statecharts dargestellt. Die Komponenten die wiederverwendet werden, werden in den dazugehörigen Teilabschnitt beschrieben.

3.3.1 Komponente $\langle C10 \rangle$: Client

In diesem Teilabschnitt wird die korrekte Verwendung des Clients beschrieben.

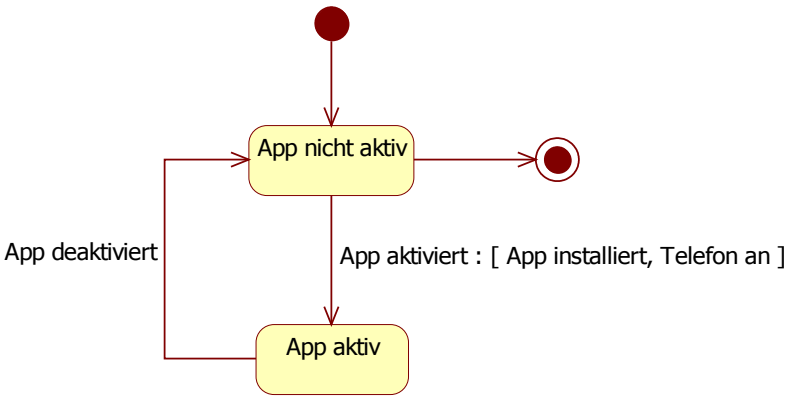

Abbildung 3.2: StateChart zur Komponente Client $\langle C10 \rangle$

Die Komponente Client besitzt, abgesehen vom Start- und Finalzustand, nur zwei Zustände. Solange die App nicht aktiv ist, passiert nichts in der Komponente. Um die App aktivieren zu können, muss sie auf dem Mobilgerät installiert und das Mobilgerät eingeschaltet sein.
Aktiv bedeutet, dass entweder die App im Vordergrund läuft, also das man sich in einem Zustand der UserGUI $\langle C11 \rangle$ befindet, oder die App Messungen und Übertragungen im Hintergrund durchführt, sich also kein Zustand der UserGUI $\langle C10 \rangle$ zuordnen lässt. Während letzterem Vorgehen kann sich das Smartphone selbst auch im Standby-Modus befinden.
Im aktiven Zustand agieren die Komponenten $\langle C12 \rangle$, $\langle C13 \rangle$ und ggf. $\langle C11 \rangle$ im Zusammenspiel miteinander und arbeiten ihre jeweiligen Abläufe ab.
Wenn man die App deaktiviert, kann sie manuell reaktiviert werden, oder aber sie aktiviert sich selbstständig, um Messungen und Übertragungen durchzuführen.

3.3.2 Komponente $\langle C11 \rangle$: UserGUI

Dieser Teilabschnitt beschreibt die korrekte Verwendung der UserGUI.

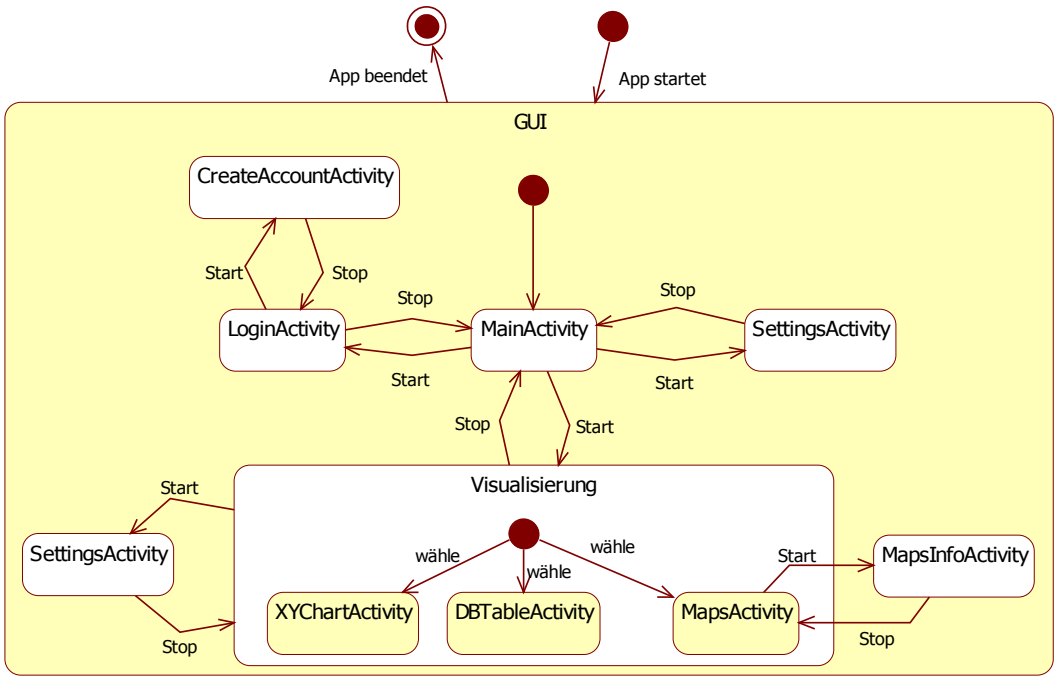

Abbildung 3.3: StateChart zur Komponente UserGUI $\langle C11 \rangle$

Die Komponente UserGUI dient zur direkten Kommunikation mit dem Nutzer. Die MainActivitiy ist der Startbildschirm beim Öffnen der X-Map App. Durch Auswahloptionen kann zum einen eine Anmeldung (LoginActivity) erfolgen. Falls der Nutzer noch kein Konto eingerichtet hat, kann er sich registrieren (CreatAccountActivity). Zum anderen können diverse Einstellungen ausgewählt werden (SettingsActivity).
Über einen Visualisierungsbutton kann der Nutzer seine Messdaten durch verschiedene Visualisierungstypen auswerten: Eine Tabelle (DBTableAcitivy), eine Google Maps (MapsActivity) oder einen XY-Graphen (XYChartActivity). Befindet sich der Nutzer in einer Visualisierung, kann er von dort aus auch direkt den Einstellungsbildschirm (SettingsActivity) erreichen. Von hier kann er wiederrum zur Visualisierung zurückkehren. Von der MapsActivity aus, kann man zusätzlich auf die MapsInfoActivity zugreifen, um Informationen über das Google Maps Overlay zu erhalten. Von dort aus kann man wieder zu Google Maps zurückkehren.
Der Nutzer kann jederzeit zum Startbildschirm zurückkehren, außer er befindet sich in der MapsInfoActivity oder der von der Visualisierung aufgerufenen SettingsActivity. Von jeder Activity aus kann der Benutzer die App und somit die GUI beenden.

Für eine Applikation mit einer ähnlichen Schichten-Architektur ist die UserGUI für eine Wiederverwendung geeignet.

3.3.3 Komponente $\langle C12 \rangle$: Backend

Im Backend werden Messungen durchgeführt und Daten verwaltet. Das beinhaltet deren Speicherung und Übertragung. Das StateChart geht auch nur auf diese beiden Funktionen ein. Es gibt zwar noch weitere Funktionen und Berechnungen, die in dieser Komponente durchgeführt werden, jedoch sind diese zustandslos und werden deshalb nicht weiter betrachtet.

Backend Übersicht

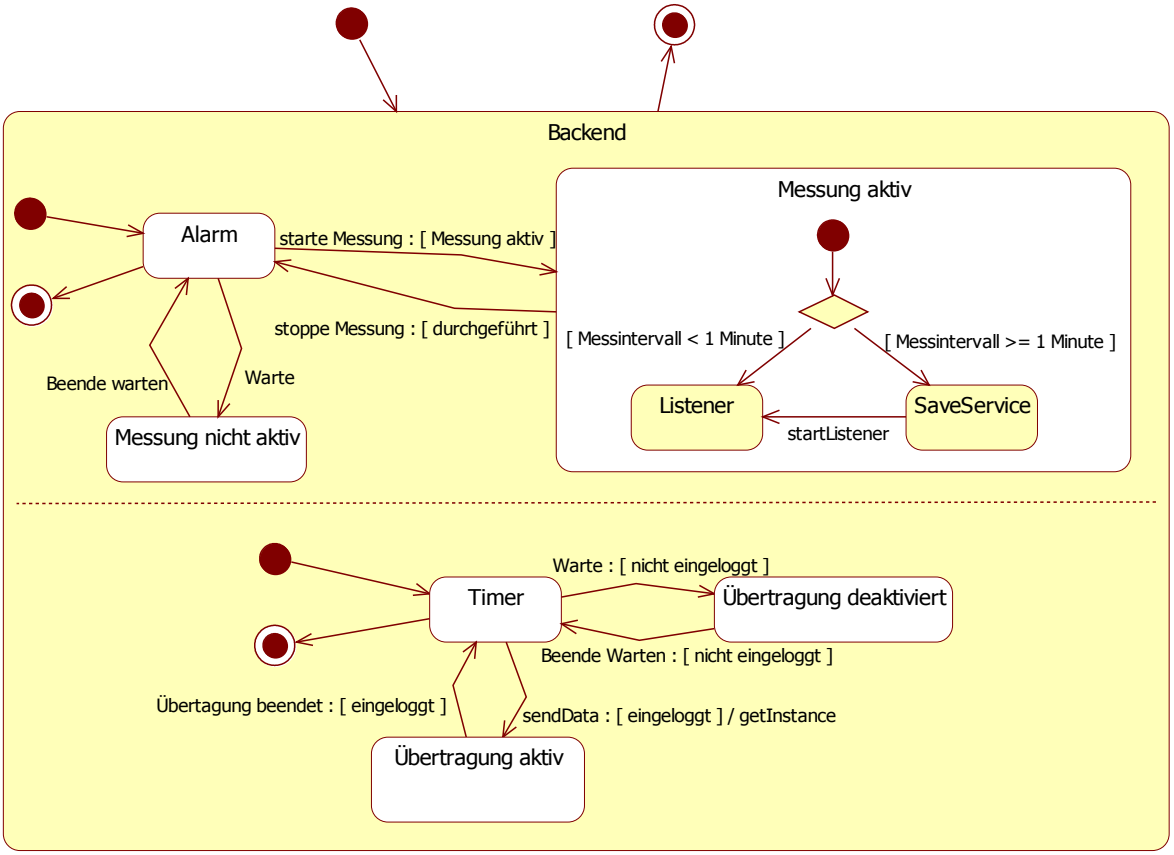

Abbildung 3.4: StateChart zur Komponente Backend $\langle C12 \rangle$

Es gibt zwei Timer („Alarm“ und „Timer“), die jeweils einmal während des vom User vorgegeben Zeitintervalls eine Aktion ausführen. Diese beiden Timer laufen dabei parallel, was durch die gestrichelte Linie gekennzeichnet wird. „Alarm“ initiiert eine Messung und deren anschließende Speicherung mithilfe der Persistenzschicht. Dabei wird, wenn das Messintervall kleiner als eine Minute ist direkt der Listener gestartet, damit GPS aktiviert bleibt, um eine dauerhaftes Positionsupdate zu erhalten ohne Verzögerungen. Ansonsten wird der Umweg über den SaveService genommen, um den GPS-sensor zu aktivieren und diesem Zeit zu geben, die Position bestimmen zu können. „Timer“ startet die Übertragung an den Server durch Übergabe der Datensätze

an die Client-Subkomponente. Dabei wird im Zustand “Übertragung aktiv“ das nachfolgende StateChart zum Client (Abbildung 3.5) gestartet. Zum Übertragen muss der Benutzer sich einloggen, um dem Server bekannt zu sein.

Client

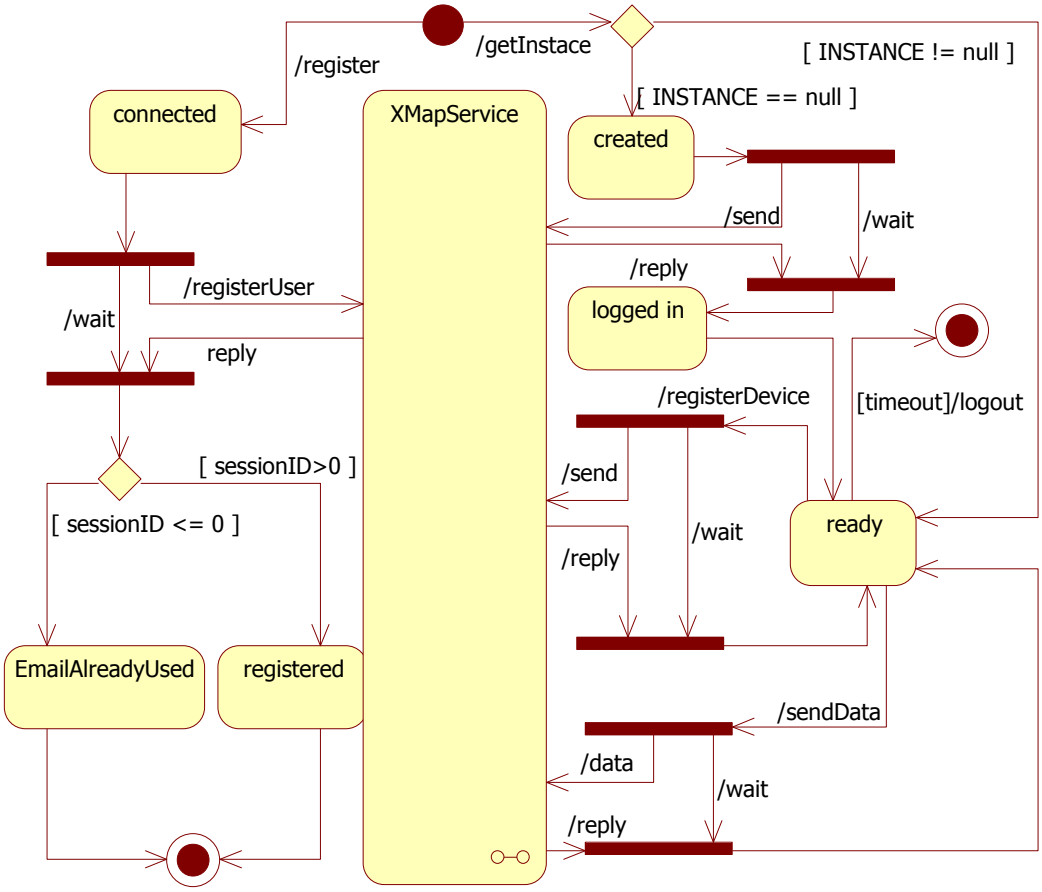

Abbildung 3.5: StateChart zur Kommunikation

Der Client kann entweder zur Registrierung oder zur regulären Kommunikation instanziiert werden. Nach der Registrierung wird die Verbindung sofort wieder beendet. Beim regulären Aufruf bleibt der Client aktiv, bis eine Timeoutzeit erreicht ist oder ein manuelles Logout ausgeführt wird. Nach der erfolgreichen Anmeldung kann das Gerät registriert oder Daten übertragen werden. Treten Fehler auf, werden diese zur Behandlung an die aufrufenden Entitäten weitergeleitet, welche die Fehler dann in einem entsprechenden Fehlerlog vermerken.

3.3.4 Komponente $\langle C13 \rangle$: DatabaseClient

Dieser Teilabschnitt beschreibt die Datenbank der Applikation, die für die Verwaltung der lokalen Daten vorhanden ist. Dabei wird diese Komponente mehrfach wiederverwendet, da die Applikation in periodischen Zeitabständen Daten abspeichert, welche auch mehrfach für die Visualisierung und Übertragung an den WebService ausgelesen werden. Dabei muss jede Methode und Klasse, die die Datenbank geöffnet hat, diese auch explizit wieder schließen.

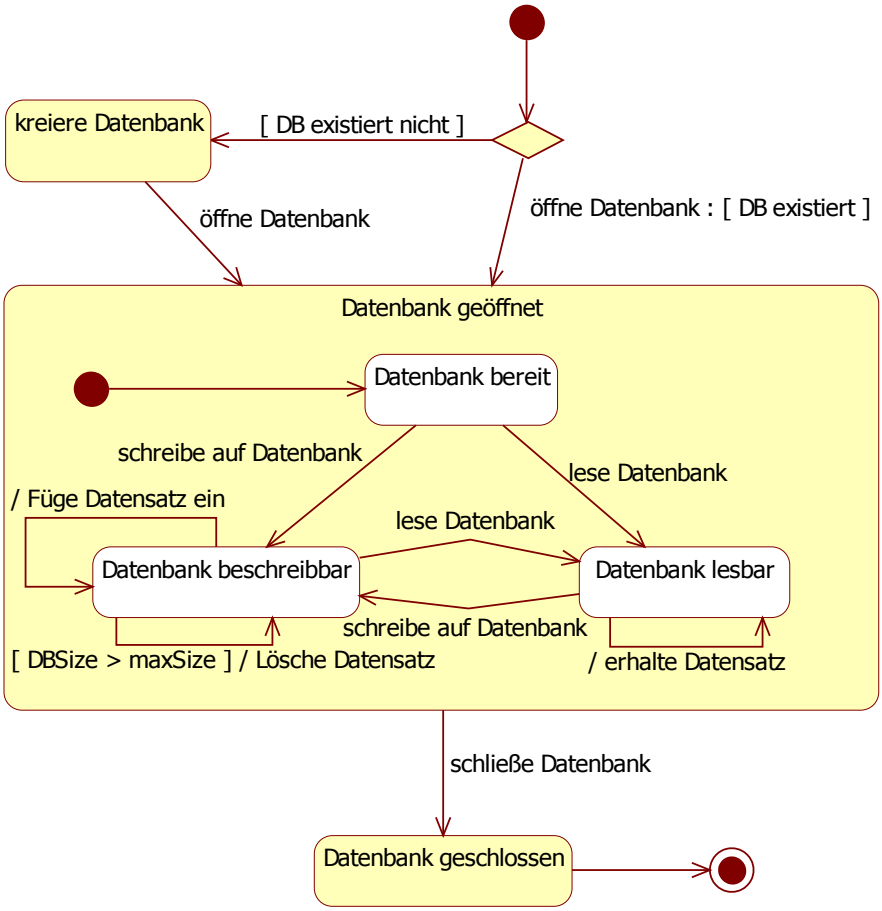

Abbildung 3.6: StateChart zur Komponente DatabaseClient $\langle C13 \rangle$

Die Komponente kreiert die Datenbank, falls diese noch nicht existiert, und öffnet sie. Wenn die Datenbank bereits existiert, wird stattdessen jene direkt geöffnet.
Während die Datenbank geöffnet ist, kann man in ihr schreiben und lesen. Schreibvorgänge sind das Einfügen und Löschen von Datensätze. Dabei werden Daten nur gelöscht, wenn die Datenbankgröße die maximal eingestellte Speichergröße überschreitet. Für das Lesen der Datenbank werden die jeweils benötigten Datensätze ausgelesen. Solange die Datenbank geöffnet ist, kann sie jederzeit geschlossen werden.
Nachdem die Datenbank einmal geöffnet wurde, muss sie zuerst wieder geschlossen werden, bevor sie beendet werden kann.

3.3.5 Server

Die Komponenten des Servers sind zustandslos, abgesehen vielleicht von „Busy“-„Idle“-Zuständen. Daher werden sie an dieser Stelle nicht weiter beschrieben.

