HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

novasina
The Art of Precision Measurement

# NovaMobile

## Studien- und Bachelorarbeit HS19/20

Abteilung Informatik

Hochschule für Technik Rapperswil

**Version:** 1.0


**Autor (BA):** Luca Tavernini

**Autor (SA):** Christoph Streiff

Betreuer: Daniel Politze

Projektpartner: Novasina AG Lachen

Experte: Ramon Schildknecht

Gegenleser: Andreas Steffen



**Erstellt am:** 17. September 2019

**Letzte Änderung am:** 09. Januar 2020

# Abstract

**Problem**

Die Novasina AG ist eine KMU mit Sitz in Lachen SZ, welche sich auf Messtechnik und Sensorik spezialisiert hat. Ihr primäres Expertisengebiet sind Feuchtigkeits- und Temperatursensoren, die vor allem in Lebensmittel- und Medizintechnik zum Einsatz kommen.

Da diese Sensoren nicht nur stationär installiert werden, sondern zum Beispiel für Audits auch mobil einsetzbar sein müssen, besteht Bedarf nach einem handlichen Gerät, das mit den Sensoren kompatibel ist und deren Messresultate auslesen, anzeigen sowie speichern kann.

Ein solches Gerät, auf proprietärer Hardware basierend, existiert mit dem Novasina ClimMate-Gerät bereits. Die Novasina AG wünscht sich jedoch aus Marketing- und Kostengründen eine Mobile-App, welche dieselbe Funktionalität für handelsübliche Smartphones anbietet.

**Ziel**

Ziel dieser Arbeit ist die Erstellung einer solchen App, die unter Android betriebsfähig ist. Kernpunkte sind dabei:

- Anschliessen eines Novasina-Sensors mit USB-Kabel
- Auslesen der gemessenen Klimadaten
- Speichern dieser Daten auf dem Smartphone
- Tabellarisches und grafisches Darstellen der Messwerte
- Klimarechner für die Berechnung fiktiver Klimawerte

**Wesentliche Ergebnisse**

Finales Resultat der Arbeit ist eine Android-Applikation, die das Xamarin-C#-Framework nutzt und auf handelsüblichen Smartphones (Android-Version 8.0 Oreo oder neuer) einsetzbar ist. Sie ermöglicht es, die nSens-Fühler der Novasina AG zur Messung von Feuchtigkeit und Temperatur mittels USB-Kabel an das Smartphone anzuschliessen und deren Messdaten auszulesen, Momentaufnahmen oder längere Logs abzuspeichern sowie zu exportieren. Darüber hinaus bietet sie einen Klimarechner an.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Aufgabenstellung

novasina
The Art of Precision Measurement

## *Entwicklungs-Projekt 2: App für Sensordaten und Klimarechner, Bluetooth Anbindung der Fühler*

### Einführung

Novasina ist ein Schweizer KMU, spezialisiert auf hochwertige Feuchtesensoren zur Klimaüberwachung und Kontrolle. Die Grundlage ist ein selbst entwickelter Fühler mit elektrolytisch-resistivem Messprinzip. Dies erlaubt hohe Genauigkeit und langzeitstabile Messung. Daneben steht auch ein herkömmliches Messprinzip zur Verfügung als kostenoptimierte Variante.

Diese nSens Fühler sind ein digitales System, das heisst der Messwert wird mit den direkt auf dem Fühler gespeicherten Kennlinien und Kalibrationsdaten auf den Endwert relative Luftfeuchte und Temperatur umgerechnet. Der nSens Fühler verwendet zur Kommunikation der Messwerte den eigens entwickelten «nBus» der Novasina.

**Die nSens Fühler können heute bereits per USB Kabel direkt an einen PC angeschlossen und ausgelesen werden.**

### Ziel:

Dasselbe USB Kabel (mit Hostadapter) mit nSens Fühler soll an ein Smartphone anschliessbar sein.
Grundfunktion der App:

- Auslesen der Fühlerdaten (Temp, rel. Feuchte)
- Klimarechner
- Datenspeicher (Logger) auf Smartphone mit anpassbaren Einstellungen.
- Datenexport CSV Dateien,
- Nice to have: Grafische Darstellung der Daten (Trendlinien, auswahl Datumsbereich, einfache statistische Angabe wie min/max, average)

### Sekundär

Bluetooth Adapter für nSens Fühler.
Ein Bluetooth Module steckbar für den nSens Fühler (Batteriebetrieben, ev USB recharge).
Dadurch kabellose Verbindung zum nSens App.

**Kontakt:**
Philippe Trösch, Produkt Manager, +41 55 642 6769
philippe.troesch@novasina.ch



Novasina AG, Neuheimstrasse 12, 8853 Lachen, Switzerland
Telephone +41 55 642 67 67 Fax +41 55 642 67 70, E-Mail: climate@novasina.ch, www.novasina.com

# 2 Technischer Bericht

## 2.1 Einleitung

**Problemstellung, Vision**

Die Novasina AG ist ein in Lachen SZ ansässiges KMU, welches sich auf Präzisionsmess- und Sensortechnik spezialisiert hat und in dieser Disziplin zu den weltweit führenden Unternehmen gehört. Sie stellt hauptsächlich Sensoren und Sensorzubehör wie Kalibrierungsgeräte, Mess-Logger und Verbindungen für Sensorik in den Bereichen Klimadaten- und Wasseraktivitätsmessung her. Zu ihren verbreitetsten Produkten gehört insbesondere die nSens-Reihe [1].

Dabei handelt es sich um hochwertige Temperatur- und Feuchtigkeitssensoren, die vor allem in der Lebensmittel- und Medizintechnologie Anwendung finden. Beispielsweise werden sie gebraucht, um festzustellen, ob eine bestimmte Zutat bei der Lagerung nicht zu feucht war oder um die Klimabedingungen bei einem Herstellungsprozess im Rahmen eines Audits zu prüfen. Die primären Messwerte, die ein solcher Sensor liefert, sind relative Luftfeuchtigkeit und Temperatur. Aus diesen zwei Messwerten lassen sich danach weitere, sekundäre Werte, wie zum Beispiel die Taupunkttemperatur berechnen.


**Abbildung 1: nSens-HT-Sensor [1]**

Die Sensoren der Novasina AG können über einen proprietären Bus, dem sogenannten nBus, angesprochen werden, um ihre Daten auszulesen. Dazu existiert eine dreipolige Kabelsteckverbindung, über welche analog kommuniziert werden kann. Für die digitale Kommunikation gibt es den nBus-Adapter, mit welchem der Sensor an eine USB-Buchse angeschlossen werden kann.

Da diese Sensoren allerdings nicht immer fest an einem Ort installiert werden, sondern auch für regelmässige Kontrollen oder Audits tragbar sein müssen, bietet die Novasina AG ein portables, akkubetriebenes Handgerät mit dem Namen ClimMate [2] an. Dieses Gerät verfügt über eine Steckbuchse für einen nSens-Sensor und ermöglichtes, Klimadaten direkt abzulesen und im Gerätespeicher abzulegen, sie später wieder einzusehen und grafisch darzustellen. Kurz: ClimMate deckt die Anwendungsfälle eines Auditors sehr gut ab.

Das ClimMate-Handgerät ist allerdings nicht immer unproblematisch. Einerseits ist es mit gewissen Kosten verbunden, andererseits hat es einige Bedienbarkeitsprobleme und ist schlaganfällig (siehe Kapitel 2.2.1). Die Novasina AG wünscht sich deshalb einen zuverlässigen Ersatz in Form einer App.

**Ziele**

Ziel dieser Arbeit ist das Erstellen einer App für Mobilgeräte, welche das ClimMate-Handgerät der Novasina AG funktional ersetzen kann. Ein USB-Kabel mit nBus-Adapter soll an ein handelsübliches Smartphone angesteckt werden können, welches dann die Funktion des ClimMate-Gerätes emuliert und in der Praxis seinen Platz einnehmen kann. Zu der Funktionalität, die in der App zu diesem Zweck vorhanden sein muss, gehört insbesondere:

- das Auslesen der Fühlerdaten (Temperatur und relative Feuchtigkeit)
- ein Klimarechner, der sekundäre Klimadaten aus Messdaten errechnen kann
- das Abspeichern (Loggen) von Messdaten im Gerätespeicher
- der Export von gespeicherten Messdaten als CSV-Datei

Optionale Ziele, die nach dem Erreichen der obigen, wichtigsten Ziele ins Auge gefasst werden können, wären dazu:

- die tabellarische und grafische Darstellung der gemessenen Daten
- das Anbieten einer Bluetooth-Schnittstelle für den nSens-Fühler, über welche eine drahtlose Verbindung möglich ist (zum Zeitpunkt des Verfassens dieser Arbeit existiert allerdings noch kein solcher nBus-Bluetooth-Anschluss)

**Rahmenbedingungen**

Die Rahmenbedingungen dieser Arbeit wurden aus zweierlei Quellen bestimmt. Einerseits gelten die bestehenden Richtlinien der HSR zu Studien- und Bachelorarbeiten [3]. Andererseits kamen zusätzliche Auflagen von der Novasina AG hinzu, da das Projekt in ihrem Auftrag ausgearbeitet wird. Zu letzteren gehören insbesondere Eingrenzungen des Projekt-Scopes und Technologieempfehlungen:

- Gewisse Einschränkungen und Empfehlungen bezüglich des Betriebssystems (Android), Sprache (C#), Technologien und Frameworks wurden herausgegeben. Zu Details siehe Kapitel 3.3.
- Die Applikation muss beim Release nur mit den nSens-Fühlern der HT-Reihe umgehen können. Dabei handelt es sich um Sensoren, die Feuchtigkeit und Temperatur messen können. Erweiterungen sind später allerdings möglich und müssen eingeplant werden.
- Die Applikation muss beim Release nur Verbindungen über die USB-Schnittstelle des Mobilgerätes akzeptieren können. Ein nSens-Bluetooth-Adapter ist zwar in Planung, jedoch noch nicht genug fortgeschritten, um schon mit in das Projekt einbezogen werden zu können. Die Möglichkeit, neue Verbindungsmedien zu akzeptieren, muss allerdings eingeplant werden, um das Produkt mit zukünftigen Entwicklungen kompatibel zu machen.