4 Verteilungsentwurf

Deses Kapitel beschreibt die Verteilung der Komponenten mit einem Verteilungsdiagramm. Da das Produkt eine Client-Server-Architektur besitzt, wird die Verteilung auch dementsprechend sein.

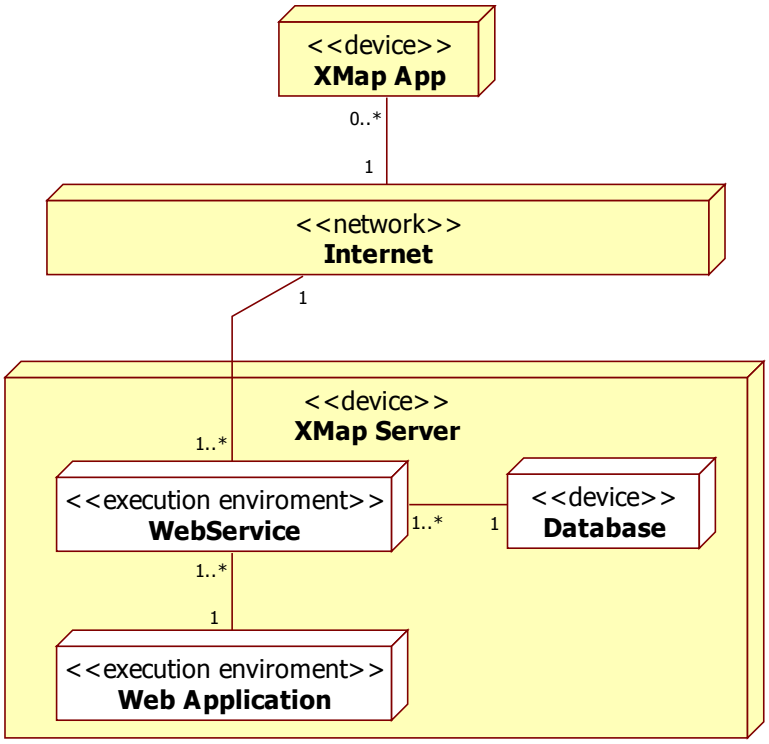

Abbildung 4.1: Verteilungsdiagramm

Die Abbildung 4.1 zeigt ein einfaches Verteilungsdiagramm. Auf dem X-Map Server laufen mehrere multithreaded WebServices, eine Web Applikation und eine Datenbank. Dabei ist die Datenbank ein Hardware Element (device) und die WebServices und die Web Applikation Laufzeitumgebungen (execution enviroment). Sowohl Datenbank als auch Web Applikation kommunizieren mit den WebServices.
Die WebServices des X-Map Servers sind mittels der physikalischen Verbindung, dem Internet, mit beliebig viele X-Map Apps verbunden.
Eine X-Map App beschreibt dabei ein Mobilgerät und somit auch einen Client.

5 Implementierungsentwurf

Die folgenden Unterkapitel detaillieren für jede einzelne der in Kapitel 3 beschriebenen Komponenten den gewählten Weg der Implementierung.
Zunächst zur Erinnerung ein weiteres Mal das Komponentendiagramm, welches in Originalgröße in Abbildung 3.1 abgebildet ist.

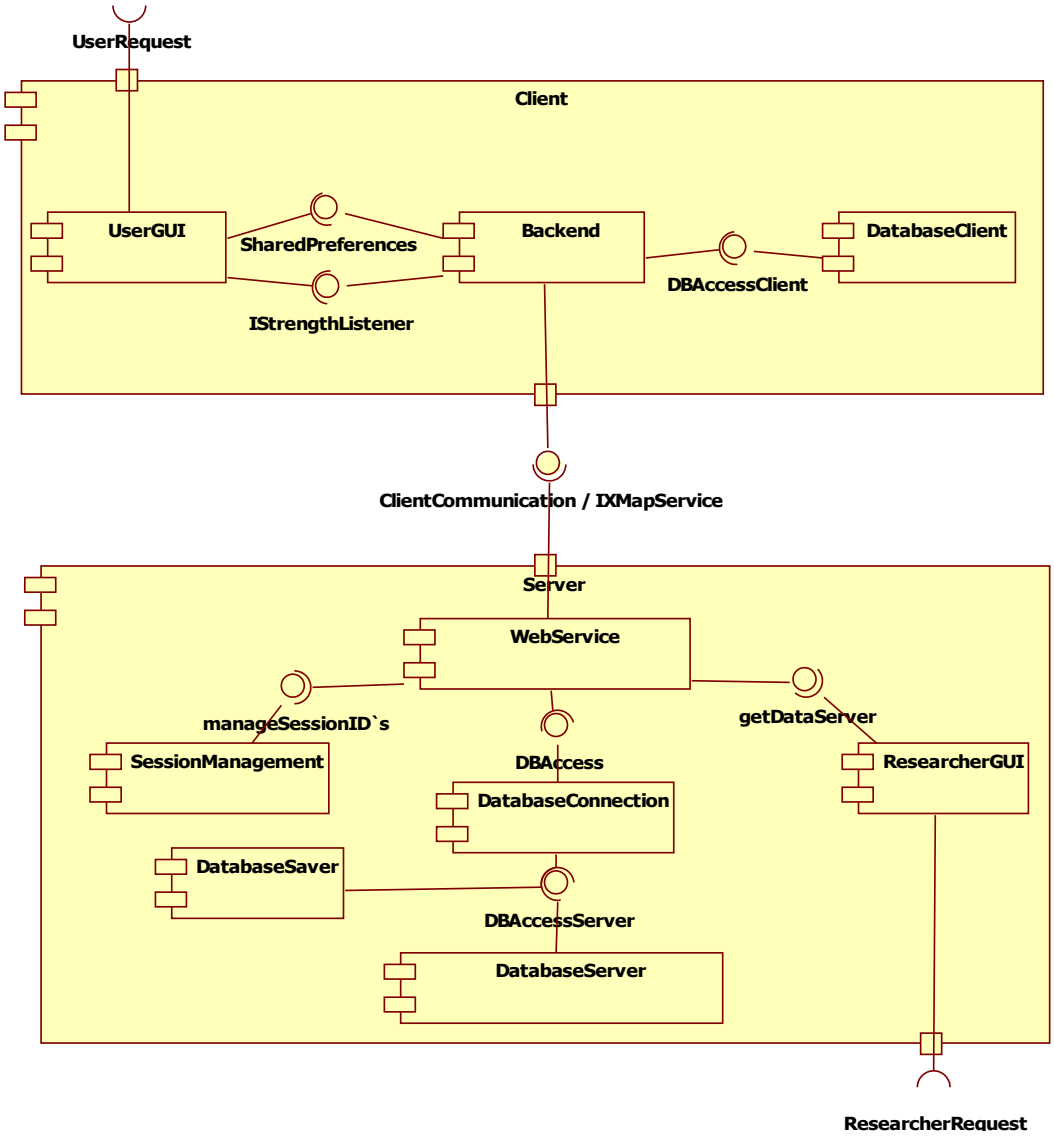

Abbildung 5.1: Komponentendiagramm (im Original im Kapitel 3.1)

Hinweis zu den Abbildungen des folgenden Kapitels:
Durch die notwendige Größe der Abbildungen und der für dieses Kapitel vorgegebenen Struktur ist es oftmals nicht möglich, die Abbildungen passend in den Text einzubetten. Sie werden automatisch auf eine Folgeseite ausgelagert. Um den Bezug zu erhalten wurden an den entsprechenden Stellen Verweise samt interner Verlinkungen in das Dokument eingefügt.

5.1 Implementierung von Komponente $\langle C10 \rangle$: Client:

Diese Komponente wird die gesamte Architektur der Applikation beschreiben. Da die Komponenten $\langle C11 \rangle$, $\langle C12 \rangle$ und $\langle C13 \rangle$ Subkomponten sind und dort jeweils detailliert erläutert werden, wird hier nur ein kurzer Überblick über die Architektur angegeben.

5.1.1 Paketdiagramm

Im folgenden Paketdiagramm in Abbildung 5.2 wird die Architektur der Komponente $\langle C10 \rangle$ mittels Pakete beschrieben.

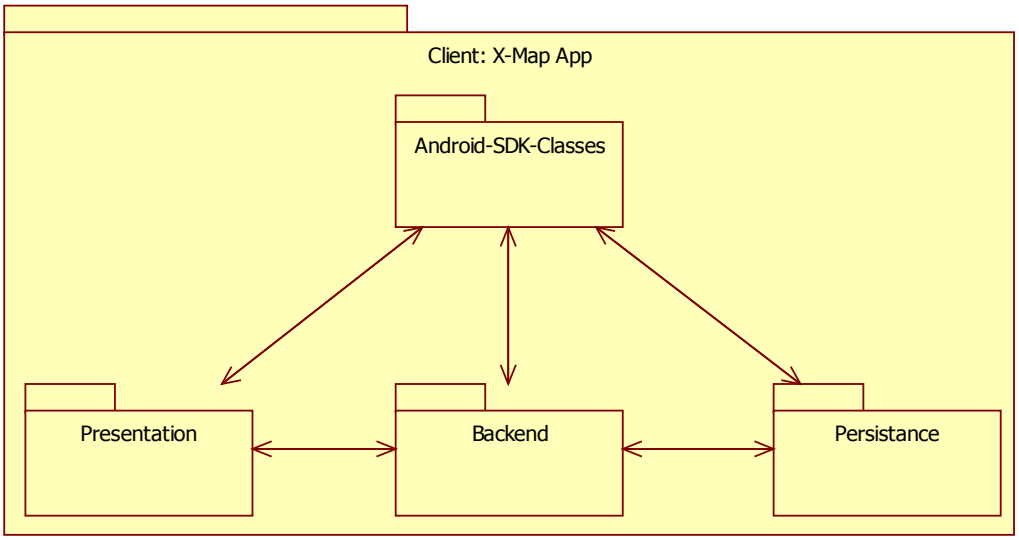

Abbildung 5.2: Paketdiagramm für Komponente $\langle C10 \rangle$

5.1.2 Erläuterung

Die Komponente $\langle C10 \rangle$ beschreibt die gesamte Android-Applikation und ihren Aufbau. Dabei wird die 3-Schichten-Architektur eingehalten. Die drei Schichten entsprechen den folgenden drei Komponenten und werden dort jeweils auch weiter spezifiziert. Das Paket “Presentation“

wird Abschnitt 5.2 beschrieben. Der Großteil des “Backend“-Pakets entspricht der Komponente in Abschnitt 5.3 und der Teil des “Backend“-Pakets, der die Datenbank betrifft, und das “Persistance“-Paket werden in Abschnitt 5.4 genauer erläutert.
Das letzte Paket “Android-SDK-Classes“ enthält Klassen, die das Android-Betriebssystem benötigt und die durch das Android-SDK automatisch generiert werden. Da die Entwickler auf diese Klassen keinen Einfluss haben, werden diese auch nicht weiter betrachtet. Dieses Paket hat Zugriff auf alle Klassen innerhalb der Applikation, wirkt sich aber nicht auf die 3-Schichten Architektur aus, da die Klassen im Paket “Android-SDK-Classes“ nur die vom Entwickler erstellten Klassen und das Android-Betriebssystem miteinander verbindet. So werden zum Beispiel, die von der Android-Applikation verwendet Ressourcen in einer extra, vom Android-SDK generierten Klasse erzeugt.

Aus diesem Grund sind in diesem Diagramm auch keine Klassen enthalten, sondern nur die dazugehörigen Pakete.

5.2 Implementierung von Komponente $\langle C11 \rangle$: UserGUI:

Diese Komponente implementiert die grafische Benutzeroberfläche der Applikation „X-Map App“.

Die MainActivitiy ist der Startbildschirm beim Öffnen der X-Map App.
Im Loginfenster (LoginActivity) kann sich der Nutzer mit Email und Passwort anmelden oder neu registrieren (CreateAccountActivity). Für die Registrierung wird eine Instanz vom Client erzeugt, über SOAP eine Datenübertragung initiiert und eine Verbindung mit dem Server hergestellt. Verwendet wird dabei das Interface $\langle I30 \rangle$.

Im Settingsmenu (SettingsActivity) werden mit Hilfe des Android Preference Manager Nutzerdefinierte Einstellungen verwaltet. Ausgewählte Optionen durch den Nutzer, werden in einer Datei gespeichert, beispielsweise die zu messenden Parameter oder das Synchronisationsintervall mit dem WebService. Wenn der Nutzer die Einstellungen wieder ändert, führt das System mit Hilfe von $\langle I40 \rangle$ ein Update im korrespondierenden Wert in der Datei durch.

Als Visualisierungsoptionen für die Empfangsqualität stehen dem Nutzer GoogleMaps (MapsActivity), ein Graph (XYChartActivity) sowie eine Tabelle (DBTableActivity) zur Verfügung. Sie sind Listener von der Singleton Backend Klasse StrengthListener. Dafür implementieren sie das Interface $\langle I20 \rangle$, um auf Änderungen der Signalstärke reagieren zu können und als Activity im Vordergrund neue Werte „on the fly“ anzuzeigen.

Wählt der Benutzer eine bestimmte Visualisierungsoption aus, wird das gewünschte Visualisie-

rungsobjekt von der GUI erstellt und dem Nutzer mit den entsprechenden Messdaten angezeigt. Weiterhin kann der Nutzer die maximale Anzahl an angezeigten Messwerten festlegen. Dazu interagiert die GUI mit der Datenbank über die Schnittstelle $\langle I10 \rangle$ und liest mit Hilfe eines Cursors der Datenbank die gewünschte Anzahl der Datensätze aus der Datenbank aus, welche dem Nutzer angezeigt werden.

Die Komponente ist in Java geschrieben, welches durch das Android SDK erweitert wird. Für den xy-Graph der XYChartActvivity wird die Bibliothek AndroidPlot importiert. Diese stellt einen einfach zu implementierenden xy-Graph dar, der dynamisch erweitert werden kann. Dabei wird in der Klasse MultitouchPlot die in AndroidPlot vorhandenen Klassen soweit erweitert, dass der Graph mittels Fingerbewegungen reagiert und somit zoombar, skalierbar und verschiebbar ist.

5.2.1 Paket-/Klassendiagramm

Abbildung 5.3 und Abbildung 5.4

5.2.2 Erläuterung

Im Folgenden werden die Klassen in den Klassendiagrammen in Abbildung 5.3 und Abbildung 5.4 genauer beschrieben:
Um die Übersicht zu gewähren, wurde die Präsentations-Schicht in zwei Klassendiagramme eingeteilt: Das Klassendiagramm in Abbildung 5.3 stellt die GUI bzgl. Startbildschirm sowie Login- und Registrierungsfunktionalität dar. Darüber hinaus die Einstellungen für Messungen der Empfangsqualität.
Das Klassendiagramm in Abbildung 5.4 stellt die GUI bzgl. Visualisierungsmöglichkeiten (GoogleMap, Tabelle, Graph) der Messungen für die Empfangsqualität dar.
Aus Platzgründen werden Klassenattribute beider Diagramme nicht angezeigt.

MainActivity$\langle CL110 \rangle$

Aufgabe

Visualisierung der Start-Seite

Attribute

locationManager: Objekt der Klasse LocationManager zur GPS Aktualisierung.

res: Objekt der Klasse Resources zur Darstellung der Objekte des Layouts.

Operationen

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Wird zum initialisieren sämtlicher Elemente

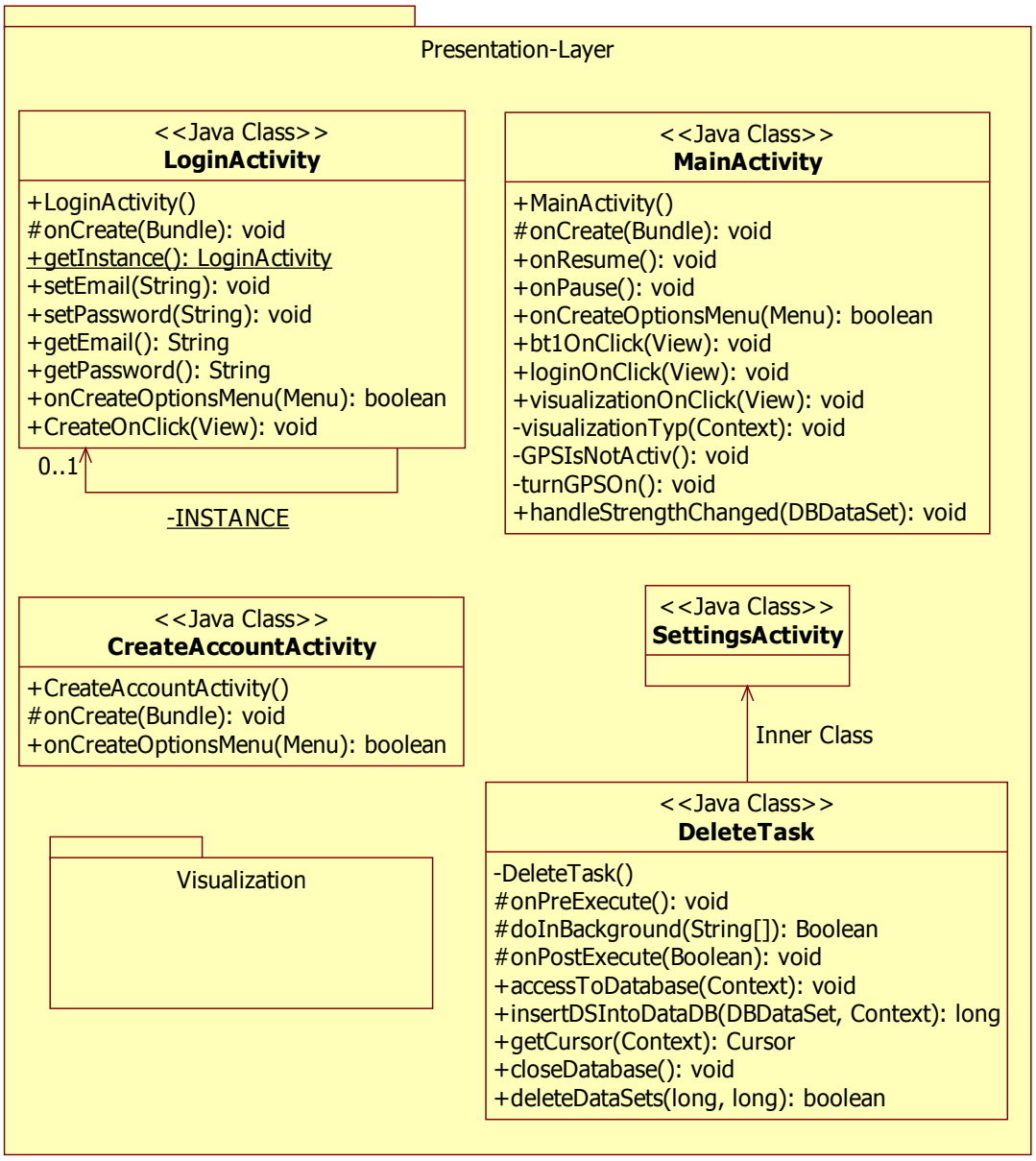

Abbildung 5.3: Klassendiagramm für Komponente $\langle C11 \rangle$

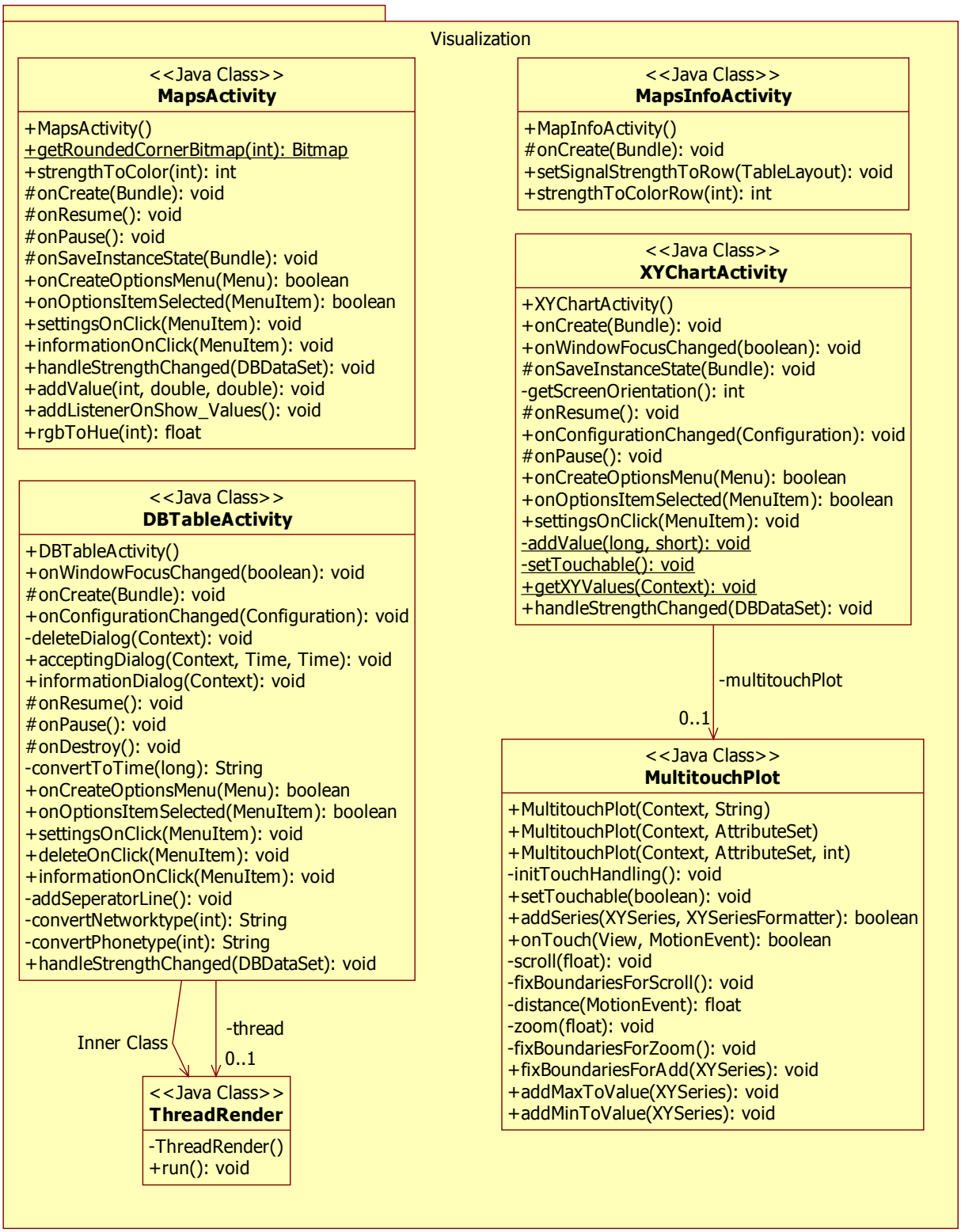

Abbildung 5.4: Klassendiagramm für Komponente $\langle C11 \rangle$

des Layouts verwendet.
onResume(): Eine von Android bereitgestellte Operation, die automatisch nach Unterbrechung der Aktivität (onPause) aufgerufen wird.
onPause(): Eine von Android bereitgestellte Operation. Wird bei Unterbrechung der Aktivität aufgerufen.
onCreateOptionsMenu(Menu): Eine von Android bereitgestellte Operation, die den Inhalt des Auswahlmenüs initialisiert.
bt1OnClick(View): Button zum Start der SettingsActivity.
loginOnClick(View): Button zum Start der LoginActivity.
visualizationOnClick(View): Button zum Start eines Auswahlmenüs bzgl. Visualisierung.
visualizationTyp(Context): Auswahlmenü (AlertDialog, von Android bereitgestellt), in dem der Benutzer seine gewünschte Visualisierung wählen kann. Startet entsprechend gewünschte Aktivität (DBTableActivity $\langle CL119\rangle$, XYChartActivity $\langle CL117\rangle$ oder MapsActivity $\langle CL115\rangle$).
GPSIsNotActiv(): Wird beim Start der Anwendung aufgerufen, falls GPS nicht aktiv ist. Gibt Hinweis an den Nutzer aus, inkl. Möglichkeit zum Aktivieren von GPS (turnGPSon()).
turnGPSOn(): Operation zum Aktivieren von GPS in Android.
handleStrengthChanged(DBDataSet): Operation zur Ausgabe der aktuellen Messdaten.

Kommunikationspartner

Die Klasse SignalStrengthListener $\langle CL1210\rangle$, welche bei veränderter Signalstärke Daten misst.

LoginActivity$\langle CL111\rangle$

Aufgabe

Visualisierung der Login-Seite

Attribute

email: Textfeld zur Eingabe der E-Mail-Adresse.
password: Textfeld zur Eingabe des Passworts.
loginClicker: Button zum Login.

Operationen

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Wird zum initialisieren sämtlicher Elemente des Layouts verwendet.
getInstance(): Get-Methode für Instanz.

setEmail(String): Set-Methode für E-Mail-Adresse.
getEmail(String): Get-Methode für E-Mail-Adresse.
setPassword(): Set-Methode für Passwort.
getPassword(): Get-Methode für Passwort.
onCreateOptionsMenu(Menu): Eine von Android bereitgestellte Operation, die den Inhalt des Auswahlmenüs initialisiert.
CreateOnClick(View): Button zum Start der CreateAccountActivity.

Kommunikationspartner

Die Klasse Client $\langle CL124 \rangle$, welche die Verbindung mit dem WebService erstellt.

CreateAccountActivity$\langle CL112 \rangle$

Aufgabe

Visualisierung der CreateAccount-Seite

Attribute

email: Textfeld zur Eingabe der E-Mail-Adresse.
password: Textfeld zur Eingabe des Passworts.
loginClicker: Button zur Erstellung eines Accounts.

Operationen

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Wird zum initialisieren sämtlicher Elemente des Layouts verwendet.
onCreateOptionsMenu(Menu): Eine von Android bereitgestellte Operation, die den Inhalt des Auswahlmenüs initialisiert.

Kommunikationspartner

Die Klasse Client $\langle CL124 \rangle$, welche die Verbindung mit dem WebService erstellt.

SettingsActivity$\langle CL113 \rangle$

Aufgabe

Visualisierung des Einstellungsmenüs

Operationen

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Wird zum initialisieren sämtlicher Elemente des Layouts verwendet.
onSharedPreferenceChanged(SharedPreference, String): Methode, die automatisch aufgerufen wird, wenn sich ein Einstellungswert ändert.

Kommunikationspartner

Die Klasse DeleteTask $\langle CL114 \rangle$, da DeleteTask eine innere Klasse ist.

DeleteTask⟨$CL114$⟩

Aufgabe

Dialog zum Löschen der Datenbank, falls gewählte max. Datenbankgröße größer als aktuelle Datenbankgröße

Attribute

dialog: Objekt der Klasse ProgressDialog zum Erstellen eines Dialogs mit dem Benutzer.
db: Dieses Attribut stellt ein Datenbank-Objekt dar, um darauf die Operationen auszuführen.

Operationen

onPreExecute(): Öffnet einen Dialog mit dem Benutzer, bevor gelöscht wird.
doInBackground(String[]): Methode zur Erstellung eines Hintergrund Threads zum Löschen von Daten.
onPostExecute(Boolean): Methode zum Schliessen des Dialogs.
accessToDatabase(Context): Methode zum Öffnen der Datenbank.
insertDBIntoDataDB(DBDataSet, Context): Keine eigene Funktion, muss wegen DBAccess vorhanden sein.
getCursor(Context): Keine eigene Funktion, muss wegen DBAccess vorhanden sein.
closeDatabase(): Methode zum Schliessen der Datenbank.
deleteDataSets(long, long): Keine eigene Funktion, muss wegen DBAccess vorhanden sein.

Kommunikationspartner

Die Klasse SettingsActivity ⟨$CL113$⟩, da DeleteTask eine innere Klasse ist.

MapsActivity⟨$CL115$⟩

Aufgabe

Visualisierung der Signalstärke orts- und qualitätsbezogen auf einer GoogleMap.

Attribute

gMap: Objekt einer GoogleMap.
db: Eine Instanz der Klasse DBForPresentation.
markers: Eine ArrayList, in der Marker (Overlay-Objekte auf der GoogleMap) gesammelt werden.
hsv: Array des Datentypen float, der für die Konvertierung von RGB zu HSV genutzt wird, um den ersten (hue-)Wert für Färbung des Markers zu extrahieren.
show_Value: Objekt einer Checkbox, mit welcher die Sichtbarkeit von Markern gesteuert wird.
sharedPref: Eine SharedPreferences Instanz, welche auf eine default Datei zeigt, in der Einstellungen gespeichert werden.

yourLocation: Objekt der Klasse CameraUpdate, in welcher Längen- und Breitgradkoordinaten der letzen Messung gespeichert werden.

Operationen

getRoundedCornerBitmap(int): Diese Operation liefert ein gerundetes Rechteck eines Bitmap-Objektes zurück. Dieses Objekt soll colorierte Kreise auf der GoogleMap darstellen.

strengthToColor(int): Diese Operation konvertiert die Signalstärke in einen Hexadezimal-Wert. Rot steht für eine schlechte Empfangsqualität und grün für eine sehr gute. Grau impliziert eine fehlende Empfangsqualität.

onCreate(Bundle): Eine von Android bereitgestellte Operation, um die Aktivität zu initialisieren und das Layout zu setzen. Beispielsweise wird hier der Listener initialisiert, der darauf achtet, ob die Checkbox an ist oder nicht und initial eine ausgeschaltete Checkbox festgelegt. Desweiteren wird ein Cursor initialisiert, welcher auf der Datenbank arbeitet und beispielsweise eine ausgewählte Anzahl an Messungen ausliest.

onPause(): Eine von Android bereitgestellte Operation, welche das Verhalten der Aktivität festlegt, wenn der Nutzer die Aktivität verlässt. Hat der Nutzer keine allzeit aktive Echtzeit-Messung der Empfangsqualität festgelegt, lauscht die Aktivität als Listener der Singleton Klasse SignalStrengthListener nicht mehr, ob sich die Signalstärke ändert.

onResume(): Eine von Android bereitgestellte Operation, welche das Verhalten der Aktivität festlegt, wenn der Nutzer die Aktivität verlassen hat und wieder öffnet. Hat der Nutzer keine allzeit aktive Echtzeit-Messung der Empfangsqualität festgelegt, lauscht die Aktivität als Listener der Singleton Klasse wieder, ob sich die Signalstärke ändert.

onSaveInstanceState(Bundle): Eine von Android bereitgestellte Operation, welche den Zustand der Aktivität in einem Bundle speichert, bevor sie beispielsweise verlassen wird. Gespeichert wird die aktuelle Position, Zoom sowie Sichtbarkeit der Marker. Genutzt wird die Methode, um bei Drehung des Bildschirms alle Änderungen seit ursprünglicher Initialisierung durch die onCreate-Methode beizubehalten.

onCreateOptionsMenu(Menu): Eine von Android bereitgestellte Operation, welche der Aktivität eine Menu hinzufügt.

onOptionsItemSelected(MenuItem): Diese Operation fügt Optionen zum Menu hinzu und reagiert auf die ausgewählte Option, indem der zugehörige Bildschirm geöffnet wird. Der Nutzer kann zu den Einstellungen gelangen, zu einer Information und weiterhin verschiedene Kartentypen, beispielsweise Satellit, auswählen.

settingsOnClick(MenuItem): Diese Operation öffnet den Settings-Bildschirm

informationOnClick(MenuItem item): Diese Operation öffnet einen Informations-Bildschirm mit einer Legende, in der Signalstärke einer entsprechenden Farbe und Bedeutung zugewiesen wird.

handleStrengthChanged(DBDataSet): Diese Operation reagiert auf einen neuen Datensatz, falls sich die Signalstärke geändert hat.

addValue(int, double, double): Diese Operation implementiert die Reaktion, falls sich die Signalstärke geändert hat. Marker in Kreisform werden entsprechend der Signalstärke gefärbt und der GoogleMap ortsbezogen auf die Messung hinzugefügt. Marker in Pinform werden parallel gefärbt und in der ArrayList markers gespeichert, um später deren Sichtbarkeit über eine Checkbox kontrollieren zu können.

addListenerOnShow_Values(): Diese Operation lauscht, ob die Checkbox angeklickt ist oder nicht. Ist sie angeklickt, werden Marker in Pinform auf den Markern in Kreisform platziert. Ist sie nicht angeklickt, sind die Marker in Pinform nicht sichtbar.

rgbToHue(int): Diese Operation konvertiert einen RGB-Wert zu einem HSV-Wert und extrahiert die erste Komponente des HSV-Wertes, den hue-Wert. Dieser wird für die Färbung von Markern in Pinform verwendet.

Kommunikationspartner

Die Klasse DBDataSet $\langle CL131 \rangle$, die ein einzelnes Datenbanktupel der Datenbank entsprechen kann, damit dieser Datensatz weiterverwendet werden kann.

Die Klasse DBForPresentation $\langle CL133 \rangle$, welche Daten aus der Persistenzschicht in die Presentationsschicht überträgt.

Das Interface IStrengthListener $\langle CL1211 \rangle$, dessen Methode handleStrengthChanged aufgerufen wird, wenn sich die Signalstärke ändert.

Die Klasse SettingsHelper $\langle CL121 \rangle$, in welcher die Einstellung bezüglich des Kartentypes gespeichert wird sowie die Anzahl der angezeigten Messwerte.

Die Klasse SignalStrengthListener $\langle CL1210 \rangle$, von der MapsActivity ein Listener ist und entsprechend gestartet bzw. gestoppt wird, wenn die Aktivität verlassen bzw. wieder geöffnet wird.

MapInfoActivity$\langle CL116 \rangle$

Aufgabe

Darstellung einer Legende zur MapActivity, in der Signalstärke einer entsprechenden Farbe und Bedeutung zugewiesen wird.

Attribute

tbl: Das Tabellenlayout. Die Tabelle besteht aus einer Spalte.

tr: Die aktuelle Tabellenreihe, welche gefüllt wird.

tv: Das aktuelle Textfeld in der Tabellenreihe, welches mit Text gefüllt wird.

Operationen

onCreate(Bundle savedInstanceState): Eine von Android bereitgestellte Operation, um die Aktivität zu initialisieren und das Layout, eine Tabelle mit einer Spalte, zu setzen. Desweiteren wird die Tabelle mit Werten der Signalstärke gefüllt.

setSignalStrengthToRow(TableLayout l): Diese Operation implementiert die Befüllung von der Tabelle mit Werten der Signalstärke von -51 bis -113 in vierer-Schritten. Mit

entsprechender Farbe zu einer Signalstärke wird der Hintergrund der Zeile gefärbt. Grün bedeutet eine sehr gute Empfangsqualität, rot eine sehr schlechte. Eine Signalstärke von 1 stellt eine fehlende Empfangsqualität dar, die Zeile wird grau eingefärbt. Weiterhin wird schriftlich hinter drei Werten der Signalstärke festgehalten, welcher Wert sehr gut ist, welcher Wert sehr schlecht ist oder bei welchem Wert keine Empfangsqualität besteht.

strengthToColorRow(int strength): Diese Operation konvertiert die Signalstärke in einen Hexadezimal-Wert. Rot steht für eine schlechte Empfangsqualität und grün für eine sehr gute. Grau impliziert eine fehlende Empfangsqualität.

XYChartActivity⟨$CL117$⟩

Aufgabe

Visualisierung der Signalstärke zeitabhängig in einem xy-Graphen.

Attribute

multitouchPlot: Eine Instanz von MultitouchPlot ⟨$CL118$⟩, die den Graphen darstellt.

series: Eine von der Bibliothek AndroidPlot bereitgestellte, dynamische SimpleXYSeries, die alle auf den Graphen dargestellten Koordinaten enthält.

TEN_MINUTES_IN_MILLIS: Ein finaler Integer, der den Wert von 10 Minuten in Millisekunden enthält.

fillPaint: Ein Paint-Objekt, dass den Farbverlauf, von grün nach rot, unterhalb der Kanten der xy-Koordinaten bildet.

sharedPref: Eine SharedPreferences Instanz, welche auf eine default Datei zeigt, in der Einstellungen gespeichert werden.

Operationen

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Wird zum initialisieren sämtlicher Elemente des Layouts verwendet.

onWindowFocusChanged(boolean): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn sich Layoutelemente in ihrer Größe ändern.

onSaveInstanceState(Bundle outState): Eine von Android bereitgestellte Operation, welche den Zustand der Activity in einem Bundle speichert, bevor sie beispielsweise verlassen wird. Gespeichert werden die derzeitigen Positionen des Graphen auf dem Bildschirm. Genutzt wird die Methode, um bei Drehung des Bildschirms alle Änderungen seit ursprünglicher Initialisierung durch die onCreate-Methode beizubehalten.

getScreenOrientation: Gibt die aktuelle Ausrichtung des Bildschirmes des Gerätes zurück.

onResume(): Eine von Android bereitgestellte Operation, die automatisch nach Unterbrechung der Aktivität (onPause) aufgerufen wird.

onConfigurationChanged(Configuration): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn sich Konfigurationen von Android ändern. Hier wird es verwendet, wenn der Bildschirm gedreht wird.
onPause(): Eine von Android bereitgestellte Operation. Wird bei Unterbrechung der Aktivität aufgerufen.
onCreateOptionsMenu(Menu): Eine von Android bereitgestellte Operation, welche der Aktivität eine Menu hinzufügt.
onOptionsItemSelected(MenuItem): Diese Operation fügt Optionen zum Menu hinzu und reagiert auf die ausgewählte Option, indem der zugehörige Bildschirm geöffnet wird.
settingsOnClick(MenuItem item): Diese Operation öffnet den Settings-Bildschirm
addValue(long, short): Fügt series einen neuen Wert hinzu. Der Graph erhält eine neue Koordinate.
setTouchable(): Setzt die Funktion, ob der xy-Graph auf Berührungen reagieren soll oder nicht.
getXYValues(Context): Liest die Datenbank aus und fügt alle in der Datenbank enthaltenden Werte in die series ein.
handleStrengthChanged(DBDataSet): Diese Operation reagiert auf einen neuen Datensatz, falls sich die Signalstärke geändert hat.

Kommunikationspartner

Die Klasse MultitouchPlot $\langle CL118 \rangle$, da auf dem MultitouchPlot der xy-Graph aufgetragen wird.

MultitouchPlot$\langle CL118 \rangle$

Aufgabe

Fügt der im AndroidPlot enthaltenden Klasse XYPlot Touchfunktionen, wie zoomen, skalieren und verschieben, zu.