## 2.2 Stand der Technik – Was gibt es schon?

### 2.2.1 Bestehende Lösungsansätze und Normen

**Kurzbeschreibung** Aufgrund der Tatsache, dass der Markt für Präzisionsmesstechnik in dem Ausmass, wie die Novasina AG sie betreibt, relativ klein ist, gibt es nicht allzu viele bestehende Lösungen. Dies gilt vorallem für die proprietäre nSens-Technologie. Der primäre bestehende Lösungsansatz ist das am Anfang erwähnte ClimMate-Handgerät, das heute von der Novasina AG hergestellt und vertrieben wird.

Es handelt sich dabei um ein akkubetriebenes, portables Gerät mit einem Touchscreen, einer Eingabefläche und einem nBus-Anschluss, an den ein nSens-Klimasensor angesteckt werden kann.


**Abbildung 2: ClimMate-Handgerät [2]**

Das Gerät kann dann auf den so verbundenen Sensor zugreifen und dessen gemessene Daten (üblicherweise Temperatur und Feuchtigkeit, weitere Sensortypen sind aber ebenfalls möglich) sowie weitere, aus diesen Primärmessgrössen errechnete Werte anzeigen, beispielsweise den Taupunkt oder die Wasseraktivität.

Darüber hinaus kann das ClimMate-Gerät Logs aus diesen Daten erstellen und sie im Gerätespeicher ablegen sowie diese Logdaten grafisch und tabellarisch anzeigen. Dies ist insbesondere für Auditoren von Interesse, die an verschiedenen Orten Messwerte entnehmen müssen, um zu prüfen ob die Bedingungen der Norm entsprechen.

Als weiterer Lösungsansatz, der diese Arbeit aber eher tangential betrifft, existiert ein Programm der Novasina AG für Windows, das eine ähnliche Funktionalität wie das ClimMate-Gerät bietet. Es kann zusätzlich auch für die Kalibrierung der Sensoren verwendet werden. Eine solche Funktion für die App ist aber nicht im Scope dieses Projektes.

Schlussendlich existiert auch ein Multisensor-Messsystem (QuantaDat [4]), an das sich bis zu vier Sensoren anschliessen lassen. Dieses ist jedoch zur festen Installation an einem Ort gedacht.


**Abbildung 3: QuantaDat mit zwei nSens-Sensoren [4]**

**Defizite**

Bei der Analyse des ClimMate-Handgerätes, das uns von der Novasina AG im Rahmen dieser Arbeit als Beispiel zur Verfügung gestellt wurde, fielen einige Defizite insbesondere bezüglich der Usability auf:

- Das grafische User Interface ist gelegentlich kompliziert und eher unintuitiv. Einige Menüpunkte werden erst nach Konsultation des Benutzerhandbuchs klar. Die Applikation ist an ein paar Stellen zu komplex, um selbsterklärend zu sein. Ein neuer Nutzer wird also eine gewisse Zeit brauchen, bis er sich daran gewöhnt hat.
- Der Input ans Gerät erfolgt über eine Mischung aus Touchscreen-Eingabe und Touch-Eingabetasten. Oft ist unklar, in welcher Situation welche Eingabemethode gewählt werden muss, was zu fehlerhaften oder wirkungslosen Inputs führt.
- Das Gerät ist schlaganfällig. Ein kurzer, aber bestimmter Stoss gegen eine Oberfläche führt dazu, dass es sich abschaltet. Dabei können einerseits Daten oder Eingaben verloren gehen, andererseits muss es dann neu gestartet werden, was ca. eine halbe Minute dauert. Für ein tragbares Gerät, welches für den Einsatz vor Ort gedacht ist, stellt dieser Umstand einen nicht unerheblichen Nachteil dar.
- Das Gerät beginnt mit dem Messprozess bevor der Benutzer irgendeinen Input tätigt. Minimum, Durchschnitt und Maximum werden angezeigt, obwohl der Benutzer die Messung noch gar nicht gestartet hat. Dies kann gegebenenfalls zu Verwirrungen führen.

- Das Gerät hat den Nachteil, an die durch die Hardware vorgegebene Anschlussmethode gebunden zu sein. Dadurch, dass nur eine einzige direkte nSens-Sensorsteckverbindung am Gerät vorhanden ist, kann jeweils immer nur ein Sensor gleichzeitig angeschlossen werden. Dies verunmöglicht es beispielsweise, die Messwerte zweier Sensoren direkt zu vergleichen.

Schlussendlich gibt es ein kostentechnisches Problem: Das Gerät wird üblicherweise in einem Boxset bezogen und ist dann mit einem Preis von mehreren Tausend Franken verbunden. Eine Mobile-App hingegen könnte dem Kunden kostengünstig oder sogar gratis angeboten werden. Dies würde das Produkt wettbewerbsfähiger machen.

Das QuantaDat-Gerät eignet sich für die Anwendung als tragbares Gerät eher wenig, da es zur festen Installation vorgesehen ist, weswegen es hier nur am Rande erwähnt wird. Es unterstützt jedoch im Gegensatz zum ClimMate-Gerät bis zu vier Sensoren gleichzeitig, was eine Eigenschaft ist, welche sich die Novasina AG ebenfalls für ihr mobiles Gerät wünschen würde.

Die bestehende Windows-Applikation teilt die Eigenschaft des QuantaDat – ein ClimMate-Gerät oder ein Smartphone sind wesentlich einfacher unterwegs verwendbar als ein Laptop, worunter die Usability leidet. Es handelt sich also ebenfalls nicht um eine valide Alternative.

### 2.2.2 Bewertung

**Kriterien**

Ausgehend vom primären Anwendungsbereich des ClimMate-Geräts, welcher sich durch den Bedarf an Portabilität und Flexibilität auszeichnet, sowie der Zielgruppe stechen zwei wichtige Bewertungskriterien hervor.

- Es muss eine gute Usability gegeben sein, damit auch nicht technologieaffine oder unerfahrene Benutzer das Gerät ohne erheblichen Lern- oder Logistikaufwand anwenden können.
- Das Produkt soll genauso stand-alone sein wie das ClimMate. Es darf nicht auf sperrige Komponenten oder eine Verbindung zu einem Computer angewiesen sein, sondern muss frei herumtragbar sein und sich unabhängig von anderen Geräten, ausser den gerade verbundenen Sensoren, einsetzen lassen.

**Diskussion**

Aus den oben erwähnten Kriterien, der Anforderungsanalyse mit den Spezialisten der Novasina AG und den bestehenden Lösungen kristallisiert sich heraus, dass die ideale Lösung für die Novasina AG die nächste Iteration des ClimMate-Prinzips ist.

Beim Produkt, das die Novasina AG sich wünscht, handelt es sich um eine App, welche dieselbe Grundfunktionalität wie das bestehende Handgerät anbietet. Allerdings soll diese mehr Komfort und Usability für den Anwender bieten und durch die Entkopplung von proprietärer Hardware vielseitiger einsetzbar sein. Ziel dieser Arbeit ist also nicht, das Rad neu zu erfinden, sondern die bestehenden Prinzipien zu analysieren, sie auf eine Mobile-Applikation zu portieren und wo möglich zu verbessern.

Bedingt durch die Tatsache, dass eine Mobile-App weniger an die durch die Hardware gegebenen Möglichkeiten gebunden ist, kann von Anfang an auf die Erweiterbarkeit der Applikation Wert gelegt werden, sowohl auf dem Weg zum Release als auch im Hinblick auf etwaige Technologien, die die Novasina AG erst in Zukunft entwickeln wird. Dazu gehören beispielsweise neue Sensortypen wie ein CO2-Sensor oder neue Anschlussmedien, wie der bereits erwähnte Bluetooth-Adapter.

## 2.3 Vorgehen

**Analyse**

Gemäss den oben definierten Kriterien wurde zu Beginn des Projektes gemeinsam mit Experten der Novasina AG in einer Reihe von Meetings erarbeitet, welche Eigenschaften und Features das ClimMate-Handgerät in der Praxis so wertvoll machen und deshalb in die Mobile-App übernommen werden müssen. Anderseits wurde auch diskutiert, wo die grössten Schwächen des ClimMate-Geräts liegen, die eventuell zu verbessern wären und welche Features vernachlässigt oder sogar bewusst weggelassen werden können.

**MVP-Definition**

Durch diese Analyse war es bereits in den ersten zwei Wochen der Elaborationsphase möglich ein grobes Modell unseres Zielproduktes zu synthetisieren, welches dann iterativ weiter verfeinert wurde, bis es sich mit den Anforderungen der Novasina AG gut deckte. Dazu wurden insbesondere viele GUI-Konzeptzeichnungen und Wireframes verwendet, um den Experten der Novasina AG zu versinnbildlichen, wie die erwünschten Features aussehen könnten. Aus dieser Lösung wurde ein MVP (Minimum Viable Product) definiert, welches der Novasina AG so kommuniziert wurde und auf dessen Vervollständigung dieses Projekt hinarbeitete.

**Feedback**

Während der Elaborationsphase wurde einmal pro Sprint mit der Novasina AG Rücksprache gehalten, um sicher zu gehen, dass die richtigen Features mit der korrekten Priorisierung geplant wurden.

Sobald in der Konstruktionsphase ein lauffähiger Prototyp mit neuen Features verfügbar war, wurde die Novasina AG darüber informiert und ihr eine ausführbare Version davon zur Evaluation zugestellt. Bedingt durch die Tatsache, dass Features jeweils als Gesamtpaket von Funktionen gemäss Use Cases fertiggestellt werden mussten, konnte allerdings nur alle paar Wochen ein neuer Prototyp herausgegeben werden.

Durch ein agiles Vorgehen während der Entwicklung war es dennoch möglich, die Priorisierung der verschiedenen Features sowie deren Umsetzung laufend im Blick zu behalten und bei einem Änderungswunsch entsprechend anzupassen.