Attribute

NONE: Ein finaler Integer, der verwendet wird, wenn kein Finger den Bildschirm berührt.
ONE_FINGER_DRAG: Ein finaler Integer, der verwendet wird, wenn ein Finger den Bildschirm berührt.
TWO_FINGER_DRAG: Ein finaler Integer, der verwendet wird, wenn zwei Finger den Bildschirm berührt.
minXSeriesValue: Ein Number-Objekt, der den kleinsten x-Wert des xy-Graphen enthält.
maxXSeriesValue: Ein Number-Objekt, der den größten x-Wert des xy-Graphen enthält.
minYSeriesValue: Ein Number-Objekt, der den kleinsten y-Wert des xy-Graphen enthält.

maxYSeriesValue: Ein Number-Objekt, der den größten y-Wert des xy-Graphen enthält.
firstFinger: Ein von Android bereitgestelltes PointF-Objekt, dass die Position des ersten Fingers auf dem Bildschirm enthält.
lastScrolling: Ein float-Wert, der die Fingerbewegung enthält, um den Graphen zu scrollen.
distBetweenFingers: Ein float-Wert, der bei zwei Fingern, die Distanz dieser beiden Fingern bestimmt.
newMinX: Ein Number-Objekt, der den neusten minimal x-Wert des xy-Graphen enthält.
newMaxX: Ein Number-Objekt, der den neusten maximal x-Wert des xy-Graphen enthält.
touchable: Ein Boolean, der anzeigt, ob der Graph auf Berührungen reagiert.

Operationen

MultitouchPlot(...): Konstruktoren, die die Konstruktoren von XYPlot, aus der Bibliothek AndroidPlot, ausführen.
initTouchHandling(): Aktiviert den Listener, der auf Fingerberührungen des Bildschirms reagiert.
setTouchable: Setzt die Funktion, ob der xy-Graph auf Berührungen reagieren soll oder nicht.
addSeries(XYSeries, XYSeriesFormatter): Fügt dem Graphen eine Serie hinzu.
onTouch(View, MotionEvent): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn sich der Bildschirm berührt wird.
scroll(float): Scrollt den Graphen.
fixBoundariesForScroll(): Passt den Ausschnitt des Graphen an die neu gescrollte Positionen an.
distance(MotionEvent): Bestimmt die Distanz einer Fingerbewegung.
zoom(float): Zoomt den Graphen.
fixBoundariesForZoom(): Passt den Ausschnitt des Graphen an den Zoom an.
fixBoundariesForAdd(XYSeries): Passt den Ausschnitt des Graphen bei einem neu eingefügten Graphen an.
addMaxToValue(XYSeries): Setzt maxXSereisValue auf den maximalen Wert der XY-Series auf, falls dieser kleiner ist.
addMinToValue(XYSeries): Setzt minXSereisValue auf den minimalen Wert der XY-Series auf, falls dieser kleiner ist.

Kommunikationspartner

Die Klasse XYChartActivity ⟨$CL117$⟩, da XYChartActivity, einen touchfähigen Graphen darstellt.

Die Klasse XYPlot, aus der Bibliothek AndroidPlot, welche vererbt wird, um die Grundoperationen des Graphens zu besitzen und zu erweitern.

DBTableActivity⟨$CL119$⟩

Aufgabe

Visualisierung der gemessenen Daten in einer Tabelle.

Attribute

pd: Eine Progressbar, die während des Ladens der Tabelle angezeigt wird.

context: Kontext der betrachteten Activity.

table: Layout der Tabelle, welche dynamisch mit neuen Messdaten erweitert wird.

mainLayout: Layout der Activitiy.

scrollView: Objekt zur Implementierung einer vertikal scrollbaren Tabelle.

linLayout: Inneres Layout der Tabelle.

tr: Die aktuelle Tabellenreihe, welche gefüllt wird.

tv: Das aktuelle Textfeld in der Tabellenreihe, welches mit Text gefüllt wird.

headRow: Die Kopfzeile der Tabelle.

hScrollView: Objekt zur Implementierung einer horizontal scrollbaren Tabelle.

headText: Array mit Textelementen der Kopfzeile.

db: Eine Instanz der Klasse DBForPresentation.

thread: Objekt der Klasse ThreadRender.

from: Zeitpunkt, von dem gelöscht werden soll.

to: Zeitpunkt, bis zu dem gelöscht werden soll.

isCreated: Boolean Flag, der wahr ist, wenn die Tabelle erstellt wurde.

vto: Objekt der Klasse der ViewTreeObserver, der Listener der Tabelle benachrichtigt, wenn sich die Tabelle ändert.

count: Zählvariable für Datensätze in der Datenbank.

sharedPref: Eine SharedPreferences Instanz, welche auf eine default Datei zeigt, in der Einstellungen gespeichert werden.

COLUMNS: Array mit IDs der benötigten Spalten zuzuweisen.

handler: Signalisiert dem Mainlayout, dass das Layout angezeigt werden kann.

Operationen

onWindowFocusChanged(boolean): Gleicht Kopfzeilenbreite der Tabelle an.

onCreate(Bundle): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn die Aktivität gestartet wird. Startet den ThreadRender ⟨$CL1110$⟩ und startet den Ladebalken.

onConfigurationChanged(Configuration): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn sich Konfigurationen von Android ändern. Hier wird es verwendet, wenn der Bildschirm gedreht wird.
deleteDialog(Context): Dialog mit dem Benutzer zum Löschen von Messdaten in einem Zeitintervall.
acceptingDialog(Context, Time, Time): Nachfrage beim Nutzer, ob die ausgewählten Messdaten wirklich gelöscht werden sollen.
informationDialog(Context): Dialog zur Information über die Tabelle, beispielsweise Anzahl der angezeigten Messdaten.
onResume(): Eine von Android bereitgestellte Operation, die automatisch nach Unterbrechung der Aktivität (onPause) aufgerufen wird.
onConfigurationChanged(Configuration): Eine von Android bereitgestellte Operation, die automatisch aufgerufen wird, wenn sich Konfigurationen von Android ändern. Hier wird es verwendet, wenn der Bildschirm gedreht wird.
onPause(): Eine von Android bereitgestellte Operation. Wird bei Unterbrechung der Aktivität aufgerufen.
onDestroy(): Eine von Android bereitgestellte Operation. Wird bei Beendung der Aktivität aufgerufen.
convertToTime(long): Diese Methode konvertiert Millisekunden seit dem 1.1.1970 in ein Datumformat DD.MM.YYYY, HH:MM:SS.
onCreateOptionsMenu(Menu menu): Eine von Android bereitgestellte Operation, welche der Activity eine Menu hinzufügt.
onOptionsItemSelected(MenuItem item): Diese Operation fügt Optionen zum Menu hinzu und reagiert auf die ausgewählte Option, indem der zugehörige Bildschirm geöffnet wird.
settingsOnClick(MenuItem item): Diese Operation öffnet den Settings-Bildschirm.
deleteOnClick(MenuItem): Diese Operation öffnet den Delete-Dialog.
informationOnClick(MenuItem): Diese Operation öffnet den Informations-Dialog.
addSeperatorLine(): Diese Operation zeichnet Linien zwischen den Zeilen der Tabelle.
convertNetworktype(int): Diese Operation übersetzt eine Konstante in den entsprechenden Netzwerktyp, zum Beispiel LTE.
convertPhonetype(int): Diese Operation übersetzt eine Konstante in den entsprechenden Telefonstandard, zum Beispiel GSM.
handleStrengthChanged(DBDataSet): Diese Operation reagiert auf einen neuen Datensatz, falls sich die Signalstärke geändert hat.

Kommunikationspartner

Die Klasse DBDataSet $\langle CL131 \rangle$, die ein einzelnes Datenbanktupel der Datenbank entsprechen kann, damit dieser Datensatz weiterverwendet werden kann.
Die Klasse DBForPresentation $\langle CL133 \rangle$, welche Daten aus der Persistenzschicht in die

Presentationsschicht überträgt.
Das Interface IStrengthListener $\langle CL1211 \rangle$, dessen Methode handleStrengthChanged aufgerufen wird, wenn sich die Signalstärke ändert.
Die Klasse SettingsHelper $\langle CL121 \rangle$, in welcher die Einstellung bezüglich der maximalen Datenbankgröße gespeichert werden.
Die Klasse SignalStrengthListener $\langle CL1210 \rangle$, von der DBTableActivity ein Listener ist und entsprechend gestartet bzw. gestoppt wird, wenn die Aktivität verlassen bzw. wieder geöffnet wird.
Die Klasse ThreadRender $\langle CL1110 \rangle$, da ThreadRender eine innere Klasse ist.

ThreadRender$\langle CL1110 \rangle$

Aufgabe

Erstellt aus den Daten aus der Datenbank eine dynamische Tabelle.

Operationen

ThreadRender(): Konstruktor der Klasse.
run(): Wird beim Start des Threads gestartet. Es wird die Tabelle aus den Daten der Datenbank erstellt.

Kommunikationspartner

Die Klasse DBTableActivity $\langle CL119 \rangle$, da ThreadRender eine innere Klasse ist.

5.3 Implementierung von Komponente $\langle C12 \rangle$: Backend:

Das Backend ist eine zentrale Kontroll- und Steuerungseinheit. Es implementiert die Funktionalität der Empfangsqualitätsmessung und die Übertragung der Daten zum Webservice. Um Messungen zu den Parametern der Empfangsqualität des Nutzers durchzuführen und in der Datenbank zu speichern, hat der Nutzer zwei Auswahlmöglichkeiten: Zum einen können periodische Messungen mit Hilfe eines Timers durchgeführt werden. Zum anderen kann eingestellt werden, dass der SignalStrengthListener in Echtzeit auf eine Veränderung der Signalstärke achtet. Die Datenübertragung zum Webservice wird in einstellbaren Intervallen über SOAP ermöglicht.
Die Komponente ist in Java geschrieben und wurde durch das Android-SDK erweitert. Als Bibliothek wird kSOAP 2 verwendet, um die Kommunikation mit dem WebService mittels dem SOAP-Protokoll zu gewährleisten. Dabei wird diese Bibliothek eingebunden, da von Android aus SOAP nicht integriert ist.

5.3.1 Paket-/Klassendiagramm

Abbildung 5.5 und Abbildung 5.6

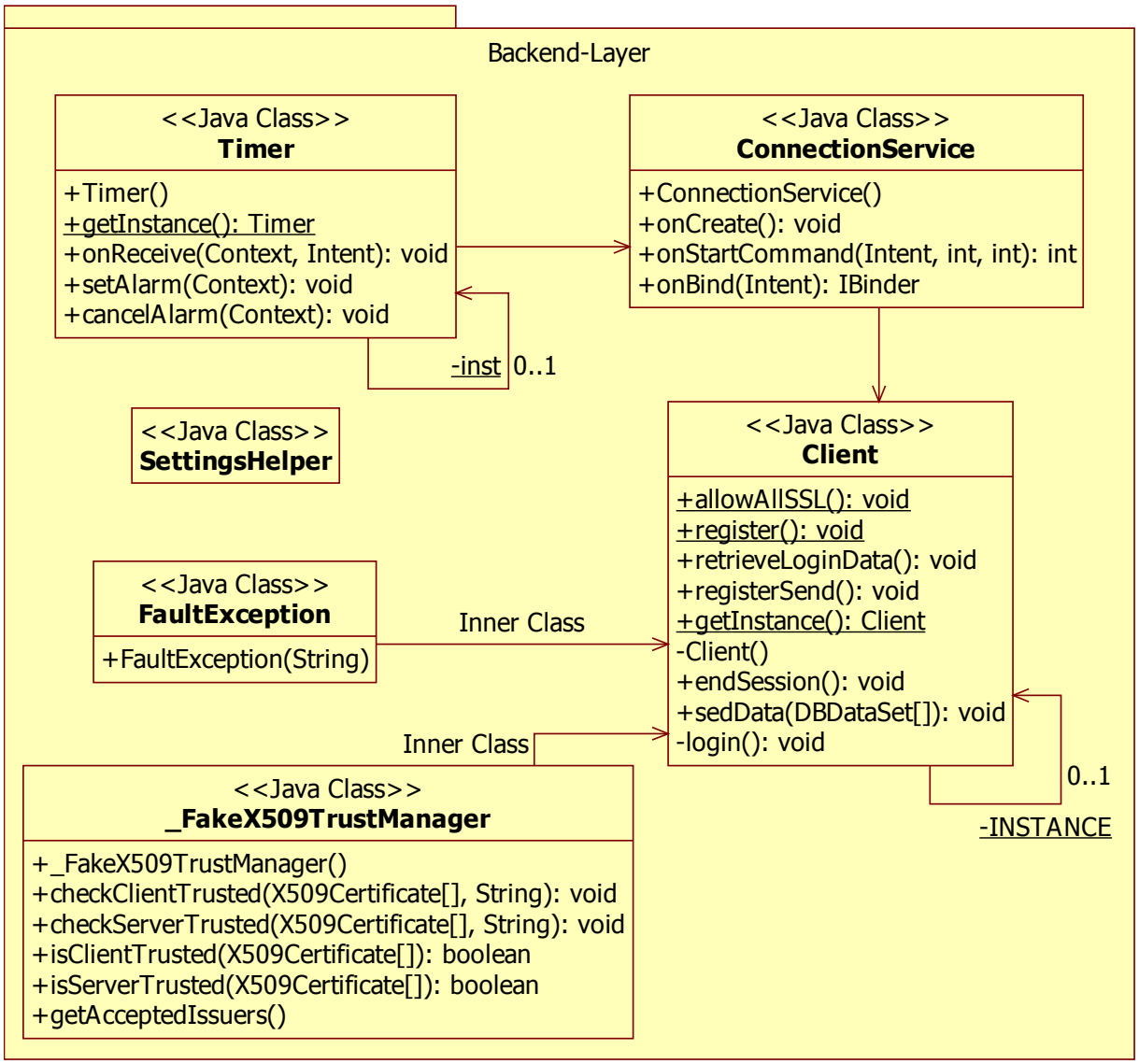

Abbildung 5.5: Klassendiagramm für Komponente $\langle C12 \rangle$

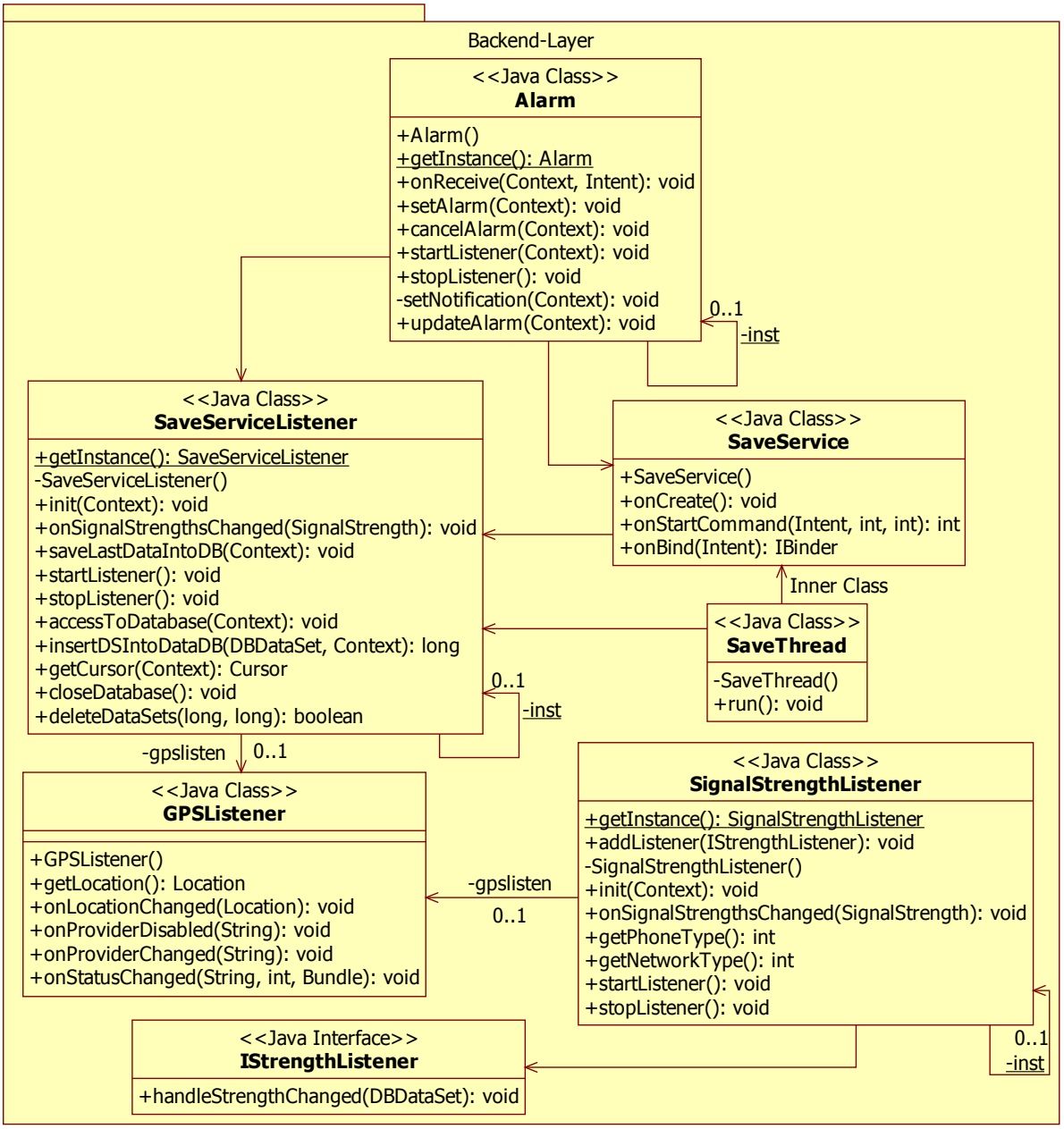

Abbildung 5.6: Klassendiagramm für Komponente $\langle C12 \rangle$

5.3.2 Erläuterung

Im Folgenden werden die Klassen in den Klassendiagrammen in Abbildung 5.5 und Abbildung 5.6 genauer beschrieben:
Um die Übersicht zu gewähren, wurde die Backend-Schicht in zwei unabhängige Klassendiagramme eingeteilt.
Aus Platzgründen werden Klassenattribute beider Diagramme sowie Get-Methoden der Klasse SettingsHelper nicht angezeigt.

Timer$\langle CL120 \rangle$

Aufgabe

Startet, in dem vom Benutzer gewünschten Intervall, den ConnectionService, um eine regelmäßige Verbindungen zum WebService zu gewähren.

Attribute

MINUTE: Ein finaler Integer, der den Wert von einer Minute in Millisekunden enthält.
inst: Die Singelton-Instanz des Timers, um nur einen einzigen Timer zu besitzen.

Operationen

Timer(): Der Konstruktor der Klasse.
getInstance(): Gibt die einzige Singleton-Instanz zurück und kreiert ggf. diese Instanz.
onReceive(Context context, Intent intent): Wird regelmäßig nach jeder Intervall-Einheit aufgerufen und startet den ConnectionService.
setAlarm(Context context): Startet den Timer und wählt das vom Benutzer gewählte Intervall, für die regelmäßige Verbindung.
cancelAlarm(Context context): Beendet den Alarm. Eine regelmäßige Verbindung bleibt aus.