## 2.4 Eigener Lösungsansatz

**Minimum Viable Product**

Um sicherzustellen, dass das Produkt, welches im Rahmen dieses Projektes entwickelt wurde, sich mit den Anforderungen der Novasina AG deckt, wurde in der Elaborationsphase eine umfassende Anforderungsanalyse vorgenommen. Primär bestand diese aus einer Analyse des ClimMate-Gerätes und dessen Stärken und Schwächen in Zusammenarbeit mit Spezialisten der Novasina AG. So wurde erarbeitet, welche Features die App übernehmen musste, was verändert werden sollte und was ganz wegfallen durfte.

Durch diese Anforderungsanalyse entstanden Use Cases (siehe Kapitel 3.1.2), nichtfunktionale Anforderungen (siehe Kapitel 3.1.3) und UI-Konzepte (siehe Kapitel 3.3.7).

Aus diesen Artefakten wurde schlussendlich ein Minimum Viable Product definiert, also ein Produkt mit dem minimalen Funktionsumfang, welcher vorhanden sein muss, um es in der Praxis anwenden zu können. Für dieses Projekt umfasst das MVP die folgenden Punkte:

- Die nSens-Fühler können via USB-Kabel angeschlossen und ausgelesen werden. Idealerweise sind mehrere Verbindungen gleichzeitig möglich.
- Die App bietet drei Hauptfunktionsgruppen: Das Aufnehmen von Messungen, einen Klimarechner und ein Archiv.
- Im Messen-Ansichtsbereich ist es möglich, alle gerade verbundenen Sensoren einzusehen und sich einen raschen Überblick über ihre Live-Messdaten zu verschaffen.
- Es existieren zwei Messmodi, Schnappschuss-Aufnahmen und längerfristiges Loggen. Beide Modi sind für alle Sensoren aus der Messen-Ansicht zugreifbar.
- In beiden Aufnahmen-Ansichten (Schnappschuss und Logger) ist eine Auswahl von zu erfassenden Messgrössen möglich. Die so selektierten Grössen werden dann abgespeichert, sobald die Messung abgeschlossen ist.
- Das Messintervall für den Logger ist individuell anpassbar.
- Der Klimarechner bietet eine Auswahl an zu berechnenden Grössen an. Wird dort eine Grösse selektiert, können die dafür benötigten Werte eingegeben werden, um das Resultat zu erhalten.
- Die Archivansicht ermöglicht einen Überblick über alle gemachten Messungen und erlaubt es auch, diese wieder zu löschen.
- In der Archivansicht kann die Detailansicht einer Messung geöffnet werden, um genauere Informationen tabellarisch darzustellen.
- Gespeicherte Messungen können mit einem Label versehen werden, welche ihre Zuordnung vereinfachen (z.B. «Messung Experiment #42»).
- Gespeicherte Messungen können als CSV (Comma Separated Values) exportiert werden.
- Sensoren können Labels erhalten, welche es vereinfachen, den Sensor zuzuordnen (z.B. «Reinraum 2»).
- Der Referenzdruck für Messungen kann individuell angepasst werden.
- Als Sprachpakete sind Englisch und Deutsch verfügbar.

Dazu kamen eine Reihe von Features, die nicht als obligatorisch, sondern als «nice to have» definiert wurden und somit eine niedrigere Priorität erhielten:

- Das Intervall, in dem Logger-Resultate gespeichert werden, ist individuell anpassbar.
- Die gesamte Datenbank kann zu Backupzwecken exportiert/importiert werden.
- Messergebnisse können in verschiedenen Einheiten dargestellt werden (z.B. Celsius und Kelvin für Temperatur).
- Die Messdaten bei Messreihen können grafisch dargestellt werden.

**Technologien**

Bereits im ersten Drittel der Elaboration-Phase war klar, dass die Implementation rein unter Android erfolgen sollte und iOS oder andere mobile Betriebssysteme kein Thema sein würden, auch in Zukunft nicht. Die Begründung für diese Entscheidung ist in Kapitel 3.3.1 genauer erläutert.

Auf Wunsch der Informatikabteilung der Novasina AG fiel die Wahl auf die Nutzung des Microsofts Xamarin-Frameworks, welches die C#-Programmiersprache unter Android verfügbar macht. Weitere Abklärungen zu möglichen Frameworks sind im Kapitel 3.3.2 zu finden.

**Grundlegender Aufbau**

Basierend auf den ausgewählten Technologien und den uns auferlegten Rahmenbedingungen wurden verschiedene Komponenten des Lösungskonzeptes identifiziert und definiert, welche zur Erfüllung der im MVP definierten Features notwendig sind. Die Applikation lässt sich nach diesen Aspekten in sieben solche grundlegenden Bausteine unterteilen:

- Das grafische User-Interface (GUI), über welches der Nutzer mit der Applikation interagiert.
- Ein Dienst, der alle angeschlossenen Sensoren verwaltet, ohne sich direkt auf die Verbindungsmedien zu stützen, und diese Sensoren zur Abfrage anbietet.
- Pro Verbindungsmedium eine Komponente, die das Medium überwacht und dem verwaltenden Dienst meldet, wenn ein neuer Sensor angeschlossen oder ein verbundener Sensor wieder getrennt wird.
- Pro Verbindungsmedium eine Komponente, welche auf dem fraglichen Medium kommunizieren kann (z.B. via USB-Treiber) und von der eine Instanz jeweils die Verantwortung der Kommunikation mit einem einzelnen Sensor übernimmt. Dazu muss das Simple-Protokoll der Novasina AG verwendet werden, welches diese Sensoren ansteuern kann.
- Ein Logger-Dienst, der im Hintergrund laufen und längerfristig Messwerte liefern und speichern kann, auch wenn die Applikation minimiert wurde.
- Eine Datenbank, in welche Messresultate und bekannte Sensoren abgelegt werden können.
- Eine Reihe von Utility-Modulen, beispielsweise zur Serialisierung von Daten oder zur Erstellung von CSV-Files.

**Erweiterbarkeit**

Die Erweiterbarkeit war eines der Elemente, auf das bei der Planung am meisten Wert gelegt wurde. Da von Anfang an klar war, dass das Produkt nach Abschluss der Arbeit nicht als «fertig» angesehen wird, sondern an die Novasina AG überstellt und dort je nach künftigem Eigenbedarf weiterentwickelt wird, konnte dies bei der Planung entsprechend berücksichtigt werden.

Aus diesem Grund wurde grosser Wert daraufgelegt, den Einbau von zukünftigen Features und Technologien so einfach wie möglich zu halten, gerade im Hinblick auf

Entwickler, die nicht so gut mit dem Projekt vertraut sind. Dies schlägt sich zum Beispiel darin nieder, dass das Datenmodell möglichst generisch gehalten ist und dass, wo möglich, Interfaces statt konkreten Klassen verwendet wurden.

An solchen Stellen könnten zukünftige Features ohne allzu grossen Aufwand ansetzen. Würde die Novasina AG beispielsweise einen neuen Sensortyp entwickeln, so wäre es ein Leichtes, die neuen Messresultate als Datentyp in die Applikation zu integrieren. Auch ein neues Anschlussmedium wie den bereits erwähnten Bluetooth-Adapter oder eine Anbindung über eine IoT-Schnittstelle wäre denkbar.

Ziel der Arbeit war es, die Applikation auf diese Art und Weise für die Novasina AG nicht nur kurz- oder mittelfristig, sondern langfristig tragbar zu machen und ein solides Fundament zu schaffen, auf dem später weiterentwickelt werden kann.

**Usability**

Das zweite wichtige Kriterium, auf das besonders geachtet wurde, war die Usability der Applikation. Dieser Fokus entstand insbesondere aus der am ClimMate-Handgerät aufgefallenen Usability-Problematik und dem Wunsch der Novasina AG, über die dort vorgefundenen Konzepte zu iterieren und Verbesserungen vorzunehmen.

Profitieren konnte die Usability unseres Produktes insbesondere davon, dass der Grossteil der Zielgruppe bereits ein Smartphone besitzt und mit den grundlegenden Navigations- und Inputelementen des Android-Betriebssystems sowie der generellen Bedienung eines Gerätes mittels Touchscreen vertraut ist.

Diese Situation steht im Gegensatz zum ClimMate-Gerät, das ein völlig eigenes Eingabesystem verwendet und somit für den nicht damit vertrauten Nutzer unzugänglich erscheinen kann. Indem bewusst viele bereits vertraute Elemente für das UI-Design verwendet wurden, konnte die Applikation dem User von Anfang an weniger fremd erscheinen.

Zu diesen Elementen zählen unter anderem die Navigation verschiedener Ansichten über Tabs am Bildschirmrand, die Verwendung eines durch drei gestapelte Punkte gekennzeichneten Dropdown-Menüs und das Verwenden von Wisch-Gesten zur Manipulierung von Listen.

## 2.5 Resultate

**Überblick**

Alles in allem wurden die mit der Novasina AG erarbeiteten Ziele erreicht und das Minimum Viable Product zum Grossteil fertiggestellt. Gewisse Vorbehalte mussten durch die beschränkte Arbeitszeit und den Umfang des Projektes zwar gemacht werden, doch dabei handelt es sich vor allem um optionale Quality-of-Life- und tief priorisierte Features. Sämtliche Kernfunktionalität ist vorhanden und alle Use Cases werden abgedeckt. Das Resultat ist im jetzigen Zustand bereit für einen praktischen Testlauf und fast fertig für den Feldeinsatz.

Für Details bezüglich der Erreichung der Ziele und des Ausblicks für die Weiterentwicklung siehe Kapitel 3.5.

# 3 Projektdokumentation

## 3.1 Anforderungsspezifikation

### 3.1.1 Kontextdiagramm

**Systemkontext**

In Abbildung 4 ist der Kontext mit allen Schnittstellen ersichtlich, welche für diese Applikation von Belang sind.