Kommunikationspartner

Die Klasse ConnectionService, die jedes Mal aufgerufen wird, wenn eine Verbindung zum WebService hergestellt werden soll.

SettingsHelper$\langle CL121 \rangle$

Aufgabe

Hilfsklasse zum Lesen und Schreiben von Einstellungen des Benutzers.

Operationen

Diese Klasse besteht lediglich aus get- und set-Methoden, daher werden die Methoden dieser Klassen nicht näher erläutert.

FaultException⟨$CL122$⟩

Aufgabe

Rückgabe von Informationen im Fehlerfall.

Operationen

FaultException(String Errormessage): Diese Methode ist der Konstruktor der Klasse. Ein String mit Fehlerinformationen wird übergeben um es der aufrufenden Methode zur Verfügung zu stellen.

Kommunikationspartner

FaultException dient als Kommunikationshilfe zwischen der Clientklasse ⟨$CL124$⟩ und den aufrufenden Methoden dieser.

ConnectionService⟨$CL123$⟩

Aufgabe

Service, der wartet bis ein Timer ausgelöst wurde um dann die Daten zu übermitteln

Attribute

context: Der vom ConnectionService verwendete Context

pm: Ein PowerManager, um Zugriff auf die Stromeinstellungen zu erhalten und, auch im Standby-Modus des Mobilgerätes, Messungen durchzuführen.

wl: Ein WakeLock vom PowerManager, um die CPU im Standby-Modus des Mobilgerätes zu erwecken.

Operationen

ConnectionService(): Der Konstruktor des Services.

onCreate(): Beim Start des Services wird diese Operation ausgeführt

onStartCommand(Intent intent, int, int): Hier wird die Operation nach dem ablaufen des Timers ausgelöst und die Verbindung zum Clienten für die Übertragung hergestellt

onBind(Intent intent): Eine von Android bereitgestellte Operation, die jeder Service besitzen muss. Wird hier aber nicht verwendet, da keine Daten an den Service gebunden werden muss.

Kommunikationspartner

Die Klasse Timer ⟨$CL120$⟩, da dieser den ConnectionService startet.

Die Klasse Client ⟨$CL124$⟩, da der ConnectionService, den Clienten kontaktiert.

Client⟨$CL124$⟩

Aufgabe

Stellt die Kommunikationsschnittstelle zum XMap-Server bereit. Die Attribute definieren die Namespaces des Servers und sind selbsterklärend.

Operationen

allowAllSSL(): Akzeptiert sämtliche im Falle von HTTPS übertragene Zertifikate mithilfe der abgeleiteten _FakeX509TrustManager Klasse ⟨$CL125$⟩.

register(): Statisch aufrufbare Methode zum Erzeugen eines Clientobjekts um eine Registrierungsanfrage zu senden.

retrieveLoginData(): Lädt gespeichertes Passwort, Email und DeviceID in die entsprechenden Variablen.

registerSend(): Sendet eine Registrierungsanfrage für einen neuen Benutzer. Zusätzlich werden gerätespezifische Daten mit dem neu erstellten Login verknüpft.

getInstance(): Statisch aufrufbare Methode zum Erzeugen einen eingeloggten Clients.

Client(): Privater Konstruktor der Client Klasse .

endSession(): Loggt den Client aus und setzt INSTANCE auf null.

sendData(DBDataSet[]): Sendet das übergebene DBDataSet-Array.

login(): Loggt den Client ein.

Kommunikationspartner

FaultException dient der Client-Klasse als Fehlerrückgabehilfe ⟨$CL122$⟩. ConnectionService ⟨$CL123$⟩ ruft in einem vorgegeben Zeitintervall sendData auf. LoginActivity ⟨$CL111$⟩ und CreateAccountActivity ⟨$CL112$⟩ verwenden ebenfalls die Client-Klasse um die App ein, aus zu loggen oder zu registrieren. Die _FakeX509TrustManager ⟨$CL125$⟩ wird zur Verwendung von HTTPS aufgerufen.

_FakeX509TrustManager⟨$CL125$⟩

Aufgabe

Stellt Methoden für die Verwendung von selbst signierten Zertifikaten unter Android bereit.

Operationen

_FakeX509TrustManager(): Konstruktor der Klasse.

checkClientTrusted(X509Certificate[], String): Überlädt die Methode der Superklasse, so dass selbst signierte Zertifikate angenommen werden. Der String gibt den Authentifizierungstyp an.

checkServerTrusted(X509Certificate[], String): Überlädt die Methode der Superklasse, so dass ünbekannteSServer verwendet werden können. Der String gibt den Authentifizierungstyp an.

isClientTrusted(X509Certificate[]: Prüft ob Zertifikate in der Liste der bereits akzeptierten sind.
isServerTrusted(X509Certificate[]: Prüft ob die in den Zertifikat angegeben Server in der Liste der bereits akzeptierten sind.
getAcceptedIssuers(): Gibt alle akzeptierten Zertifikate zurück.

Kommunikationspartner

Wird von der Client-Klasse $\langle CL124 \rangle$ für HTTPS verwendet.

Alarm$\langle CL126 \rangle$

Aufgabe

Startet, in dem vom Benutzer gewünschten Intervall, den SaveService, wenn das Intervall nicht realtime ist, um eine regelmäßige Messung und Speicherung der Daten zu gewähren. Wenn das Intervall realtime ist, wird der SaveServiceListener gestartet.

Attribute

MINUTE: Ein finaler Integer, der den Wert von einer Minute in Millisekunden enthält.
nMN: Ein NotificationManager, um eine Benachrichtigung in der Benachrichtigungleiste des Mobilgerätes, zu verwalten.
inst: Die Singelton-Instanz des Alarms, um nur einen einzigen Alarm zu besitzen.
pm: Ein PowerManager, um Zugriff auf die Stromeinstellungen zu erhalten und, auch im Standby-Modus des Mobilgerätes, Messungen durchzuführen.
wl: Ein WakeLock vom PowerManager, um die CPU im Standby-Modus des Mobilgerätes zu erwecken.
sharedPref: Eine SharedPreferences Instanz, welche auf eine default Datei zeigt, in der Einstellungen gespeichert werden

Operationen

Alarm(): Der Konstruktor der Klasse.
getInstance(): Gibt die einzige Singleton-Instanz zurück und kreiert ggf. diese Instanz.
onReceive(Context context, Intent intent): Wird regelmäßig nach jeder Intervall-Einheit aufgerufen und startet den SaveService bzw. SaveServiceListener, um Messungen durchzuführen.
setAlarm(Context context): Startet den Alarm und wählt das vom Benutzer gewählte Intervall, für die regelmäßige Messung.
cancelAlarm(Context context): Beendet den Alarm. Eine regelmäßige Verbindung bleibt aus.
startListener(Context context): Startet den SaveServiceListener und erweckt die CPU.
stopListener(): Beendet den SaveServiceListener und legt die CPU wieder schlafen, falls das Mobilgerät in Standby ist.

setNotification(Context context): Setzt eine Benachrichtigung in der Benachrichtigungsleiste des Mobilgerätes, um dem Benutzer zu signalisieren, dass Messungen laufen.
updateAlarm(Context context): Beendet den alten Alarm und startet einen neuen neu, falls des Intervall neu eingestellt wird.

Kommunikationspartner

Die Klasse SaveServiceListener $\langle CL127 \rangle$, da der Alarm den SaveServiceListener startet, wenn das Messintervall auf realtime gestellt ist.
Die Klasse SaveService $\langle CL128 \rangle$, da der Alarm den SaveService verwendet, wenn das Messintervall auf einen Minutenwert gestellt ist.

SaveServiceListener$\langle CL127 \rangle$

Aufgabe

Messen und Speichern von Messdaten.

Attribute

MINUTE: Ein finaler Integer, der den Wert von einer Minute in Millisekunden enthält.
gpslisten: Objekt der Klasse GSPListener, welches auf eine Veränderung des aktuellen Standpunktes lauscht.
cl: Objekt der Klasse GsmCellLocation, um die Position des gerade verwendeten Sendemastes zu bestimmen.
tm: Objekt der Klasse TelephonyManager, um die benötigten Telefondaten zu bestimmen.
lm: Objekt der Klasse LocationManager, um die aktuelle GPS-Position zu bestimmen.
db: Dieses Attribut stellt ein Datenbank-Objekt dar, um darauf die Operationen auszuführen.
ds: Ein Datensatz, der sich im Laufe der Messung ändert und dann gespeichert wird.
inst: Die Singleton-Instanz des SaveServiceListener.
context: Dieses Attribut ist der von Android benötigte Context, der jeweils gerade verwendeten Activity der GUI.
measure_interval: Das in den Einstellungen gewählte Messintervall, in welchem Abstand gemessen wird.
sharedPref: Ein Objekt, um die Einstellungen auslesen zu können.

Operationen

getInstance(): Erhält die Singelton-Instanz und erstellt diese gegebenenfalls.
SaveServiceListener(): Der Default-Konstruktor, der die Singleton-Instanz des SaveServiceListeners kreiert.
init(Context context): Initialisiert den SaveServiceListener.
onSignalStrengthsChanged(SignalStrength signStrength): Verändert alle Daten von ds auf den aktuellsten Stand, wenn sich die Signalstärke ändert.

saveLastDataIntoDB(Context context): speichert den letzten Datensatz in der Datenbank und stellt alle Daten auf den Standardwert zurück.
startListener(): Starten den SaveServiceListener für die Messung.
stopListener(): Beendet den SaveService, wenn nicht mehr gemessen soll.
accessToDatabase(Context context): Stellt Zugriff auf die Datenbank her.
insertDSIntoDatabase(DBDataSet ds, Context context): Fügt den Datensatz ds in die Datenbank ein.
getCursor(Context context): Wird nicht verwendet, muss aber wegen DBAccess vorhanden sein.
closeDatabase(): Schließt die Datenbank und beendet den Zugriff.
deleteDataSets(long from, long to): Wird nicht verwendet, muss aber wegen DBAccess vorhanden sein.

Kommunikationspartner

Das Interface DBAccess $\langle CL132 \rangle$ welches implementiert wird.
Die Klasse Database $\langle CL130 \rangle$, um Daten in der Datenbank zu speichern.
Die Klasse DBDataSet $\langle CL131 \rangle$, da der SaveServiceListener die Messdaten auf einem Datensatz erstellt und diesen dann speichert.
Im Klassendiagramm in Abbildung 5.7 nicht dargestellt:
Die Klasse SaveService $\langle CL128 \rangle$, um eine Messung zu starten, wenn das Messintervall größer oder gleich als eine Minute ist.
Die Klasse Alarm $\langle CL126 \rangle$, um eine Messung zu starten, wenn das Messintervall kleiner als eine Minute ist.
Die Klasse GPSListener $\langle CL1212 \rangle$, um die GPS Position zu überwachen.
Die von Android bereitgestellt Klasse PhoneStateListener, um die Signalstärke zu überwachen und auf eine Änderung zu reagieren.

SaveService$\langle CL128 \rangle$

Aufgabe

Service, der eine neue Messung durchführt und, mittels innere Klasse SaveThread, eine halbe Minute wartet, damit GPS eine Position finden kann.

Attribute

context: Der vom Service verwendete Context, damit die innere Klasse auch Zugriff auf diesen hat.
pm: Ein PowerManager, um Zugriff auf die Stromeinstellungen zu erhalten und, auch im Standby-Modus des Mobilgerätes, Messungen durchzuführen.
wl: Ein WakeLock vom PowerManager, um die CPU im Standby-Modus des Mobilgerätes zu erwecken.

Operationen

SaveService(): Der Konstruktor des Services.

onCreate(): Eine von Android bereitgestellte Operation, die beim Start des Services aufgerufen wird. **onStartCommand(Intent intent, int flag, int startId)**: Eine von Android bereitgestellte Operation, die nach dem Start ausgeführt wird. Hier wird die innere Klasse SaveThread gestartet, der SaveServiceListener gestartet und die CPU wird geweckt und wach gehalten.

onBind(Intent intent): Eine von Android bereitgestellte Operation, die jeder Service besitzen muss. Wird hier aber nicht verwendet, da keine Daten an den Service gebunden werden muss.

Kommunikationspartner

Die Klasse Alarm $\langle CL126 \rangle$, da dieser den SaveService startet.

Die Klasse SaveServiceListener $\langle CL127 \rangle$, da der SaveService, den Listener startet.

SaveThread$\langle CL129 \rangle$

Aufgabe

Ein Thread, in einem extra Prozess verlagert, um eine halbe Minute zu warten und die gemessenen Daten in der Datenbank zu speichern.

Attribute

keine

Operationen

SaveThread(): Der Konstruktor des Threads.

run(): Wird bei Start des Threads aufgerufen. Die Operation wartet eine halbe Minute, damit der GPS-Sensor die aktuelle Position bestimmen kann, speichert diese Daten in der Datenbank, stoppt den Listener, legt die CPU wieder schlafen und beendet den Service.

Kommunikationspartner

Die Klasse SaveService $\langle CL128 \rangle$, da der SaveService die äußere Klasse dieser ist.

Die Klasse SaveServiceListener $\langle CL127 \rangle$, um den Listener zu beenden und die von ihm gemessenen Daten zu speichern.

SignalStrengthListener$\langle CL1210 \rangle$

Aufgabe

Messen von Messdaten, wenn sich die Signalstärke ändert. Indirekte Benachrichtigung von Listeners der betrachteten Klasse über die neuen Messdaten findet durch das Interface IStrengthListener statt.

Attribute

gpslisten: Objekt der Klasse GSPListener, welches auf eine Veränderung des aktuellen Standpunktes lauscht.

ds: Eine Instanz der Klasse DBDataSet.

cl: Objekt der Klasse GsmCellLocation, um die Position des gerade verwendeten Sendemastes zu bestimmen.

tm: Objekt der Klasse TelephonyManager, um die benötigten Telefondaten zu bestimmen.

lm: Objekt der Klasse LocationManager, um die aktuelle GPS-Position zu bestimmen.

listeners: HashSet, in welcher Listeners der betrachteten Klasse gesammelt werden.

Operationen

getInstance(): Diese Methode erzeugt eine Singleton-Instanz der Klasse und gibt sie zurück.

addListener(IStrengthListener l): Diese Methode wird von Klassen der Visualisierungsmöglichkeiten (GoogleMaps, Tabelle, Graph) aufgerufen, um sich als Listener zu der Menge listeners hinzuzufügen.

SignalStrengthListener(): Privater Konstruktor beugt Instanziierung durch andere Klassen vor.

init(Context context): Initialisierung des TelephonyManagers sowie Locationmanagers im gegebenen Context der verwendeten Activity.

onSignalStrengthsChanged(SignalStrength): Eine von Android bereitgestellte Klasse, die auf eine Veränderung der Signalstärke achtet. Aktuelle Telefondaten werden gesetzt und in die Datenbank gespeichert. Desweiteren wird durch die Menge der Listeners iteriert und die Methode handleStrengthChanged(ds) aufgerufen, um den neuen Datensatz entsprechend zu behandeln.

getPhoneType(): Gibt den Telefontyp in Form einer Konstante zurück, beispielsweise GSM.

getNetworkType(): Gibt den Netzwerktyp in Form einer Konstante zurück, beispielsweise LTE.

startListener(): Diese Methode ruft die Methode onSignalStrengthsChanged auf. Listener der betrachteten Klasse reagieren auf eine veränderte Signalstärke.

stopListener(): Listener der betrachteten Klassen hören auf, auf eine veränderte Signalstärke zu achten.

Kommunikationspartner

Die Klasse DBDataSet $\langle CL131 \rangle$, die ein einzelnes Datenbanktupel der Datenbank darstellt, damit dieser Datensatz weiterverwendet werden kann.

Die Klasse GSPListener $\langle CL1212 \rangle$, welche auf eine Veränderung des Standortes achtet.

Das Interface IStrengthListener $\langle CL1211 \rangle$, dessen Methode handleStrengthChanged in den Listenern der betrachteten Klasse aufgerufen wird, wenn eine Änderung der Signalstärke in onSignalStrengthChanged beobachtet wird.

IStrengthListener⟨$CL1211$⟩

Aufgabe

Dieses Interface dient als Kommunikations-Schnittstelle zwischen Presentationsschicht und Backend, wenn sich die Signalstärke ändert. Ist das der Fall, wird ein neuer Datensatz im Backend gemessen und gespeichert und in der Presentationsschicht neu visualisiert.

Operationen

handleStrenghChanged(DBDataSet ds): Diese abstrakte Methode wird in den Listeners der Klasse SignalStrenghChanged implementiert. In ihr wird für jeden Listener (GoogleMaps, Tabelle, Graph) spezifisch das Verhalten bei einem neuen Datensatz festgelegt.

Kommunikationspartner

Die Klasse DBDataSet ⟨$CL131$⟩, die ein einzelnes Datenbanktupel der Datenbank darstellt, damit dieser Datensatz weiterverwendet werden kann.

GPSListener⟨$CL1212$⟩

Aufgabe

Messen der aktuellen Position mittels GPS.

Attribute

ploc: Objekt der Klasse Location, welches die aktuelle Position beschreibt.

Operationen

getLocation(): Get-Methode für GPS Position.

onLocationChanged(Location): Aktualisiert die gemessene Position.

onProviderDisabled(String): Wird aufgerufen, wenn GPS-Empfang vom Nutzer deaktiviert wurde. Überschreibt eine Android Methode und besitzt keine eigene Funktion.

onProviderEnabled(String): Wird aufgerufen, wenn GPS-Empfang vom Nutzer aktiviert wurde. Überschreibt eine Android Methode und besitzt keine eigene Funktion.

onStatusChanged(String, int, Bundle): Wird aufgerufen, wenn sich der GPS-Status ändert. Überschreibt eine Android Methode und besitzt keine eigene Funktion.

Kommunikationspartner

Die Klasse SaveServiceListener ⟨$CL127$⟩, welche die Messdaten abspeichert. Die Klasse SignalStrengthListener ⟨$CL1210$⟩, welche bei veränderter Signalstärke Daten misst.

5.4 Implementierung von Komponente $\langle C13 \rangle$: DatabaseClient:

Diese Komponente implementiert die Datenbank des Clients. Dabei werden alle Einfüge-, Lösch- und Leseoperationen verarbeitet sowie alle Datenbankzugriffe aller Klassen, die einen Zugriff benötigen, werden vorbereitet.
Die Komponente ist in Java geschrieben und verwendet zusätzlich SQL-Befehle, um den Zugriff auf die Datenbank zu gewähren. Dafür werden vom Android-SDK `“SQLiteDatabase“`, um die Datenbank zu verwenden, und `“SQLiteOpenHelper“`, um den Zugriff auf die Datenbank zu vereinfachen, genutzt.
Für die Implementierung dieser Komponente werden keine zusätzlichen Bibliotheken verwendet.

5.4.1 Klassendiagramm

Im Klassendiagramm in Abbildung 5.7 wird die Architektur der Komponente $\langle C13 \rangle$ beschrieben.

5.4.2 Erläuterung

Im Folgenden werden die Klassen des Klassendiagramms in Abbildung 5.7 genauer beschrieben: Um die Übersicht zu vereinfachen, wurden die Get- und Set-Methoden der Klasse DBDataSet entfernt.

Database$\langle CL130 \rangle$

Aufgabe

Verwaltung der Datenbank

Attribute

db: Dieses Attribut ist das von Android bereitgestellte Datenbankobjekt SQLiteDatabase, auf dem die SQL-Befehle ausgeführt werden.

TABLE: Dieses Attribut ist ein finaler String, der den Namen der einzigen Tabelle der Datenbank enthält.

maxSize: Dieses Attribut beinhaltet die maximal mögliche Datenbankgröße in Byte

dbname: Dieses Attribut enthält den vollständigen Name der Datenbank, inklusive Dateiendung “.dat“

context: Dieses Attribut ist der von Android benötigte Context, der jeweils gerade verwendeten Activity der GUI.

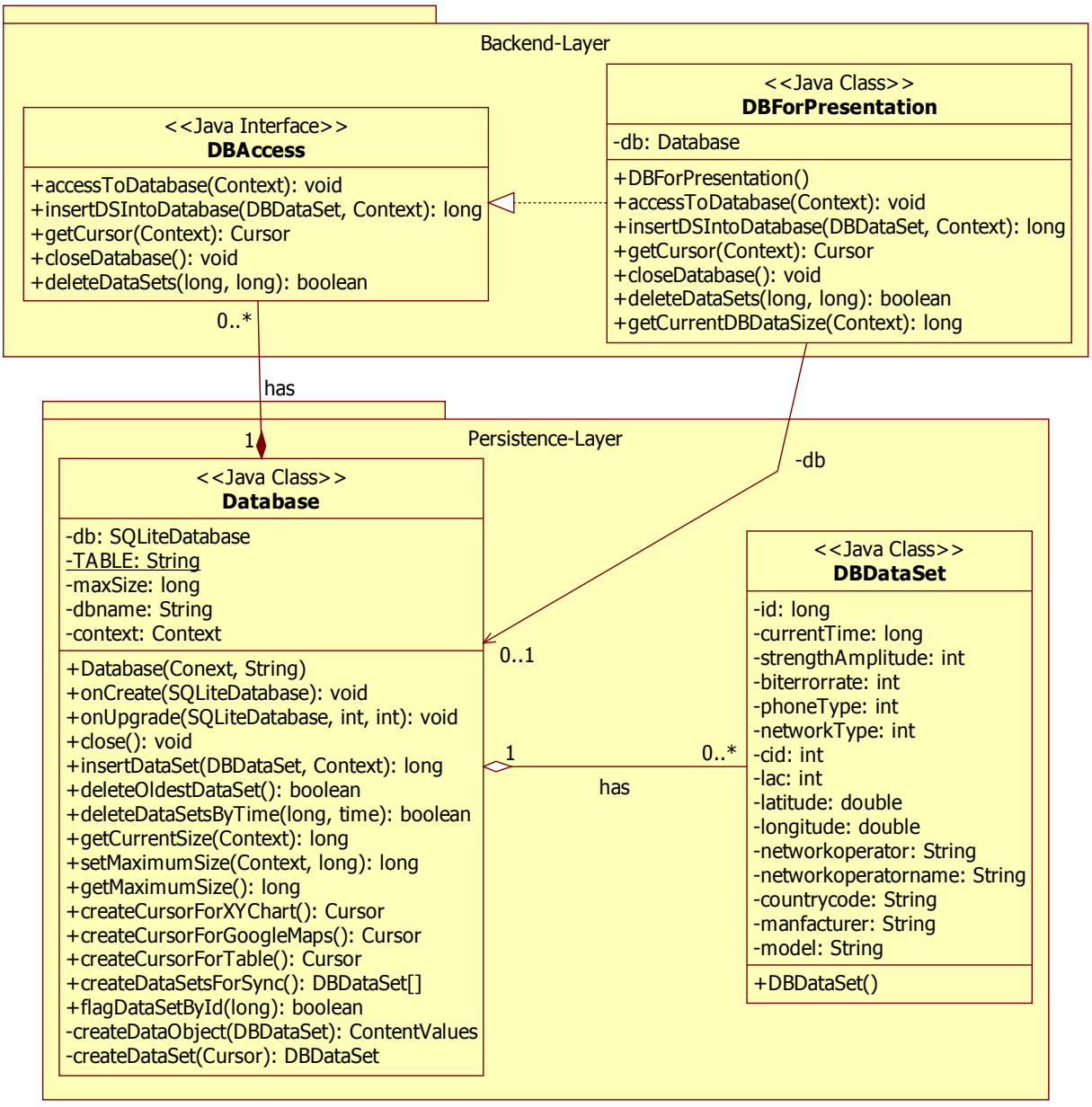

Abbildung 5.7: Klassendiagramm für Komponente $\langle C13 \rangle$

Operationen

Database(Context activity, String dbName): Diese Operation ist der Konstruktor und erstellt ein Datenbankobjekt und damit auch den endgültigen Zugriff auf die Datenbank

onCreate(SQLiteDatabase db): Eine von Android bereitgestellte Operation, um die Datenbank zu kreieren, falls noch Keine Datenbank mit dem Namen dbname existiert.

onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion): Eine von Android bereitgestellte Operation, falls sich die Datenbank ändert und eine Versionsnummer erhält.

close(): Eine von Android bereitgestellte Operation, um die Datenbank zu schließen, wenn Zugriff nicht mehr benötigt wird.

insertDataSet(DBDataSet ds, Context context): Diese Operation fügt die in dem Datensatz enthaltenden Daten in die Datenbank ein.

deleteOldestDataSet(): Löscht den ältesten Datensatz in der Datenbank.

deleteDataSetsByTime(long from, long to): Löscht alle Datensätze, die zwischen den Zeitwerten (in Millisekunden) from und to liegen.

getCurrentSize(Context context): Gibt die aktuelle Datenbank-Speichergröße zurück.

setMaximumSize(Context context, long numBytes): Setzt die maximale Speichergröße der Datenbank auf numBytes und löscht ggf. die älteste Datensätze, falls numBytes kleiner als die aktuelle Datenbankgröße ist.

getMaximumSize(): Gibt die maximale Datenbank-Speichergröße zurück.

createCursorForXYChart(): Liest die Datenbank für alle benötigten Werte für den XY-Graph aus.

createCursorForGoogleMaps(): Liest die Datenbank für alle benötigten Werte für das Google Maps Overlay aus.

createCursorForTable(): Liest die Datenbank für alle benötigten Werte für die Tabelle aus.

createDataSetsForSync(): Liest die Datenbank nach allen Daten aus, die noch nicht an den WebService übertragen wurden.

flagDataSetById(long id): Der Datensatz mit der zugehörigen id wird als an den WebService übertragen markiert.

createDataObject(DBDataSet ds): Bereitet alle im DBDataSet vorhandenen Daten für das SQL-Insert-Statement vor.

createDataSet(Cursor cursor): Setzt die Daten von der Position des Cursors, auf den ein Pointer zeigt, zu einem DBDataSet.

Kommunikationspartner

Die Klasse DBDataSet $\langle CL131 \rangle$, die einem Datenbanktupel der Datenbank entsprechen kann, damit dieser Datensatz weiterverwendet werden kann.

Das Interface DBAccess $\langle CL132 \rangle$, dass für die benötigten Klassen den Datenbankzugriff

bereitstellt.
Die Klasse DBForPresentation ⟨$CL133$⟩, die DBAccess implementiert und ein Datenbankobjekt besitzt.
Die Klasse SaveServiceListener ⟨$CL127$⟩, die DBAccess implementiert und zum Speichern einer Messung verwendet wird.
Im Klassendiagramm in Abbildung 5.7 nicht dargestellt:
Die inneren Klasse DeleteTask ⟨$CL114$⟩ von der Klasse Settingsactivity ⟨$CL113$⟩, da wenn die maximale Datenbankgröße minimiert wird, wird gegebenfalls eine Datenlöschung von statten gehen.

DBDataSet⟨$CL131$⟩

Aufgabe

Darstellung eines Datensatzes der Datenbank

Attribute

id: Dieses Attribut entspricht der ID des Datensatzes, die beim Einfügen in der Datenbank erzeugt wird.
currentTime: Dieses Attribut ist die Zeit, wann der Datensatz gemessen wurde.
strengthAmplitude: Dieses Attribut zeigt die Signalstärke des Datensatzes.
biterrorrate: Dieses Attribut entspricht der möglichen Bit-Fehler, der bei der Datenübertragung entstehen kann.
phoneType: Dieses Attribut stellt den Funktyp der Handyverbindung dar.
networkType: Diese Attribut ist der verwendete Netwerktyp der Handyverbindung.
cid: Dieses Attribut zeigt die Cell-ID des Sendemastes, der verwendet wurde.
lac: Dieses Attribut entspricht dem Local-Area-Code, in dem sich der verwendete Sendemaste befindet.
latitude: Dieses Attribut ist der Breitegrad der gespeicherten Position.
longitude: Dieses Attribut ist der Längengrad der gespeicherten Position.
networkoperator: Dieses Attribut bezeichnet, die ID des Netzwerkproviders.
networkoperatorname: Dieses Attribut ist der Name des Netwerkproviders.
countrycode: Dieses Attribut zeigt den Code des Landes, in dem man sich befand.
manufactorer: Der Smartphone-Hersteller.
model: Das Smartphone Model.

Operationen

Zu jedem Attribut gibt es sowohl eine get-Operation und eine set-Operation, die wegen der Übersichtlichkeit und der besseren Lesbarkeit, sowohl im Klassendiagramm Abbildung 5.7 als auch hier nicht weiter beachtet werden.
DBDataSet(): Konstruktor, der ein Datensatz-Objekt erzeugt.

Kommunikationspartner

Die Klasse Database ⟨$CL130$⟩, da ein Datensatz ein Teil der Datenbank ist.
Im Klassendiagramm in Abbildung 5.7 nicht dargestellt:
Die Klasse SaveServiceListener ⟨$CL127$⟩, der auf einem DBDataSet die Messdaten aufträgt und in der Datenbank speichert.
Die Klasse SignalStrengthListener ⟨$CL1210$⟩, da die Daten bei Signalstärkeänderung auf einem DBDataSet aufgetragen werden.

DBAccess⟨$CL132$⟩

Aufgabe

Verwaltung des Datenbankzugriffs für Klassen, außerhalb des Pakets Persistence-Layer.

Attribute

keine

Operationen

accessToDatabase(Context context): Stellt Zugriff auf die Datenbank her.
insertDSIntoDatabase(DBDataSet ds, Context context): Fügt den Datensatz ds in die Datenbank ein.
getCursor(Context context): Erhält einen Cursor, mit Daten aus der Datenbank, zur Weiterverarbeitung.
closeDatabase(): Schließt die Datenbank und beendet den Zugriff.
deleteDataSets(long from, long to): Löscht alle Datensätze, die zwischen den Zeitwerten (in Millisekunden) from und to liegen.

Kommunikationspartner

Die Klasse Database ⟨$CL130$⟩, da DBAcess auf der Datenbank und auf deren Operationen arbeitet.
Die Klasse DBForPresentation ⟨$CL131$⟩, die DBAccess implementiert, um die Datenbank in der Abbildung 5.7 nicht aufgeführten Presentation-Layer zur Verfügung zu stellen.
Im Klassendiagramm in Abbildung 5.7 nicht dargestellt:
Die Klasse SaveServiceListener ⟨$CL127$⟩, die DBAccess implementiert, um gemessene Daten in der Datenbank speichern zu können.
Die inneren Klasse DeleteTask ⟨$CL114$⟩ von der Klasse Settingsactivity ⟨$CL113$⟩, da wenn die maximale Datenbankgröße minimiert wird, wird gegebenfalls eine Datenlöschung von statten gehen wird. Dazu wurde von DeleteTask DBAccess implementiert, für den Datenbankzugriff.

DBForPresentation⟨$CL133$⟩

Aufgabe

Die Datenbank der Presentation-Layer zur Verfügung zu stellen und die dazugehörigen Daten durchzuschleusen.

Attribute

db: Dieses Attribut stellt ein Datenbank-Objekt dar, um darauf die Operationen auszuführen.

Operationen

accessToDatabase(Context context): Stellt Zugriff auf die Datenbank her.

insertDSIntoDatabase(DBDataSet ds, Context context): Wird für die Presentation-Layer nicht benötigt, muss aber wegen dem Interface DBAccess enthalten sein.

getCursor(Context context): Erhält einen Cursor, mit Daten aus der Datenbank, für die Visualisierungs-Oberflächen. Je nach Visualisierung wird der passende Cursor ausgewählt.

closeDatabase(): Schließt die Datenbank und beendet den Zugriff.

deleteDataSets(long from, long to): Löscht alle Datensätze, die zwischen den Zeitwerten (in Millisekunden) from und to liegen.

Kommunikationspartner

Das Interface DBAccess ⟨$CL132$⟩ welches implementiert wird.

Die Klasse Database ⟨$CL130$⟩, da DBForPresentation ein Datenbank-Objekt enthält, um auf diesem die Datenbank-Operationen anzuwenden.

Im Klassendiagramm in Abbildung 5.7 nicht dargestellt: Die Klassen MapsActivity ⟨$CL115$⟩, DBTableActivity ⟨$CL119$⟩ und XYChartActivity ⟨$CL117$⟩, um die aus der Datenbank ausgelesenen Daten zur Visualisierung durchzuschleusen.

5.5 Implementierung von Komponente $\langle C20 \rangle$: Server:

Die Oberkomponente ist eine Repräsentation des Servers und kapselt die bereits zuvor in Kapitel 3 erwähnten untergeordneten Komponenten.

5.5.1 Paketdiagramm

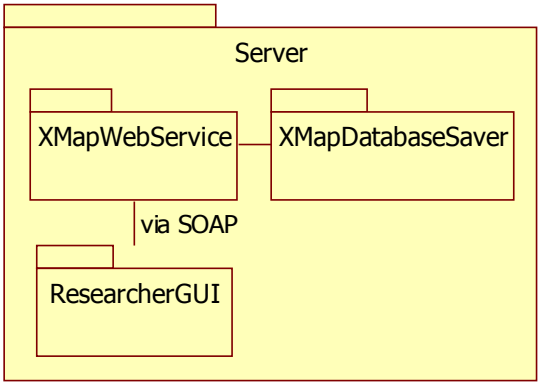

Abbildung 5.8: Paketdiagramm für Komponente $\langle C20 \rangle$

5.5.2 Erläuterung

- **XMapWebService:** stellt für die App Methoden zum Registrieren und Einloggen zur Verfügung, sowie zum Speichern der Messwerte in der Datenbank. Außerdem übergibt es an die WebApplikation die gefilterten Daten für die grafische Visualisierung.
- **XMapDatabaseSaver:** speichert die Daten der Datenbanktabelle 'MeasuredData' in XML-Dateien nach Sender(NetOperator, cellID, lac) und Tag ab.
- **ResearcherGUI:** Webseite, welche die Auswertungsfunktionen zur Verfügung stellt.

5.6 Implementierung von Komponente $\langle C21 \rangle$: WebService:

Der WebService ist für den Server das Tor zur Außenwelt. Über das SOAP-Protokoll kommuniziert er mit den Android-Geräten und mit der WebApp für Forscher.

5.6.1 Klassendiagramm

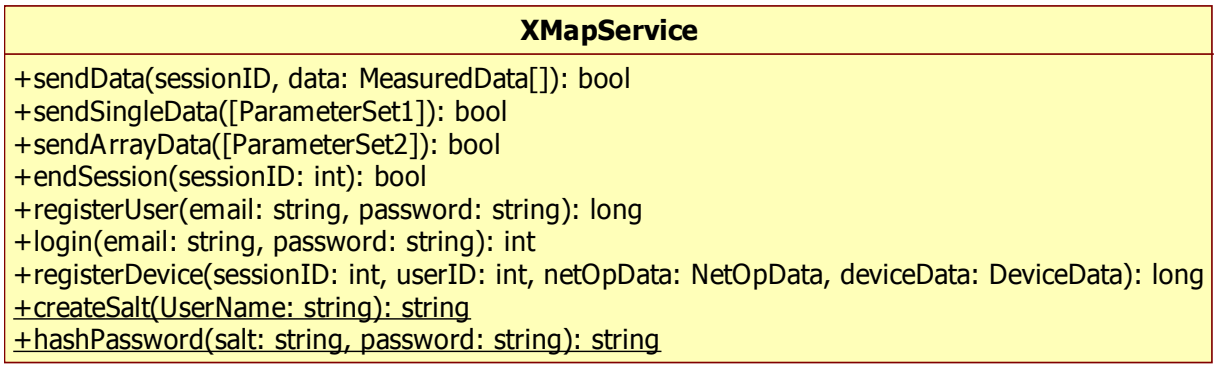

Abbildung 5.9: Klassendiagramm für Komponente $\langle C21 \rangle$

- ParameterSet1 = sessionID: int, gsmSignalStrength: int, gsmBitErrorRate: int, networkType: int, longitude: string, latitude: string, cellID: int, lac: int, networkOperator: int, netOpName: string, countryCode: string, userEquipmentID: int, timestamp: string
- ParameterSet2 = sessionID: int, gsmSignalStrength: int[], gsmBitErrorRate: int[], networkType: int[], longitude: string[], latitude: string[], cellID: int[], lac: int[], networkOperator: int[], netOpName: string[], countryCode: string[], userEquipmentID: int, timestamp: string

5.6.2 Erläuterung

Methoden

- **sendData(int sessionID, MeasuredData[] data):** überprüft, ob die SessionID gültig ist und speichert daraufhin die Daten in der Datenbank.
- **sendSingleData([ParameterSet1]):** überprüft, ob die SessionID gültig ist und speichert daraufhin die Daten in der Datenbank.
- **sendArrayData([ParameterSet2]):**überprüft, ob die SessionID gültig ist und speichert daraufhin die Daten in der Datenbank.
- **endSession(int sessionID):** Methode, um eine Session zu beenden.

- **registerUser(string email, string password):** Methode, zum Registrieren am WebService.
- **login(string email, string password):** Methode zum Einloggen am WebService, überprüft Benutzerdaten und übergibt bei richtigen Daten eine SessionID.
- **registerDevice(int sessionID, int userID, NetOpData netopData, DeviceData deviceData):**Methode, um ein neues Gerät am WebService zu registrieren, übergibt bei gültiger SessionID die UserEqipmentID.
- **createSalt(string userName):** generiert einen Salt aus dem Username für das gehashte Passwort
- **hashPassword(string salt, string password):** Methode um ein gehashtes Password mithilfe eines Salts zu generieren.

5.7 Implementierung von Komponente $\langle C22 \rangle$: SessionManagement:

Die Implementierung des SessionManagements wird im folgenden Abschnitt beschrieben.

5.7.1 Klassendiagramm

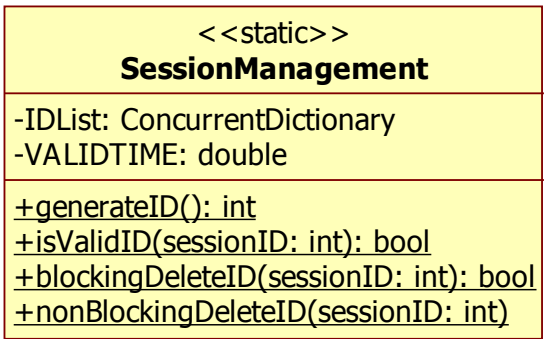

Abbildung 5.10: Klassendiagramm für Komponente $\langle C22 \rangle$

5.7.2 Erläuterung

Abbildung 5.10 zeigt die statische Klasse SessionManagement. Sie dient der Verwaltung der aktuell gültigen SessionIDs über alle Instanzen des Services hinweg.