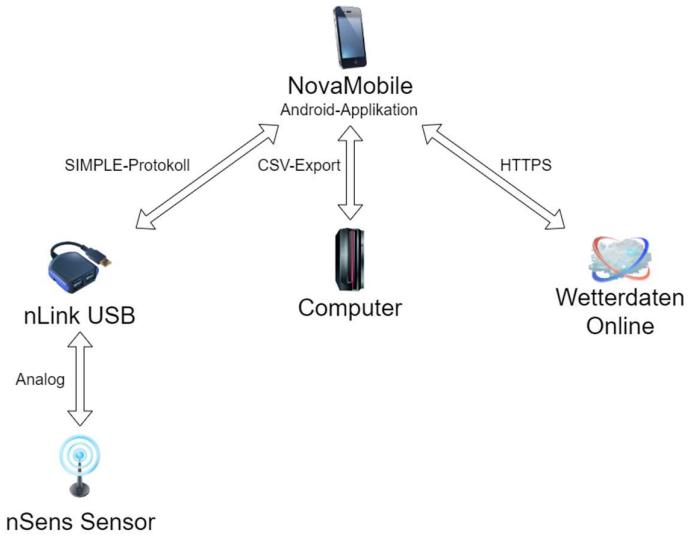


**Abbildung 4: Kontextdiagramm**

### 3.1.2 Use Cases

**Übersicht**

Die Applikation soll vom Leistungsumfang die gleichen Funktionalitäten wie das vorhandene ClimMate-Handgerät haben. Anhand der Kernfunktionen des Handgeräts lassen sich vier Use Cases daraus ableiten. In der folgenden Abbildung ist eine Übersicht dieser Use Cases dargestellt, welche in diesem Kapitel genauer beschrieben werden.

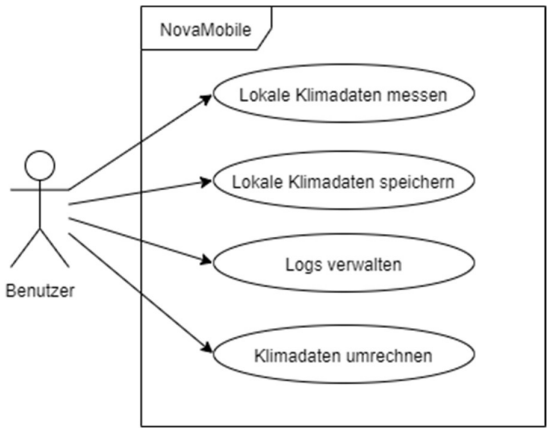


**Abbildung 5: Use-Case-Diagramm**

**UC1: Lokale Klimadaten messen**

**Primärer Actor:**

- User der App

**Stakeholders und Interessen:**

- User der App: Will schnelles und präzises Messen und Anzeigen der Klimadaten.

**Preconditions:**

- Anhand der gewünschten Klimadaten passender Sensor ist am Smartphone angeschlossen.
- Referenzdruck ist eingestellt.

**Success Guarantee:**

- User kann erwünschte Klimadaten am Gerät ablesen.

**Main Success Scenario:**

1. User entscheidet, welche Klimadaten für ihn im Moment von Belang sind.
2. User wählt korrekten (Ort, Capabilities) Sensor für die aktuelle Messung aus einer Liste aus.
3. User wählt alle relevanten Messgrössen zur Anzeige aus.
4. User liest vom Sensor ans Gerät übermittelte Messdaten ab.

**Extensions:**

- 3a: User hat versehentlich zu viele Messgrössen ausgewählt.
    a. User löscht Eintrag wieder aus Anzeigeliste.
- 4a: Gerät meldet Fehler bei Kommunikation mit Sensor.
    a. User leitet je nach Fehlertyp Massnahmen zur Problembehebung ein (Verbindungsprüfung, Software-Update, Sensor-Firmware-Update, Sensoraustausch).

**Häufigkeit:**

- Sehr häufig.

**UC2: Lokale Klimadaten speichern**

**Primärer Actor:**

- User der App

**Stakeholders und Interessen:**

- User der App: Will schnelles und präzises Messen und Speichern der Klimadaten.

**Preconditions:**

- Zu erwünschten Klimadaten passender Sensor ist an Smartphone angeschlossen.
- Referenzdruck ist eingestellt.

**Success Guarantee:**

- Erwünschte Klimadaten sind im Gerätespeicher hinterlegt.

**Main Success Scenario:**

1. User entscheidet, welche Klimadaten für ihn im Moment von Belang sind.
2. User wählt korrekten (Ort, Capabilities) Sensor für die aktuelle Messung aus Liste aus.
3. User wählt die relevanten Messgrössen zur Erfassung aus.
4. User startet Erfassung der Daten als Momentaufnahme.
5. Erfasste Daten werden im Speicher des Gerätes hinterlegt.

**Extensions:**

- 3a: User hat versehentlich zu viele Messgrössen ausgewählt.
    a. User löscht Eintrag wieder aus Anzeigeliste.
- 4a: User will anstelle einer Momentaufnahme eine Messreihe über bestimmten Zeitraum erfassen.
    a. User will Messreihe ohne Zeitlimit erfassen.
        1. User wählt Messreihenmodus aus.
        2. User startet Erfassung der Daten als Messreihe.
    b. User will zeitlimitierte Messreihe erfassen.
        1. User wählt Messreihenmodus aus.
        2. User gibt erwünschtes Zeitlimit ein.
        3. User startet Erfassung der Daten als Messreihe.
- 4b: Gerät meldet Fehler bei Kommunikation mit Sensor.
    a. User leitet je nach Fehlertyp Massnahmen zur Problembehebung ein (Verbindungsprüfung, Software-Update, Sensor-Firmware-Update, Sensoraustausch).
- 5a: User will Messresultaten einen Namen geben.
    a. User gibt erwünschten Namen in Speicherdialog ein.

**Häufigkeit:**

- Häufig.

**UC3: Logs verwalten**

**Primärer Actor:**

- User der App

**Stakeholders und Interessen:**

- User der App: Will tabellarische und grafische Einsicht in gespeicherte Messdaten.

**Preconditions:**

- Log der erwünschten Messdaten ist bereits im Gerätespeicher hinterlegt.

**Success Guarantee:**

- User kann erwünschte Messdaten tabellarisch und grafisch einsehen.

**Main Success Scenario:**

1. User entscheidet, welche Archiveinträge für ihn im Moment von Belang sind.
2. User wählt Eintrag zur Betrachtung aus Liste aus.
3. Gerät zeigt Messdaten tabellarisch und grafisch an.

**Extensions:**

- 2a: User will gezielt nach Einträgen suchen.
  a. User gibt Suchbegriffe (Bezeichnung, Sensor-ID, Datum etc.) in Suchfeld ein. Gerät zeigt nur auf die eingegebenen Begriffe passende Logeinträge in Liste an.
- 2b: User will Eintrag nicht einsehen, sondern löschen.
  a. User löscht Eintrag direkt aus Liste.
- 3a: User will Daten nach Einsicht exportieren.
  a. User wählt Exportfunktion und Exportmedium (Mail, CSV...) aus. Gerät exportiert Daten auf die gewünschte Weise.

**Häufigkeit:**

- Häufig.

**UC4: Klimadaten umrechnen**

**Primärer Actor:**

- User der App

**Stakeholders und Interessen:**

- User der App: Will bestimmte (fiktive) Klimadaten aus Klimarechner erhalten.

**Preconditions:**

- Keine.

**Success Guarantee:**

- User kann erwünschte Werte aus Rechner auslesen.

**Main Success Scenario:**

1. User entscheidet, welche Klimawerte er errechnen lassen will.
2. User wählt einen zu errechnenden Wert aus Liste aus.
3. User gibt dafür benötigte Grundwerte ein.
4. Gerät zeigt Ergebnis der Berechnungen an.

**Extensions:**

- Keine.

**Häufigkeit:**

- Selten.

### 3.1.3 Nichtfunktionale Anforderungen

**Funktionalität**

- Angemessenheit:
  - Messwerte werden auf zwei Nachkommastellen genau angezeigt.
- Richtigkeit:
  - In der Applikation kann der User keine Messdaten verändern.
- Sicherheit:
  - Die Datenbank kann nicht via Filesystem direkt aus dem Gerät ausgelesen werden.

**Zuverlässigkeit**

- Fehlertoleranz:
  - Bei Fehlern wird der Anwender benachrichtigt.

**Benutzbarkeit**

- Bedienbarkeit:
  - In Usability Tests, die sich an den Use Cases orientieren, muss es 90% der User möglich sein, den Test ohne Konsultieren der Dokumentation erfolgreich abzuschliessen.

**Effizienz**

- Zeitverhalten:
  - Das Anzeigen von Ansichten (ausgenommen grafische/errechnete Resultate) erfolgt in unter einer Sekunde. Das Referenzgerät hierzu ist ein Samsung Galaxy S8, auf dem die Applikation alleine läuft (keine Apps im Hintergrund).

**Änderbarkeit**

- Modifizierbarkeit:
  - Die Applikation bietet ein Interface an, mit welchem verschiedene Übertragungstechnologien (z.B. Bluetooth, USB) angebunden werden können.

**Übertragbarkeit**

- Installierbarkeit:
  - Die Applikation ist in einem Gesamtpaket installierbar. Es bestehen keine Abhängigkeiten zu anderen Services oder Installationen.
- Konformität:
  - Die Applikation bietet eine Datenschnittstelle, die mit dem Simple-Protokoll der Novasina AG konform ist.

### 3.1.4 Detailspezifikation

**Weitere Anforderungen**

Anforderungen, welche nicht zu den Use-Cases oder nichtfunktionalen Anforderungen zugeordnet werden konnten, werden hier festgehalten. Diese sehen wie folgt aus:

- Der Anwender kann den Prüfdruck für die Messungen einstellen oder aus einem Online-Wetterservice importieren lassen.
- Bei einem Versionierungskonflikt zwischen dem Sensor und der App wird gemäss dem Capability-Protokoll der Novasina AG der Anwender zum Aktualisieren der Software/Firmware aufgefordert.
- Bei Fehlern während der Messung werden die bis zu diesem Punkt gemessenen Daten abgespeichert.
- Die Applikation bietet die Möglichkeit an, CSV-Dateien über verschiedene Medien (z.B. E-Mail, Messenger usw.) zu exportieren.
- Permanente Löschungen müssen bestätigt werden.