Attribute

- **IDList:** Eine Instanz einer threadsicheren Implementierung eines Dictionaries. Als Key dient die jeweilige SessionID, als Value wird ein Zeitstempel eingetragen.
- **VALIDTIME:** Eine Konstante, welche die Zeit in Minuten enthält, nach welcher eine SessionID ablaufen soll. (Typ „double“ aus Implementierungsgründen notwendig.)

Methoden

- **generateID():** Generiert eine Zufallszahl als neue SessionID und gibt diese zurück.
- **isValidID(int sessionID):** Überprüft, ob in IDList eine entsprechende ID vorhanden ist und ob sie gültig ist.
- **blockingDeleteID(int sessionID):** Versucht die SessionID aus der IDList zu entfernen, signalisiert Erfolg oder Misserfolg.
- **nonBlockingDeleteID(int sessionID):** Wie blockingDeleteID, benachrichtigt aber nicht, sodass der Aufrufer nicht auf das Ergebnis warten muss.

5.8 Implementierung von Komponente $\langle C23 \rangle$: DatabaseConnection:

Die Klasse DatabaseConnection verbindet den WebService mit der Datenbank. In ihr werden die LINQ to SQL-Aufrufe getätigt, welche Datensätze in die Datenbank schreiben oder aus der Datenbank lesen.

5.8.1 Klassendiagramm

Abbildung 5.11

ParameterSet1 = from: DateTime, to: DateTime, provider: string[], devices: string[], models: string[], networkType: byte[], long1: float, long2: float, lat1: float, lat2: float, userID: long, all: bool

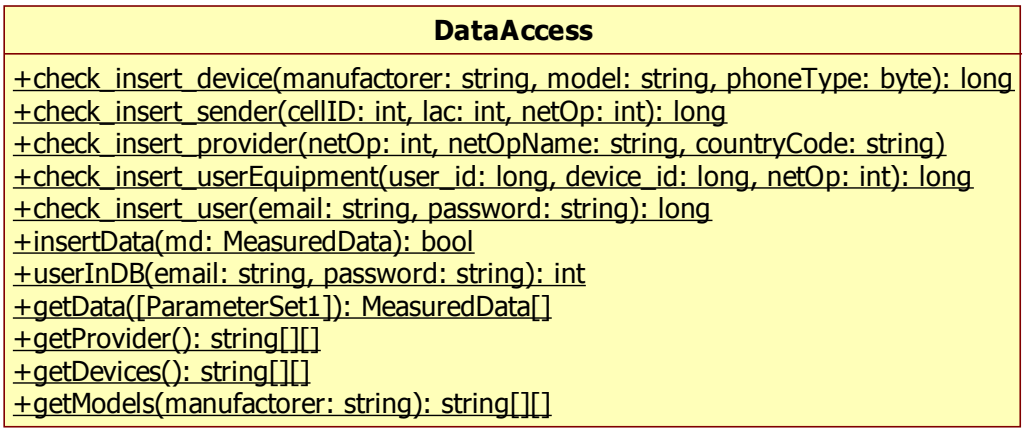

Abbildung 5.11: Klassendiagramm für Komponente $\langle C23 \rangle$

5.8.2 Erläuterung

Methoden

- **insertData(MeasuredData md):** Speichert die übergebenen Messwerte in der Datenbank.
- **check_insert_sender(int cellID, int lac, int netOp):** überprüft, ob es den übergebenen Sender schon in der Datenbank gibt und speichert diesen ansonsten. Übergibt in beiden Fällen die SenderID.
- **check_insert_provider(int netOp, string, netOpName, string countrycode):** überprüft, ob es den übergebenen Provider schon in der Datenbank gibt und speichert diesen ansonsten. Keine Rückgabe einer ID, da netOp eindeutig ist.
- **check_insert_device(string manufactorer, string model, byte phonetype:** überprüft, ob es das übergebene Gerät schon in der Datenbank gibt und speichert es ansonsten. Übergibt in beiden Fällen die DeviceID.
- **check_insert_UserEquipment(long user_id, long, device_id, int netOp):** überprüft, ob es das übergebene UserEquipment schon in der Datenbank gibt und speichert es ansonsten. Übergibt in beiden Fällen die UserEquipmentID.
- **check_insert_user(string email, string password):** überprüft, ob es den angegebenen Benutzer schon gibt und speichert diesen ansonsten. Gibt zurück, ob der Benutzer schon existiert oder erstellt wurde und, ob die E-Mail das richtige Format hat.
- **userInDB(string email, string password:** Methode, die zurückgibt, ob korrekte Benutzerdaten übergeben wurde, ob die E-Mail-Adresse gar nicht existiert oder ob das Passwort falsch ist.

- **getProvider():** übergibt alle in der Datenbank gespeicherten Provider.
- **getDevices():** übergibt alle in der Datenbank gespeicherten Geräte.
- **getModels(string manufactorer):** übergibt alle in der Datenbank gespeicherten Handymodelle eines bestimmten Herstellers.
- **getData([ParameterSet1]):** übergibt alle Messwerte, die den übergebenen Filtern entsprechen.

5.9 Implementierung von Komponente $\langle C24 \rangle$: DatabaseServer:

Die Klasse $\langle C24 \rangle$ DatabaseServer symbolisiert die auf dem Server verwendete Datenbank im Komponentendiagramm Abbildung 3.1. Als Basis für diese dient eine Microsoft-SQL-Datenbank, welche in Abschnitt 6.2, Datenmodell Server, näher erläutert wird. In diesem Kapitel wird sie daher nicht weiter betrachtet.

5.10 Implementierung von Komponente $\langle C25 \rangle$: ResearcherGUI:

Aufgrund von kürzlich aufgetretenen Problemen in der Implementierung war es zu diesem Zeitpunkt nicht möglich entsprechende Paket-/Klassendiagramme und die dazugehörigen Erläuterungen anzufertigen. Diese können nachgeliefert werden, sobald die neue Implementierung einen wieder einen weitgehend stabilen Zustand eingenommen hat.

5.11 Implementierung von Komponente $\langle C26 \rangle$: DatabaseSaver:

Das Programm DatabaseSaver ist ein Hilfsprogramm, welches Datensätze aus der Datenbank strukturiert in XML-Dateien ablegen kann.

5.11.1 Klassendiagramm

Abbildung 5.12

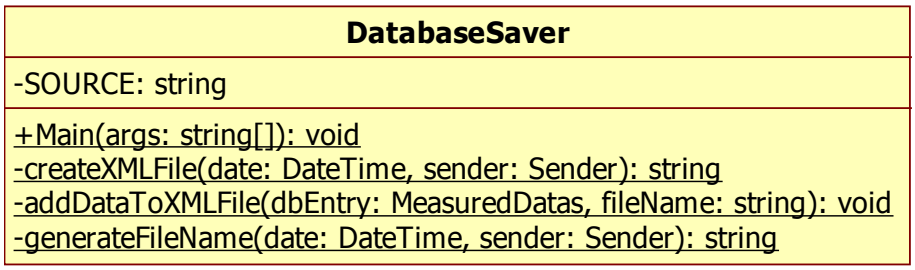

Abbildung 5.12: Klassendiagramm für Komponente $\langle C26 \rangle$

5.11.2 Erläuterung

Abbildung 5.12 zeigt die statische Klasse DatabaseSaver. Sie dient dem Extrahieren der Messdatensätze aus der Datenbank, woraufhin jene in XML-Dateien abgelegt werden.

Attribute

- **SOURCE:** Eine Konstante, welche den Pfad zum Zielordner der XML-Dateien enthält.

Methoden

- **Main(string[] args):** Enthält die Kern der Programmlogik, durch welche die passenden Datensätze ausgewählt und weggespeichert werden.
- **createXMLFile(DateTime date, Sender sender):** Legt eine neue XML-Datei mithilfe der übergebenen Datensätze an.
- **addDataToXMLFile(MeasuredDatas dbEntry, string fileName):** Erwartet eine Tabellenzeile und eine Zieldatei, in welche die Tabellenzeile abgelegt werden soll.
- **generateFileName(DateTime date, Sender sender):** Generiert durch String-Konkatenation des Source-Pfades mit ausgewählten Werten der übergebenen Objekte einen Dateinamen.

6 Datenmodell

Sowohl Client als auch Server müssen Daten über einen längeren Zeitraum speichern können. Daher werden in beiden relationale Datenbanken für die Datenhaltung eingesetzt. Obwohl die Datenbanken im Grunde ähnliche Daten speichern müssen, sind ihre Anforderungen sehr verschieden, weshalb sie grundlegend verschiedene Designs aufweisen. Sie werden in diesem Kapitel daher streng getrennt betrachtet.

6.1 Client

Clientseitig wird für die Datenhaltung auf die in Android enthaltene SQLite-Datenbank zurückgegriffen, welche im Folgenden näher erläutert wird.

6.1.1 Diagramm

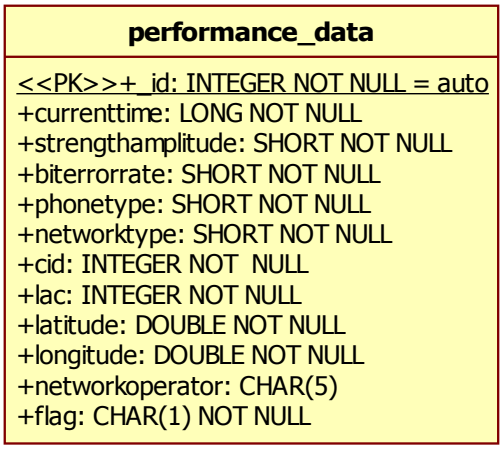

Abbildung 6.1: Klassendiagramm Client-Datenbank

Hinweise zum Diagramm Abbildung 6.1:

- PK = Primary Key, auto = wird automatisch mit Werten befüllt

6.1.2 Erläuterung

Die Datenbank des Clients wurde so konzipiert, dass Einfüge-, Lese- und Lösch-Operationen möglichst schnell vonstattengehen. Das liegt daran, dass wegen der schwächeren Hardware der Mobilgeräte und dem ressourcensparenden Android-Betriebssystem Abfragen der Datenbank viel Zeit in Anspruch nehmen können. Dieser Nachteil kann durch eine schnelle, einfache Datenbank, ohne komplexe Beziehungen, ausgeglichen werden, um die Wartezeit für den Benutzer auf ein Minimum zu reduzieren. Aus diesem Grund besitzt die Datenbank des Clients nur eine Entität ohne weitere Beziehungen.

performance_data $\langle E10 \rangle$
Diese Entität besitzt keine Beziehungen.
Eine Instanz der Entität $\langle E10 \rangle$ performance_data beschreibt genau einen Datensatz, der bei einer Messung des Mobilgerätes erstellt wird. networkoperator ist hierbei eine im Allgemeinen fünfstellige Zahl (zusammengesetzt aus MCC und MNC), die global eindeutig vergeben wird, currenttime den zu dem Zeitpunkt der Messung durchgeführten Zeitpunkt. Cid ist die Cell-ID des Sendemastes und lac der Local-Area-Code, in dem sich der Sendemast befindet.
flag bezeichnet den Zustand des Datensatzes, ob dieser schon an den WebService übermittelt wurde oder nicht. Da in der SQLite-Datenbank von Android keine Boolean-Werte möglich sind, handelt es sich hierbei um ein einzelnen Charakter, der nur den Wert ‘N‘ (für nein) oder ‘Y‘ (für ja) annehmen kann.

6.2 Server

Der Server verwendet eine Microsoft SQL Server-Datenbank. Ihr Aufbau wird in den folgenden Unterkapiteln näher erläutert.

6.2.1 Diagramm

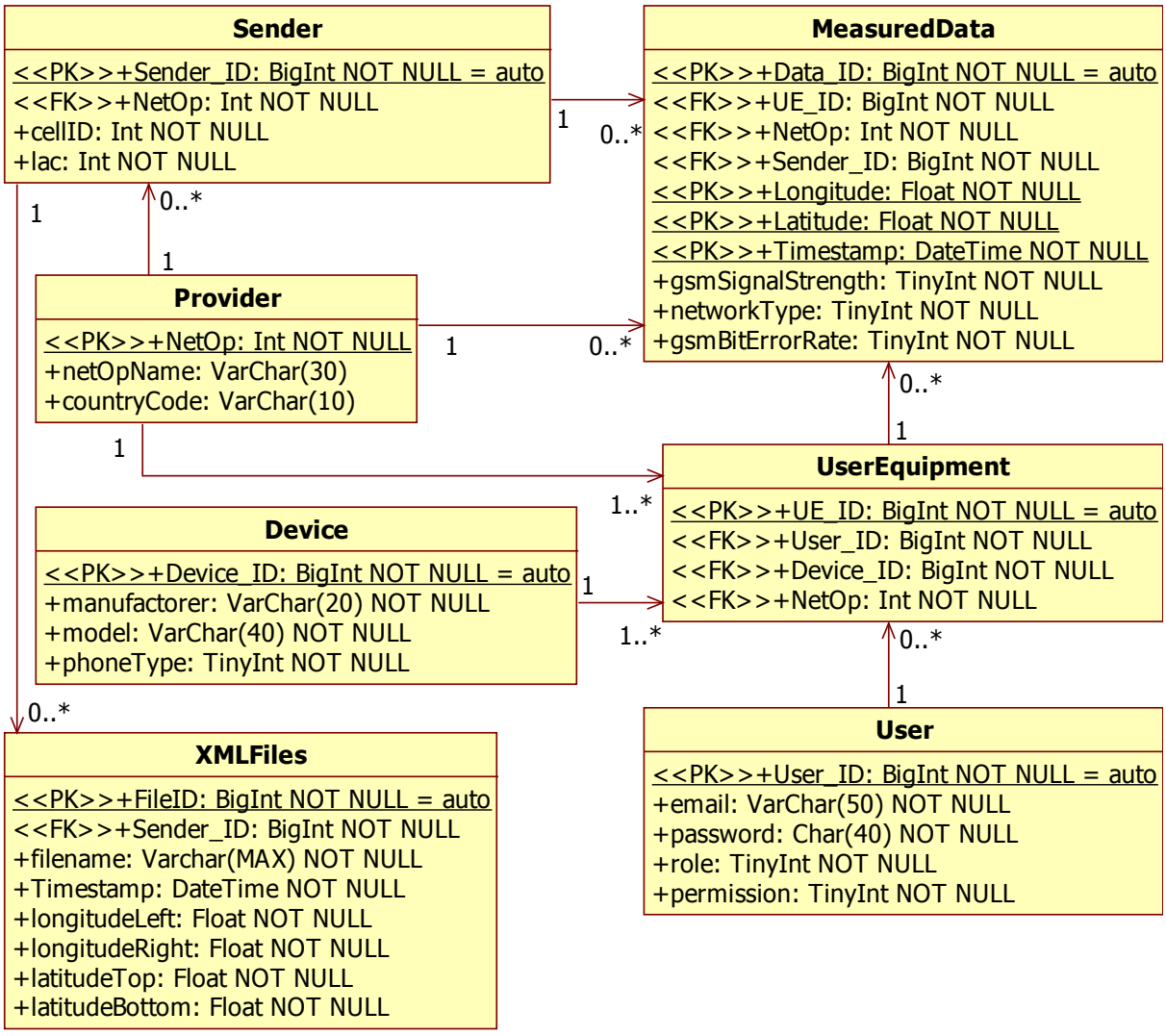

Abbildung 6.2: Klassendiagramm Server-Datenbank

Hinweise zum Diagramm in Abbildung 6.2:

- Im Diagramm sind die Beziehungen nicht benannt, um die Übersichtlichkeit zu erhöhen. Es existieren für sie keine besonderen Eigennamen. Sie werden sinngemäß anhand der Namen der zugehörigen Entitäten referenziert.
- PK = Primary Key, FK = Foreign Key, auto = wird automatisch mit Werten befüllt

6.2.2 Erläuterung

Die Datenbank ist so konzipiert, dass möglichst geringe Speicherplatzkosten für die potentiell größte Tabelle, MeasuredData, entstehen. Die Entität UserEquipment ist diesem Gedanken geschuldet, da durch sie die Zahl der (BigInt-)Fremdschlüssel um zwei reduziert werden kann.

Device $\langle E20 \rangle$

Beziehung	Kardinalität
Device-UserEquipment	1

Eine Instanz der Entität $\langle E20 \rangle$ Device beschreibt genau ein Modell eines Mobilgerätes. phoneType ist hierbei ein Zahlen-Code für die Mobilfunkfähigkeiten eines Modells (z.B. "0"für GSM-Geräte).

Provider $\langle E21 \rangle$

Beziehung	Kardinalität
Provider-UserEquipment	1
Provider-MeasuredData	1
Provider-Sender	1

Eine Instanz der Entität $\langle E21 \rangle$ Provider beschreibt genau einen Mobilfunknetzbetreiber. NetOp ist hierbei eine im Allgemeinen fünfstellige Zahl (zusammengesetzt aus MCC und MNC), die global eindeutig vergeben wird.

Sender $\langle E22 \rangle$

Beziehung	Kardinalität
Provider-Sender	0..*
Sender-MeasuredData	1
Sender-XMLFiles	1

Eine Instanz der Entität $\langle E22 \rangle$ Sender beschreibt einen Mobilfunkmaste. cellID ist hierbei die Kennung eines Masts und lac (Local Area Code) die Kennung des Bereichs, in welchem ein Mast

steht. Da jeder Mobilfunkbetreiber eigene Kennungen vergeben kann, muss hierbei der jeweilige Mobilfunknetzbetreiber mit referenziert werden.

User $\langle E23 \rangle$

Beziehung	Kardinalität
User-UserEquipment	1

Eine Instanz der Entität $\langle E23 \rangle$ User beschreibt einen Benutzer des WebServices. password wird verschlüsselt gespeichert. role ermöglicht es, die Berechtigungen eines Users anzupassen (User-, Forscher-, Adminzugriff). permission ermöglicht das gezielte Sperren und Freischalten von Nutzer, etwa um Betrug unterbinden zu können.

UserEquipment $\langle E24 \rangle$

Beziehung	Kardinalität
Device-UserEquipment	1..*
Provider-UserEquipment	1..*
User-UserEquipment	0..*
UserEquipment-MeasuredData	1

Eine Instanz der Entität $\langle E24 \rangle$ UserEquipment beschreibt genau ein spezielles Gerät eines Nutzers. Sollte ein Nutzer zwei Geräte mit demselben Mobilfunknetzbetreiber verwenden, werden diese als ein Gerät gesehen. Dies ist unumgänglich da von X-Map keine sensiblen Gerätedaten, wie z.B. die IMEI des Gerätes, gesammelt werden.

MeasuredData $\langle E25 \rangle$

Beziehung	Kardinalität
UserEquipment-MeasuredData	0..*
Provider-MeasuredData	0..*
Sender-MeasuredData	0..*

Eine Instanz der Entität $\langle E25 \rangle$ MeasuredData repräsentiert genau einen von der X-Map-Android-App gesammelten Datensatz. Der Mobilfunknetzbetreiber NetOp ist hier zusätzlich referenziert, um Roaming unabhängig vom Mobilfunknetzbetreiber des Heimatnetzes eines User erkennen zu können. gsmSignalStrength ist hierbei die eigentlich wichtigste Information, nämlich die Signalqualität in ASU. networkType ist ein Code für die jeweils verwendete Mobilfunktechnik, z.B. "13" für "LTE".