## 3.2 Analyse (Business-Modell)

### 3.2.1 Domain-Modell

**Übersicht**

Die zu verwaltenden Objekte beschränken sich bei dieser Applikation auf die Messdaten und die verwendeten Sensoren. Das daraus abgeleitete Domain-Modell fällt daher eher simpel aus. In der Abbildung 6 ist eine Übersicht der Domäne ersichtlich. Für die folgenden Erläuterungen wird das Domain-Modell in die drei Teile «Sensor», «Messprozess» und «Messung» aufgeteilt.

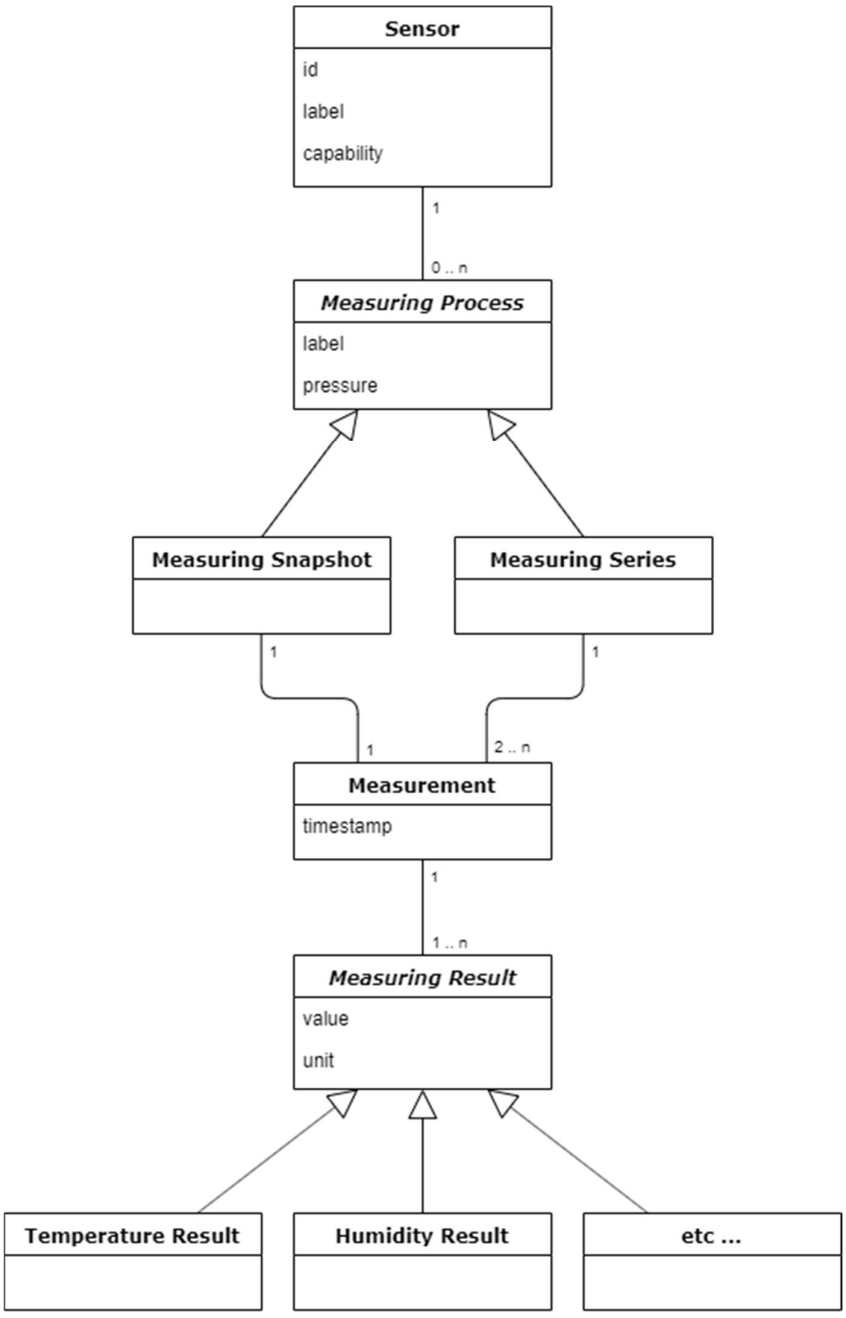


**Abbildung 6: Domainmodell**

**Sensor**

Jeder Sensor wird nach erstmaligem Anschliessen am Gerät abgespeichert. Somit wird er bei weiterem Gebrauch wiedererkannt. Ausserdem sind folgende Aspekte wichtig:

- Bei der ID handelt es sich um eine Hardware-ID. Diese ist einzigartig und wird aus dem Sensor ausgelesen und zur Identifikation gebraucht.
- Der Benutzer hat mit dem Label die Möglichkeit, den Sensor mit einer Beschriftung zu versehen. Somit kann dem Sensor ein Kontext (z.B. Keller Gebäude 2) gegeben werden, mit welchem die Suche im Archiv erleichtern werden soll.
- Ein Sensor muss nicht zwingenderweise mit einem Messprozess verknüpft sein. Dieser Fall tritt ein, wenn ein Sensor nur am Gerät angeschlossen und benannt, jedoch keine Messung mit ihm durchgeführt wird.

**Messprozess**

Ein Messprozess (Measuring Process) beinhaltet alle für den Messvorgang notwendigen Informationen. Hierbei sind folgende Punkte bemerkenswert:

- Wie beim Sensor hat auch hier der Benutzer mit dem Label die Möglichkeit, den Messprozess mit einer Beschriftung zu versehen (z.B. Audit Reinraum 2), damit die Suche im Archiv leichter wird.
- Bei jedem Messprozess wird der Referenzdruck aus den Einstellungen abgespeichert. Dieser wird benötigt, um nachträglich weitere Klimadaten aus den Sensordaten berechnen zu können. Anhand der Domain-Recherche mit dem Kunden zusammen hat sich ergeben, dass es in Zukunft und bei möglichen weiteren Sensoren keine zusätzlichen Referenzwerte brauchen wird. Daher wurde bei diesem Attributfeld auf eine Generalisierung verzichtet.
- Jeder Messprozess ist immer einem Sensor zugeordnet. Somit ist ersichtlich, mit welchem Sensor dieser Messprozess aufgenommen wurde.
- Jeder Messprozess ist entweder eine Einzelmessung (Measuring Snapshot) oder eine Messreihe (Measuring Series). Bei einer Einzelmessung wird nur eine einzige Messung abgespeichert. Bei einer Messreihe werden Messungen während einer bestimmten Zeitdauer in vorbestimmten Intervallen abgespeichert.

**Messung**

Da Sensoren mehrere Messresultate pro Messvorgang messen und zu einem späteren Zeitpunkt zusätzliche theoretische Messwerte berechnet werden können, repräsentiert eine Messung (Measurement) eine Ansammlung von Messresultaten (Measuring Result) zu einem bestimmten Zeitpunkt. Darüber hinaus sind folgende Punkte wichtig:

- Jede Messung hat einen Zeitstempel, an welchem die Messung erfolgt ist.
- Eine Messung hat immer mindestens ein Messresultat, kann jedoch auch eine beliebige Anzahl mehr besitzen.
- Ein Messresultat ist immer nur einer Messung zugeordnet.
- Jedes Messresultat besitzt einen Messwert und eine Masseinheit.
- Für jedes konkrete Messresultat wird eine eigene Klasse implementiert, denn lediglich über die Einheit ist ein Messresultat zu wenig genau definiert. Ausserdem können zu einem späteren Zeitpunkt auch weitere Masseinheiten, wie z.B. Kelvin, für bestehende Messresultate implementiert werden.

## 3.3 Design

**Detailierungsgrad**

Im Verlauf des folgenden Kapitels wird konzeptionell und funktionell auf verschiedene Betriebssysteme und Frameworks eingegangen. Es ist nicht das Ziel dieser Dokumentation, deren Funktions- und Anwendungsweise bis in Detail zu beschreiben. Es werden jedoch die für das Projekt wichtigsten Punkte hervorgehoben und auf die dazugehörigen offiziellen Dokumentationen verwiesen. Bei weiterem Bedarf können diese konsultiert werden.

### 3.3.1 Mobile-Betriebssystem

**Kriterien**

Für die Auswahl des Betriebssystems sind folgende Kriterien relevant:

- Verbreitung des Betriebssystems
- Gute Entwicklungsumgebung (Dokumentation, Testing usw.)
- Zukunftssichere Unterstützung
- Kostengünstige Anschaffung von neuen Mobil-Geräten

**Möglichkeiten**

Der aktuelle Markt (Jahr 2019) für Betriebssysteme von Mobilgeräten wird zurzeit von zwei Herstellern dominiert. Android von der Firma Google führt weltweit mit einem Marktanteil von 85%. iOS von Apple steht mit 13.3% an zweiter Stelle [5]. Alle anderen Betriebssysteme haben zusammen nur einen Anteil von 1.7%, weshalb man sie für die Zwecke dieses Projektes vernachlässigen kann.

Bei diesen Daten handelt es sich um den weltweiten Markt. Auf eine kundenorientierte Analyse unter den Abnehmern der Novasina AG wurde aus Zeitgründen verzichtet.

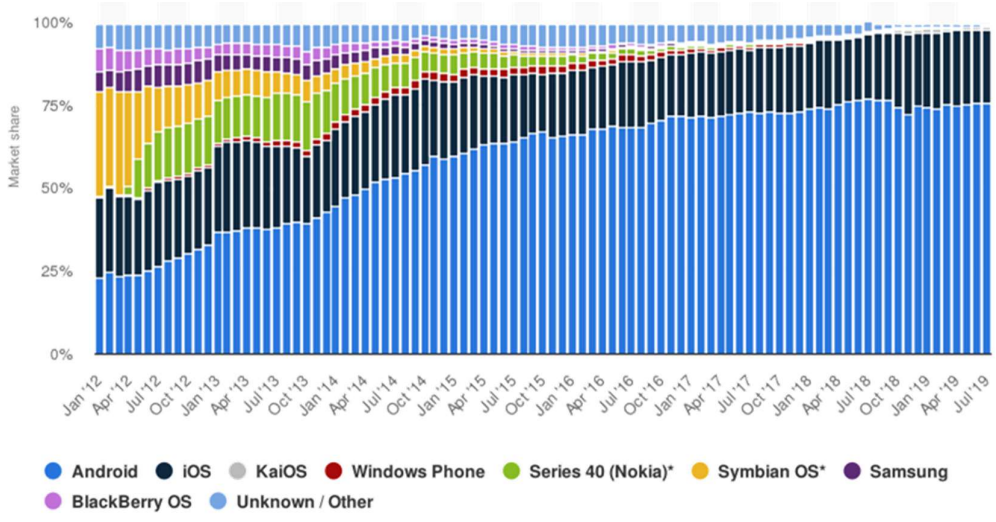


**Abbildung 7: Marktanteil von mobilen Betriebssystemen 2012-2019 [6]**

**Entscheidung**

Bezüglich Entwicklungsumgebung und Zukunftssicherheit sind sich Android und iOS ebenbürtig. Beide Betriebssysteme sind seit bereits über zehn Jahre im Einsatz und verfügen über ausführliche Tools und Onlinedokumentationen [7] [8].

Einzig in der Verbreitung und den Anschaffungskosten zeigen sich Unterschiede. Wie in Abbildung 7 dargestellt, besitzt Android das Mehrfache des Marktanteils von iOS. Betrachtet man zusätzlich noch die Preisentwicklung von Android- und iOS-Geräten vom letzten Jahrzehnt an, wird ersichtlich, dass der durchschnittliche Preis für iOS-Geräte konstant über 600 U.S. Dollar blieb, wahrendessen dieser für Android-Geräte auf fast 200 U.S. Dollar fiel. Die Neuanschaffung von Android-Geräten ist in diesem Fall um einiges kostengünstiger als die von iOS-Geräten.

Zu diesen marktorientierten Gesichtspunkten kommt die Tatsache, dass die Novasina AG bereits über mehr Development-Erfahrung im Android-Bereich verfügt und selbst schon eine eigene Applikation für diese Plattform entwickelt hat. Dies war mit ein Grund dafür, dass die finale Entscheidung schliesslich auf Android fiel.

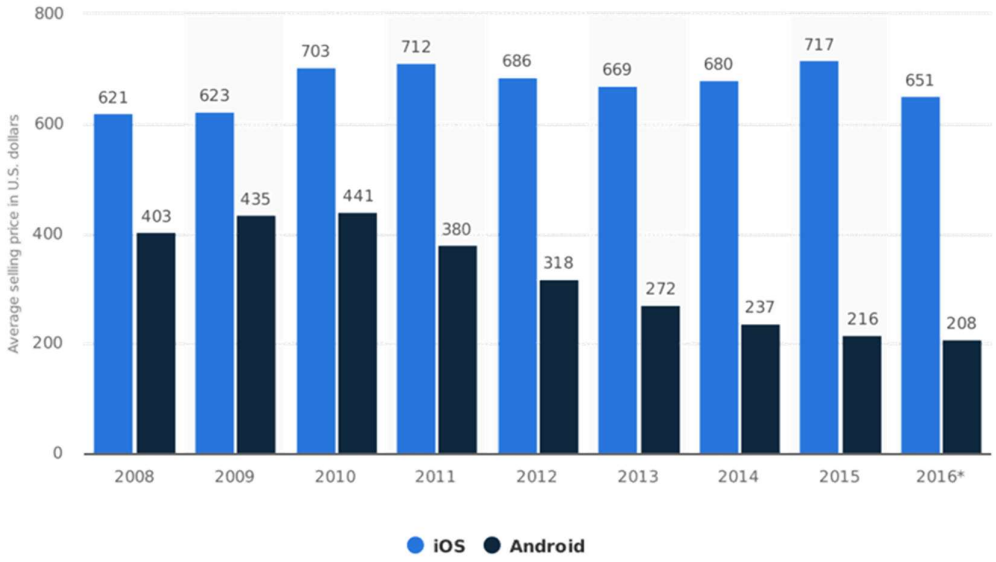


**Abbildung 8: Durchschnittspreis von Android- und Apple-Smartphones 2008-2016 [9]**

### 3.3.2 Mobile-Framework

**Kriterien**

Für die Auswahl des mobilen Frameworks sind folgende Kriterien relevant, hier absteigend sortiert nach Wichtigkeit:

- Programmiersprache (C#, da bei der Novasina AG hauptsächlich C#-Programmierer arbeiten)
- Langlebige Unterstützung des Frameworks
- Gute Dokumentation
- Portabilität auf andere Betriebssysteme

**Möglichkeiten**

Im Verlauf der letzten drei Jahre wurden folgende Android Frameworks am meisten verwendet [10] [11] [12]:

- Unity [13]
- Xamarin [14]
- React Native [15]
- Flutter [16]

**Entscheidung**

Da die oben genannten Frameworks am meisten verwendet werden, sind sie sich betreffend Dokumentation und Langlebigkeit ungefähr ebenbürtig. Einzig in der Portabilität und den verwendeten Programmiersprachen zeigen sich Unterschiede.

Bezüglich Portabilität schneidet React Native am besten ab, da es auf Web-Technologien wie HTML, CSS und Javascript basiert. Somit folgt React Native dem aktuellen Trend, Web-Technologien auf die verschiedensten Plattformen zu bringen. Deshalb kann das Frontend auch ohne grosse Anpassungen am Code mit Hilfe anderer Frameworks auf andere Plattformen wie z.B. Desktop (mit Electron [17]) übertragen werden. Lediglich der Treiber müsste neu implementiert werden. Alle anderen Frameworks binden den Code so stark an sich, dass er kaum unabhängig verwendet werden kann. Eine Portierung auf eine andere Plattform ist folglich nur innerhalb desselben Frameworks möglich. Jedoch erwies sich die Programmiersprache als Killerkriterium für die Auswahl des Frameworks. Aufgrund der Anforderung der Novasina AG, dass die zu verwendende Programmiersprache C# sein muss, wurde die Auswahl der Frameworks stark eingeschränkt. Das einzige Mobile-Framework für Android mit der Programmiersprache C# ist Xamarin. Xamarin selbst existiert in verschiedenen Ausführungen:

- **Xamarin Forms:**
  Ist ein Cross-Platform-Framework, mit welchem die programmierte Applikation für alle gängigen Plattformen (Android, iOS, Windows, OSX usw.) portiert werden kann. Da es für alle Plattformen funktionieren muss, wird in diesem Framework auch nur die Schnittmenge an Funktionen aller Plattformen angeboten. Spezifische Betriebssystemfunktionen, welche auf anderen Plattformen nicht existieren, sind im Framework nicht verfügbar.
- **Xamarin Android Native:**
  Ist eine direkte Übersetzung des Android Native Java API in C#. Alle Betriebssystemfunktionen von Android werden somit angeboten. Eine direkte Portierung auf eine andere Plattform ist ohne weiteres nicht möglich.
- **Xamarin iOS Native:**
  Ist eine direkte Übersetzung des iOS Native Swift API in C#. Alle Betriebssystemfunktionen von iOS werden somit angeboten. Eine direkte Portierung auf eine andere Plattform ist ohne weiteres nicht möglich.

Da für dieses Projekt ein eigener Treiber für die nSens-Sensoren von Novasina programmiert werden muss, sind spezifische Betriebssystemfunktionen betreffend der Übertragungsmedien vom Mobiltelefon notwendig. Aus diesem Grund wurde Xamarin Android Native für die Umsetzung dieses Projekts ausgewählt.

### 3.3.3 Weitere Frameworks, Libraries und Protokolle

**Klimarechner**

Der Klimarechner der App wurde komplett von der Novasina als C-Code zur Verfügung gestellt, da ein solcher Rechner bereits dort entwickelt wurde. Der Quellcode wurde im Rahmen des NovaMobile-Projektes nach C# portiert, jedoch sonst nicht verändert. Aus diesem Grund und in Absprache mit der Novasina AG übernehmen die Autoren dieser Arbeit keine Verantwortung für den Inhalt, die Qualität und die Korrektheit des Klimarechnercodes.

**Simple-Protokoll**

Das Simple-Protokoll ist ein proprietäres Protokoll der Novasina AG, welches dazu dient, mit den Sensoren der nSens-Reihe zu kommunizieren. Wie der Name schon sagt, handelt es sich dabei um ein vergleichsweise einfaches Protokoll, das nach dem Frage-Antwort-Schema funktioniert: Ein Sensor erhält eine Nachricht via nBus, die eine Instruktion enthält, und gibt eine passende Antwort dazu. Für weitere Informationen siehe Anhang G.

**UsbSerialForAndroid**

Für die Kommunikation mit dem Sensor muss der nBus-Adapter (Analog-zu-Digital-Adapter) der Novasina AG verwendet werden. Dieser Adapter hat für die USB-Verbindung einen FTDI-Chip eingebaut. Damit mit diesem auf Byte-Ebene kommuniziert werden kann, wird ein FTDI-Chiptreiber benötigt. Vom Hersteller gibt es eine 89-seitige Anleitung für Entwickler [18], die einen solchen Treiber implementieren wollen.

Fokus dieser Arbeit war von Anfang an die Erstellung einer App. Die Implementierung eines eigenen FTDI-Treibers hätte den Rahmen dieser Arbeit gesprengt. Deshalb wurde die Library UsbSerialForAndroid [19] verwendet, die einen solchen für die App geeigneten Treiber bereitstellt. UsbSerialForAndroid ist eine Portierung der beliebten Java-Library usb-serial-for-android [20] nach C# und kann somit unter Xamarin verwendet werden. Dieser Treiber übernimmt lediglich die Initialisierung der Verbindung (Baudrate usw.) und steuert die weiterführende Kommunikation, welche per Halbduplex erfolgt.

**Newtonsoft.Json**

Für den Gebrauch der asynchronen Kommunikation innerhalb des Android-Systems anhand von Intents [21] müssen oftmals Objekte serialisiert werden. Hierfür wird das NuGet-Paket Newtonsoft.Json [22] verwendet.

**SQLite-net**

Um die Datenbank schlank zu halten und doch auf eine mächtige Abfrage-API zugreifen zu können, wurde SQLite als Datenbanktyp für dieses Projekt ausgewählt. Hierfür wird das NuGet-Paket SQLite-net [23] verwendet. Dies ist ein SQLite Client, welcher speziell für Xamarin optimiert wurde und wegen seinen zahlreichen Funktionen (z.B. asynchrone API) oft verwendet wird.