XMLFiles $\langle E26 \rangle$

Beziehung	Kardinalität
Sender-XMLFiles	0..*

Eine Instanz der Entität $\langle E26 \rangle$ XMLFiles repräsentiert eine XML-Datei, in welche Datensätze bezogen auf Mobilfunkmast und Zeit aus der Datenbank ausgelesen und gespeichert werden.

Die vier Werte zu Longitude und Latitude ermöglichen eine beschleunigte geografische Suche anhand des Bounding Box-Verfahrens.

7 Serverkonfiguration

Um die Serverkomponente von X-Map einsetzen zu können sind folgende softwareseitigen Voraussetzungen zu erfüllen:

Webserver Microsoft IIS mindestens auf Version 6.0

Datenbank Microsoft SQL Server mindestens auf Version 2005

In der Datenbank muss entsprechend Abschnitt 6.2 ein eine X-Map-Datenbank erstellt werden.

X-Map verwendet zur Anbindung von Android- und WebApp einen in C# geschriebenen WebService. Damit dieser ordnungsgemäß ausgeführt werden kann, ist es unabhängig von der restlichen Konfiguration des Servers notwendig, dass das .NET-Framework mindestens in der Version *ASP.NET 4.0.30319* installiert ist.

7.1 Konfiguration des IIS

Der IIS wird zunächst mit Default-Einstellungen eingerichtet. Darüber hinaus sind wie folgt in Abhängigkeit von der verwendeten IIS-Version diese Einstellungen zu machen:

Wird IIS-Version 6.0 oder 7.0 verwendet, so sind folgende Einstellungen zu machen:

- Der Webdienst bzw. die Webseite muss als Virtuelles Verzeichnis unter „Default Web Site“ eingerichtet werden.
- Die Einstellung „Execute permission“ ist auf „Scripts and Executables“ zu stellen.
- Die Einstellung „Application pool“ ist auf „DefaultAppPool“ zu stellen.

Wird IIS-Version 7.5 oder höher verwendet, so sind folgende Einstellungen zu machen:

- Der Webdienst bzw. die Webseite muss als eigene Anwendung auf eigenem Port bzw. Host laufen.
- Der Anwendungspool ist *ASP.NET v4.0 Classic*

8 Erfüllung der Kriterien

Folgendes Kapitel beschreibt den aktuellen Stand der Erfüllung der Kriterien aus dem Pflichtenheft.

8.1 Musskriterien

Es folgt die Beschreibung zu den Musskriterien des Pflichtenheftes.

8.1.1 App

Kriterien der Android-App:

- $\langle RM1 \rangle$ Aufzeichnung ortsabhängiger Daten des Mobilfunknetzes
  Dieses Kriterium wird erfüllt. Es werden dazu Methoden zum Auslesen der Telefon- und Verbindungsinformationen (aus den Klassen SignalStrength, CellLocation und TelephonyManager), sowie zum Bestimmen der aktuellen Position (aus der Klasse Location), verwendet, welche bereits in Android enthalten sind.
- $\langle RM2 \rangle$ Einfache Graphendarstellung der vom eigenen Gerät gesammelten Daten
  Dieses Kriterium wird erfüllt. Es wird ein einfacher Graph mit x- und y-Achse erstellt, der die Signalstärke auf die y-Achse aufträgt und die Zeit auf die x-Achse. Dazu wird ein externes Paket "`AndroidPlot`" verwendet, da in Android keine Graphendarstellung bereitgestellt wird. Durch eine Vererbung der in "`AndroidPlot`" bereitgestellten Klasse "`XYPlot`" wurden weitere Methoden hinzugefügt, mit denen zusätzliches Zoomen und Skalieren des Graphen möglich wird.
- $\langle RM3 \rangle$ Möglichkeit, die gesammelten Daten an den zugehörigen WebService zu übermitteln
  Dieses Kriterium wird erfüllt. Um die gesammelten Daten zu übermitteln, werden in gewissen Zeitabständen mittels des externen Paketes "`kSOAP 2`" die aktuellsten Messdaten an den WebService übertragen.

- $\langle RM4 \rangle$ Option, die Übermittlungsfrequenz und den maximalen Speicherbedarf zu variieren
  Dieses Kriterium wird erfüllt. Dem Benutzer wird die Wahl gelassen, in welchen Abständen die Applikation Messdaten an den WebService übertragen soll. Dazu kann er bis jetzt zwischen 2 Stunde, 5 Stunden und 10 Stunden wählen. Optional kann er auch eine Übertragung sofort durchführen lassen, wenn der Benutzer es wünscht.
  Die Speichergröße ist veränderbar. Dabei wird die Maximalgröße der Datenbank eingestellt. Einstellbare Speichermöglichkeiten sind 10 MB, 50 MB und 100 MB. Hierdurch kann der Speicherbedarf an die Wünsche des Nutzers angepasst werden.
- $\langle RM5 \rangle$ Lokales Speichern der aufgezeichneten Messdaten
  Dieses Kriterium wird durch eine auf dem Mobilgerät eingerichtete Datenbank erfüllt. Dazu wird das in Android bereitgestellte "`SQLite`"-Datenbankverwaltungssystem verwendet. Durch diese wird eine einfache Verwaltung der Daten ermöglicht.
- $\langle RM6 \rangle$ Auswahl der zu messenden und lokal zu speichernden Parameter
  Dieses Kriterium wird erfüllt. Die vom Nutzer ausgewählten Einstellungen werden mittels des (bereits in Android enthaltenen) PreferenceManagers in einer XML-Datei permanent gespeichert. Die gespeicherten Einstellungen sind mittels des Interfaces SharedPreferences innerhalb jeder Klasse der App sichtbar.
- $\langle RM7 \rangle$ Selektion verschiedener Datenschutzoptionen
  Aufgrund von Änderungen ist dieses Musskriterium nun mit $\langle RM6 \rangle$ verbunden. Dieses Kriterium gibt es nun nicht mehr alleinstehend, wird aber in $\langle RM6 \rangle$ dennoch erfüllt.
- $\langle RM8 \rangle$ Möglichkeit zur Erstellung eines Benutzerkontos für den WebService
  Dieses Kriterium wird erfüllt. Es ist möglich, dass der Benutzer sich mittels E-Mail-Adresse und Passwort am WebService registriert.
- $\langle RM9 \rangle$ Möglichkeit zum Login am WebService
  Dieses Kriterium wird erfüllt. Es ist möglich, dass der Benutzer sich mittels seiner E-Mail-Adresse und seinem gewählten Passwort am WebService anmeldet.

8.1.2 WebService

Kriterien des WebService:

- $\langle RM10 \rangle$ Registrierung von Mobilgeräten und Verwaltung der Anmeldedaten
  Dieses Kriterium wird erfüllt. Bei der Registrierung nimmt der WebService E-Mail-Adresse und Passwort als Nutzerdaten entgegen und speichert diese in der Datenbank nach einer Prüfung, ob die E-Mail-Adresse schon existiert. Bei jeder Anmeldung werden die Anmeldedaten mit den Daten aus der Datenbank verglichen und der App wird eine SessionID zur

Datenübertragung bzw. eine Fehlermeldung bei nicht erfolgreicher Anmeldung übergeben. Das Passwort wird verschlüsselt abgespeichert.

- $\langle RM11 \rangle$ Entgegennehmen der von den Mobilgeräten gesendeten Daten und Speicherung in einer MSSQL Datenbank
  Dieses Kriterium wird erfüllt. Der WebService nimmt ein Paket an Datensätzen von Mobilgeräten entgegen und speichert diese einzeln in der Datenbank.
- $\langle RM12 \rangle$ Grundlegende Möglichkeiten zur Auswertung und Visualisierung der vorhandenen Daten
  Dieses Kriterium ist bisher noch nicht erfüllt. Geplant ist, dass die Daten nach verschiedenen Kriterien gefiltert werden können, z.B. nach Provider oder Handymodell, und es dann Möglichkeiten gibt, die Messdaten als Tabelle oder als ein Overlay auf einer GoogleMaps-Karte (s. $\langle RC6 \rangle$) darzustellen.

8.2 Sollkriterien

In der Folge Kommentare zur Erfüllung der angestrebten aber nicht zwingend nötigen Kriterien:

8.2.1 App

Kriterien der Android-App:

- $\langle RS1 \rangle$ Energiesparende Ressourcennutzung
  Dieses Kriterium wird erfüllt. Der größte Ressourcenverbrauch der Anwendung fällt auf die Verwendung des GPS-Moduls zum Bestimmen der aktuellen Position. Der Nutzer hat die Möglichkeit, den Messintervall zwischen zwei Positionsbestimmungen in den Einstellungen selbst zu bestimmen. Zwischen den Messungen wird das GPS-Modul deaktiviert.
- $\langle RS2 \rangle$ Anonymisierung der Messdaten
  Dieses Kriterium wird erfüllt. Der Benutzer kann einstellen, ob seine Messdaten mit seinem Mobilgerät verbunden werden oder nicht. Ein Bezug zu den Benutzerdaten erfolgt nicht.
- $\langle RS3 \rangle$ Englische und deutsche Lokalisierung
  Dieses Kriterium wird durch die von Android bereitgestellte Lokalisierungsmethode erfüllt. Dabei wird die Lokalisierung an die eingestellte Systemsprache von Android abhängen. Englisch ist dazu die Standardspracheinstellung.
  Wenn das Mobilgerät auf deutsche Sprache eingestellt ist, wird die X-Map App ebenfalls auf Deutsch dargestellt. Bei allen anderen Spracheinstellungen wird die X-Map App auf Englisch angezeigt.

- $\langle RS4 \rangle$ Bei Überschreitung einer maximalen Antwortzeit ist der Nutzer darauf hinzuweisen
  Dieses Kriterium wird erfüllt. Der Benutzer wird bei einer zu langen Antwortzeit durch einen Ladebildschirm darauf hingewiesen, dass die X-Map App zum Arbeiten noch Zeit benötigt und noch nicht antworten kann.
- $\langle RS5 \rangle$ Bewegungsmuster dürfen nicht nachvollzogen werden können
  Dieses Kriterium wird erfüllt. Die Messdaten der App werden dazu ohne die dazugehörigen Zeitpunkte der Messungen an den WebService übertragen. Die Daten werden auf dem WebService nur mit dem Zeitpunkt der Übertragung markiert. Dadurch wird ein Bewegungsmuster nicht ersichtlich.
  Innerhalb der App kann der Benutzer sein Bewegungsmuster aber nachvollziehen. Der Zeitpunkt der jeweiligen Messung wird lokal gespeichert. Diese Daten bleiben aber lokal auf dem Mobilgerät und werden nicht weitergegeben.

  Dieses Soll-Kriterium wurde zu Beginn falsch interpretiert und bezieht sich darauf, dass das Bewegungsmuster auf dem WebService und damit durch Dritte nicht nachvollziehbar sein darf. Der Benutzer selbst kennt sein Bewegungsmuster und kann dieses somit auf seinem Mobilgerät nachvollziehen.
- $\langle RS6 \rangle$ Verhalten im Fehlerfall soll vom Nutzer konfigurierbar sein
  Dieses Kriterium wird erfüllt. Dem Nutzer ist es möglich, das Verhalten der App bei fehlgeschlagener Übertragung im Menü "`Einstellungen`" zu konfigurieren.

8.2.2 WebService

Kriterien des WebService:

- $\langle RS7 \rangle$ Erweiterbarkeit der Kapazität ohne Softwareaktualisierung
  Dieses Kriterium wird weitgehend umgesetzt. Wird auf eine Datenbank mit höherer Kapazität gewechselt, müsste aktuell gegebenenfalls die Adresse der Datenbank im Quellcode des WebService sowie das Mapping der Entitäten der Datenbank auf die Klassen des Datenbankzugangs aktualisiert werden.

8.3 Kannkriterien

Es folgen Kommentare zu den wünschenswerten aber weniger wichtigen Kriterien:

8.3.1 App

Kriterien der Android-App:

- $\langle RC1 \rangle$ Fehlerprotokoll / Statusbericht
  Dieses Kriterium wird nicht erfüllt. Aus zeitlichen Gründen wird auf eine Implementierung dieses Kriteriums verzichtet.
- $\langle RC2 \rangle$ Fortschrittsanzeigen bei Übertragungsverzögerungen
  Dieses Kriterium wird nicht erfüllt. Aus zeitlichen Gründen wird auf eine Implementierung dieses Kriteriums verzichtet.
- $\langle RC3 \rangle$ Nachtmodus (mit einstellbaren Zeiten) / Sparmodus / Zugmodus
  Der Sparmodus wird durch das Soll-Kriterium $\langle RS1 \rangle$ erfüllt. Der Benutzer kann die Dauer zwischen den Messungen einstellen.
  Da dem Benutzer genügend Messintervalle zur Auswahl stehen, wird auf einen separaten Zug- und Nachtmodus verzichtet.
- $\langle RC4 \rangle$ Overlay für eine visuelle Auswertung der Empfangsqualität entlang der eigenen Spur mittels eines Kartendienstes
  Dieses Kriterium wird erfüllt. Der Benutzer kann sich auf dem Kartendienst "`Google Maps`" seine Empfangsqualität, durch colorierte, variable, geometrische Strukturen, anzeigen lassen.
- $\langle RC5 \rangle$ Keine sofortige Interaktion des Nutzers bei Fehlern in Hintergrundfunktionen nötig
  Dieses Kriterium wird erfüllt. Wenn Hintergrundfunktionen ausfallen, wird der Benutzer nicht benachrichtigt und muss nicht eingreifen.

8.3.2 WebService

Kriterien des WebService:

- $\langle RC6 \rangle$ Overlay für eine visuelle Auswertung räumlicher Versorgungseigenschaften mittels eines Kartendienstes
  Dieses Kriterium ist bisher noch nicht erfüllt. Die gesammelten Messdaten könnten von der Web-Applikation als ein Overlay über einer Karte dargestellt werden.

- $\langle RC7 \rangle$ Zusätzliche Möglichkeit zur Verwendung einer MySQL Datenbank
  Dieses Kriterium kann erfüllt werden. Da LINQ to SQL für die Datenbankverbindung benutzt wird, muss nur die Datenbank-Connection und das Mapping der Entitäten der Datenbank auf die Klassen des Datenbankzugriffs aktualisiert werden.
- $\langle RC8 \rangle$ E-Mail-Adressen verifizieren
  Dieses Kriterium ist bisher noch nicht erfüllt. Dabei soll nach der Registrierung am WebService eine E-Mail vom Server verschickt werden, die einen Link o.ä. enthält, wodurch der Nutzer seine E-Mail-Adresse verifizieren kann.

8.4 Abgrenzungskriterien

Nachfolgend Kommentare zu den explizit nicht zu erfüllenden Kriterien:

- $\langle RW1 \rangle$ Normalen Benutzern die Auswertung der Daten auf dem Server ermöglichen
  Geplant ist die Unterscheidung von Gruppen verschiedener Benutzer, z.B. „Forscher“ und „App-User“. Der Zugriff auf die Möglichkeiten der Web-Applikation kann so Gruppenweise geregelt werden.
- $\langle RW2 \rangle$ Veröffentlichung der App im Google Play Store
  Stattdessen soll die App zunächst nur direkt in Form ihrer Application Package-Datei weitergegeben werden.

9 Glossar

.NET Framework:
Bezeichnet eine von Microsoft entwickelte Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. .NET besteht aus einer Laufzeitumgebung (Common Language Runtime), in der die Programme ausgeführt werden, sowie einer Sammlung von Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (Services).

Android:
„Android“ ist der Name eines von Google entwickelten Betriebssystems für Mobilgeräte.

Android-App:
Kurzform für Android-Applikation

ANDROID_ID:
Die ANDROID_ID ist eine eindeutige 64-bit Folge, welcher bei der Erstaktivierung eins Android-Gerätes generiert wird und potentiell auch nach einem Reset erhalten bleibt.

AndroidPlot:
AndroidPlot ist ein von einer kleinen Gruppe von Mitarbeitern in und um Silicon Valley erstelltes externes Paket, dass eine Graphendarstellung in Android ermöglicht. Dabei ist AndroidPlot für jede öffentliche, private und gewerbliche Nutzung zugägnlich.

Android Application Package File (.apk)
Komprimiertes Java Archiv, welches speziell zum Packen von Android-Applikationen genutzt wird.

Android SDK:
Software Development Kit (SDK), die die Java-Programmiersprache erweitert, um eine Programmierung für Androidgeräte zu ermöglichen.

Eclipse:
Ein quelloffenes Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Ursprünglich wurde Eclipse als integrierte Entwicklungsumgebung (IDE) für die Programmiersprache Java genutzt.

Overlay:
Overlays sind Objekte auf der Karte, die an Breitengrad/Längengradkoordinaten gebunden sind und mit der Karte zusammen bewegt werden, wenn Sie diese verschieben oder zoomen. Overlays

stehen für Objekte, die Sie zur Karte hinzufügen, um Punkte, Linien, Bereiche oder Sammlungen von Objekten anzugeben.

Google Play Store:
Ein E-Commerce-System zum Vertrieb von Android-Apps und deren Verwaltung, sowie zum Erwerb von Büchern, Musik und Filmen in digitaler Form.

GPS:
Das Global Positioning System (GPS) ist ein ursprünglich militärisches System zur weltweiten satellitengestützen Positionsbestimmung.

IIS:
s. Microsoft Internet Information Services

IMEI:
Die International Mobile Station Equipment Identity (IMEI) ist eine eindeutige 15-stellige Seriennummer, anhand derer jedes GSM- oder UMTS-Endgerät theoretisch eindeutig identifiziert werden kann.

LINQ:
Abkürzung für *Language Integrated Query*. Ein in C# integriertes Konzept zum Zugriff auf Daten. SQL-artige Querys können verwendet werden um auf Daten verschiedenster Art zuzugreifen.

LINQ to SQL:
LINQ für den Zugriff auf SQL-basierte Datenbanken.

LINQ to XML:
LINQ für den strukturierten Zugriff auf XML-Dateien.

MCC:
Der Mobile Country Code (MCC) ist eine dreistellige Zahl, welche jedem Land der Erde von der International Telecommunication Union zugeteilt wurde.

Microsoft Internet Information Services:
Eine von Microsoft entwickelte und vertriebene Software zum Hosten von Dateien und Dokumenten in Netzwerken. Optional auf quasi allen Desktop-Betriebssystemen von Microsoft seit Windows 2000 installierbar.

Microsoft Visual Studio 2010:
Eine von dem Unternehmen Microsoft angebotene, integrierte Entwicklungsumgebung für verschiedene Programmiersprachen.

MNC:
Der Mobile Network Code (MNC) ist eine im Allgemeinen zweistellige Zahl, welche jedem Mobilfunktnetzbetreiber eines Landes von einer Behörde des jeweiligen Landes zugeteilt wird.

SOAP:
Das *Simple Object Access Protocol* ist ein Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP stützt sich auf XML zur Repräsentation der Daten.

Windows Server 2008 R2:
Ein speziell für Server optimiertes Betriebssystem von Microsoft. Wurde parallel mit dem bekannten Betriebssystem Windows 7 entwickelt und kam im Oktober 2009 auf den Markt.