**NUnit**

Zur Automatisierung der Tests wurde die weit verbreitete NUnit-Testing-Library [24] verwendet. Für Details siehe Kapitel 3.4.2.

**FluentAssertions**

FluentAssertions [25] ist eine Testing-Library, welche die Assertion von Testbedingungen über Extension Methods ermöglicht und somit NUnit um einige praktische Features erweitert. Für Details siehe Kapitel 3.4.2.

### 3.3.4 Architektur

**Architekturdiagramm**

Bei der Erstellung der Architektur wurde auf eine saubere Trennung der verschiedenen Schichten geachtet. Ausserdem sind folgende Punkte bemerkenswert:

- Der Layer «View» wird von der Struktur her anhand von Activities [26] und Fragments [27] komplett von Android vorgegeben und beschäftigt sich hauptsächlich mit den View-Logiken und der grafischen Darstellung. Genauere Informationen hierzu sind im Kapitel 3.3.7 zu finden.
- Der Logger-Service und der Sensor-Service sind spezifische Android-Objekte. Android bietet die Möglichkeit, eigene Services [28] zu implementieren. Anhand dieser ist es möglich, Funktionalitäten auch bei geschlossener App anzubieten. Diese Eigenschaft ist für das Betreiben der Sensoren sowie das Loggen von Messdaten erforderlich. Weitere Informationen zur konkreten Implementation sind im Kapitel 3.4.1 zu finden.
- Für den Gebrauch von Android-Services muss die asynchrone Kommunikation von Android anhand von Intents [21] verwendet werden. Sämtliche Kommunikationskanäle zwischen und zu den Services laufen somit asynchron.

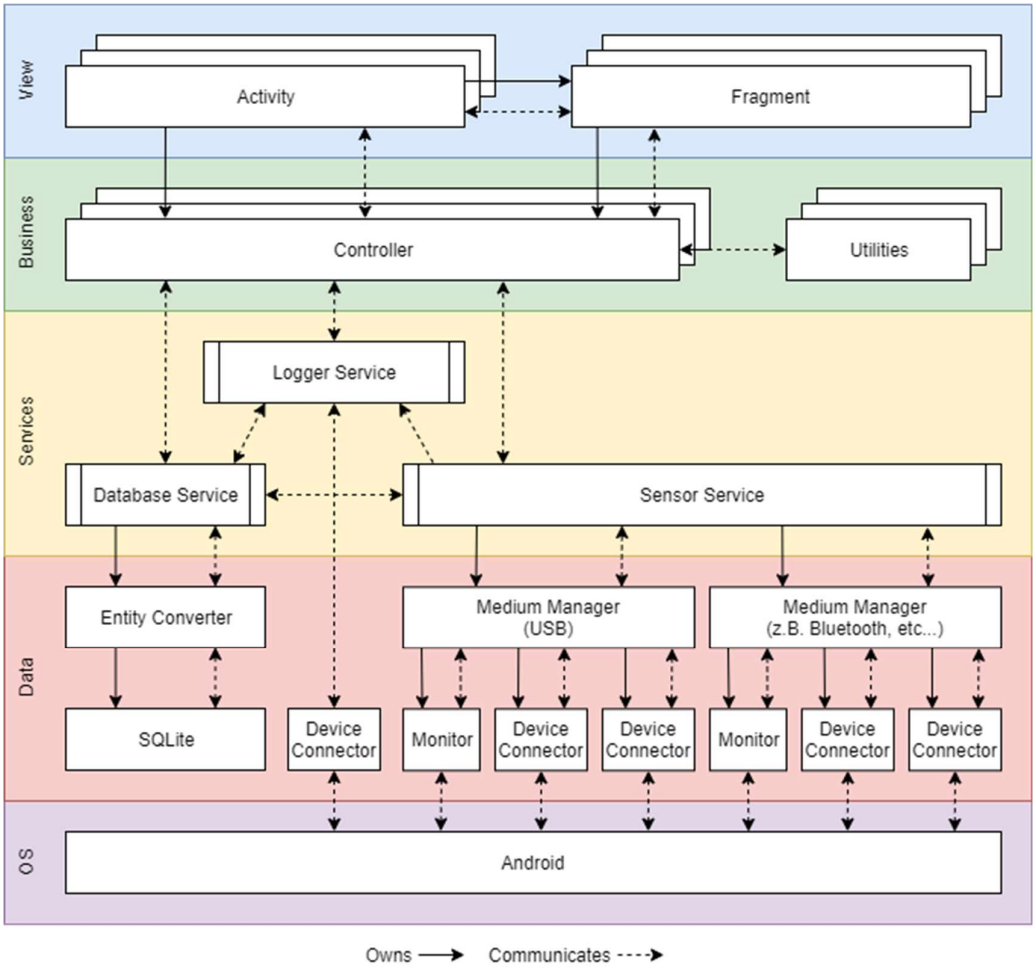


**Abbildung 9: Architekturdiagramm**

### 3.3.5 Package-Diagramm

**Package-Diagramm**

Die ganze Applikation wurde in folgende sieben Projekte aufgeteilt:

- **Common:** Beinhaltet alles, was verstreut an mehreren Orten in der Applikation gebraucht wird, z.B. Hilfsklassen, Generalisierungen, Domain-Objekte usw.
- **NovaMobile:** Beinhaltet Benutzeroberfläche und dazugehörige Businesslogik.
- **LoggerService:** Android-Foreground-Service, um Messreihen aufzunehmen.
- **SensorService:** Android-Background-Service, um auf Sensoren zuzugreifen.
- **Database:** SQLite Datenbank mit allen nötigen Queries.
- **UsbConnection:** MediumManager für USB. Siehe Kapitel 3.3.6.
- **UsbSerialForAndroid:** USB-Treiber für Android, importiertes Projekt.

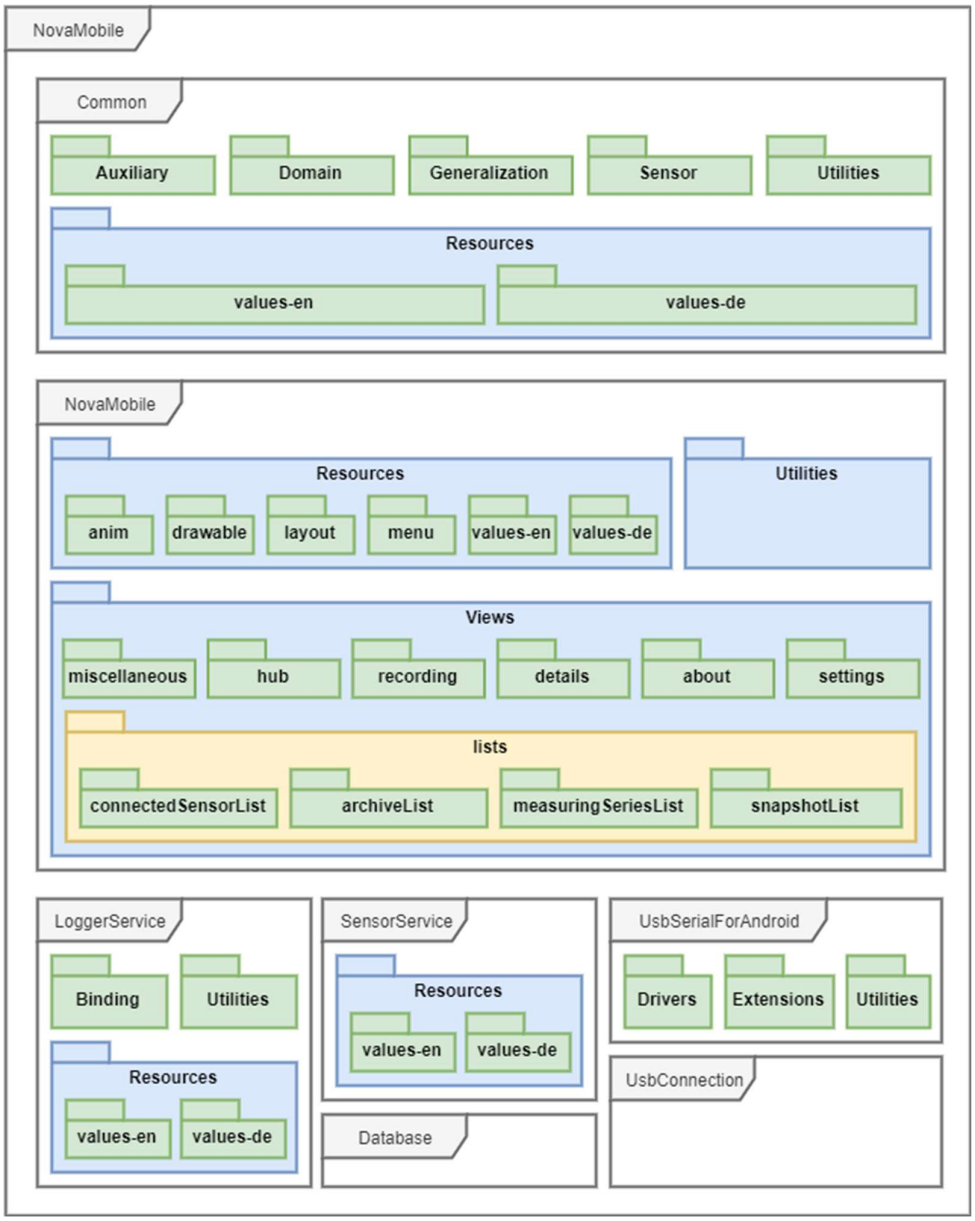


**Abbildung 10: Package-Diagramm**

### 3.3.6 Wichtige Klassenkonzepte

**Wichtige Interfaces und abstrakte Klassen**

Wie eingangs erwähnt wurde bei der Entwicklung grossen Wert auf die Erweiterbarkeit der Applikation gelegt. Insbesondere beinhaltet dies das Anschliessen von Sensoren über neue Medien (beispielsweise Bluetooth) und das Anschliessen von neuen Sensortypen (beispielsweise CO2-Sensoren).

Um dies gewährleisten zu können, war Abstraktion notwendig. Der Grossteil der Applikation sollte sich gar nicht damit befassen müssen, wie genau ein Sensor angeschlossen ist oder was für Messresultate er theoretisch liefern kann – lediglich die tatsächlich vorhandenen Resultate sollten von Belang sein.

Aus diesem Grund wurde die Entscheidung gefällt, wo immer möglich mit Interfaces und abstrakten Klassen zu arbeiten, um diese Abstraktionsstufe zu erreichen. Die wichtigsten davon werden anschliessend im Detail aufgeführt, speziell im Hinblick auf die Konsultation bei der Entwicklung von Erweiterungen für die Applikation.

**DeviceConnector**

Die abstrakte Klasse DeviceConnector schreibt vor, wie die Applikation mit einem angeschlossenen Sensor (also einem Device) kommunizieren muss, beziehungsweise was für eine Schnittstelle eine solche Verbindung gegenüber der Applikation anbietet. Sie ist der primäre Weg, mit den Sensoren der Novasina AG zu kommunizieren.

Die Klasse enthält fünf Methoden:

- **InitiateSimpleProtocol:** Diese abstrakte Methode dient dazu, das Startsignal für das Simple-Protokoll [29] an den Sensor zu senden und ihm so zu signalisieren, dass in Kürze Abfragen erfolgen werden. Diese Methode ist vor allem zum Aufruf im Konstruktor oder beim Anschluss-Handling ableitender Klassen gedacht.
- **GetSerialNumber:** Diese Methode fragt den Sensor nach seiner einzigartigen Seriennummer, die danach applikationsweit als ID für diesen spezifischen Sensor verwendet wird. Sie ist nicht zum öffentlichen Aufruf gedacht, sondern wird genau einmal aufgerufen, wenn erstmals die öffentliche SerialNumber-Property des DeviceConnectors von aussen her abgefragt wird. Dann wird das Resultat der GetSerialNumber-Abfrage in dieser Property zwischengespeichert und bei weiteren Anfragen wieder herausgegeben, um den Sensor nicht mit redundanten Anfragen zu belasten – die Seriennummer ändert sich schliesslich nie.
- **GetSensorType:** Diese Methode fragt den Sensor nach seinem Typen. Während der Entwicklung wurde nur mit dem Sensortyp HumidityTemperature gearbeitet, doch weitere Typen wären problemlos implementierbar.
  Bei GetSensorType kommt dasselbe «Lazy-Loading»-Prinzip wie bei GetSerialNumber zum Tragen. Es existiert eine öffentliche SensorType-Property vom Enum-Typ SensorType, die bei der erstmaligen Abfrage GetSensorType aufruft und das dort erhaltene Resultat danach zwischenspeichert. Aus dem Sensortyp lassen sich später auch die primären (direkt vom Sensor gelieferten) und sekundären (aus den primären Resultaten ableitbaren) Messresultate eruieren.
- **SetMeasurementStrategy:** Diese Methode setzt die Strategie zur Generierung von Measurements (siehe unten) fest. In einer solchen MeasurementStategy steht die Art und Weise, wie aus den Antwortbytes des Sensors auf eine Measurement-Anfrage die tatsächlichen Daten herausgelesen und korrekt interpretiert werden können. Sie variiert je nach Typ des angeschlossenen Sensors, hängt also von der SensorType-Property ab. SetStrategy sollte ein einziges Mal aufgerufen werden, nachdem der Typ des Sensors ermittelt werden kann (idealerweise im Konstruktor).

- **GetMeasurement:** Dies ist die einzige öffentlich verfügbare Methode der DeviceConnector-Klasse. Bei ihrem Aufruf fragt sie den Sensor nach einem aktuellen Messwert. Die so erhaltenen rohen Bytes werden dann über die zuvor gesetzte, vom Sensortyp bestimmte MeasurementStrategy korrekt interpretiert, in eine neue Instanz der Measurement-Klasse verpackt und zurückgegeben.

Die Entscheidung, die DeviceConnector-Klasse abstrakt und relativ generisch zu halten, wurde aufgrund der Tatsache gefällt, dass jede konkrete Implementation zwangsweise sehr stark von dem Medium (und dessen Implementation auf dem Android-Betriebssystem) abhängt, auf das sie sich stützt.
Im Falle des UsbDeviceConnectors handelt es sich dabei beispielsweise um das Android-USB-API und den UsbSerialForAndroid-Treiber, der zur Kommunikation mit dem USB-Adapter der Novasina AG dient. Weitere Medien könnten allerdings massiv davon abweichen. Dies macht es erforderlich, später folgenden Implementationen viel Freiraum zu lassen, was durch die Vagheit dieser Klasse gewährleistet wird.

**IMediumManager**

Das IMediumManager-Interface steht im Architekturdiagramm eine Ebene über der konkreten DeviceConnector-Implementation für das jeweilige Medium. Seine Aufgabe (beziehungsweise die Aufgabe der implementierenden Klasse) ist es, sich um alle DeviceConnectors zu kümmern, die das Medium verwenden. Dazu gehört insbesondere das Herstellen einer Verbindung, wenn ein neuer Sensor angeschlossen wurde, das saubere Aufräumen von Sensoren, deren Verbindung gekappt wurde, und das Melden von Fehlern in der Kommunikation mit dem Sensor. Zu diesem Zweck bietet das IMediumManager-Interface drei Events an, die konktrete Implementationen invoken müssen:

- **DeviceAdded** wird beim Anschluss eines neuen Sensors aufgerufen und übergibt einen DeviceConnector als Argument. Damit wird Interessenten mitgeteilt, dass ein neuer DeviceConnector angeschlossen wurde und sein Setup komplett ist, sodass er nun abgefragt werden kann.
- **DeviceRemoved** wird immer dann aufgerufen, wenn ein Sensor getrennt wurde. Es dient zur Signalisierung, dass nicht mehr versucht werden soll, mit diesem Sensor Kontakt aufzunehmen und unterbricht laufende Messvorgänge.
  Wie beim DeviceAdded übergibt auch DeviceRemoved einen DeviceConnector als Argument, damit klar wird, welcher Sensor gerade wegfiel.
- **DeviceErrorOccurred** signalisiert, dass bei der Kommunikation mit einem Sensor ein Fehler aufgetreten ist und dieser nicht mehr zur Verfügung stehen kann. Dabei wird ein String mitgegeben, welcher den oberen Schichten als Fehlermeldung dienen kann.

Ausserdem bietet das Interface zwei öffentliche Methoden ohne Rückgabewert an:

- **ScanExistingDevices** dient dazu, das Medium nach bereits vorhandenen angeschlossenen Sensoren abzufragen, diese zu verarbeiten (beispielsweise im Falle des UsbMediumManagers die Zugriffsbewilligung vom Betriebssystem einzuholen) und in die Liste der Sensoren aufzunehmen. Diese Methode wird idealerweise direkt nach der Konstruktion des MediumManagers einmal aufgerufen und soll verhindern, dass der Nutzer Sensoren, die bereits vor dem Start der Applikation angeschlossen worden waren, wieder trennen und erneut anschliessen muss, damit sie von der Applikation als verbunden erkannt werden können.

- **UnregisterMonitors** ist zum einmaligen Aufruf vor dem Abbau des MediumManagers gedacht und dient dazu, eventuell gehaltene Ressourcen (wie beispielsweise offene Datenbank-Verbindungen oder Android-Broadcast-Receiver) zu schliessen, damit nach dem Abbau kein Leak entsteht.

Ähnlich wie im Fall des DeviceConnectors wurde auch das IMediumManager-Interface bewusst vage gehalten. Es ist stark abhängig vom konkreten Medium und den damit verbundenen Schnittstellen, Einschränkungen und Rahmenbedingungen, die der konkrete MediumManager verwalten muss.

**MeasurementStrategy**

Die abstrakte MeasurementStrategy (beziehungsweise deren konkrete Subklassen) lösen das Problem, dass vor der Aktivierung des Sensors über das Simple-Protokoll nicht bekannt ist, um was für einen Sensortypen es sich handelt. Aus diesem Grund kann die Logik zum Auslesen der Messresultate nicht fix definiert werden, sondern muss dynamisch geladen werden, nachdem der Sensortyp feststeht. Wie der Name schon sagt, hilft die MeasurementStrategy hierbei, indem sie gemäss des Strategy Patterns [30] erstellt und später aufgerufen wird.

Jede Subklasse von MeasurementStrategy muss die Logik implementieren, wie aus einem bestimmten Sensortypen Measurements entnommen werden können. Dies umschliesst vor allem das Parsen der rohen Antwortbytes in C#-konforme Werte, meist Doubles. So bietet die MeasurementStrategy auch nur eine Methode an, GetMeasurementFromBytes, die ein Byte-Array entgegennimmt und das daraus abgeleitete Measurement zurückgibt.

**TypeFillerStrategy**

Die abstrakte TypeFillerStrategy setzt ebenso das Strategy Pattern ein und ist dafür verantwortlich, dass eine TypeFiller-Instanz weiss, welche Primärgrössen sie in einem Measurement erwarten und welche Sekundärgrössen sie daraus ableiten kann. Alle von ihr ableitenden Klassen müssen einerseits die Liste der Sekundärtypen definieren, andererseits eine Implementation von FillMeasurementWithTypes anbieten, welche ein unfertiges Measurement mit den nötigen Sekundärtypen auffüllen kann.

### 3.3.7 Wichtige Schnittstellen

**Services**

Die Kommunikation zwischen den Schichten erfolgt weitestgehend über Android-Intents und Broadcasts [21], welche mit Actions (übergebene Strings) einen Zweck erhalten, damit eventuelle Zuhörer wissen, was die Nachricht bewirken soll. Wo dies nötig ist, werden Extras verwendet, um den Intents Parameter hinzuzufügen.

Wichtig ist dabei insbesondere die Kommunikation mit dem SensorService, da dieser die wichtigen Schnittstellen zu Sensoren und Hardware anbietet, sowie dem LoggerService. Die folgenden Schnittstellen werden angeboten:

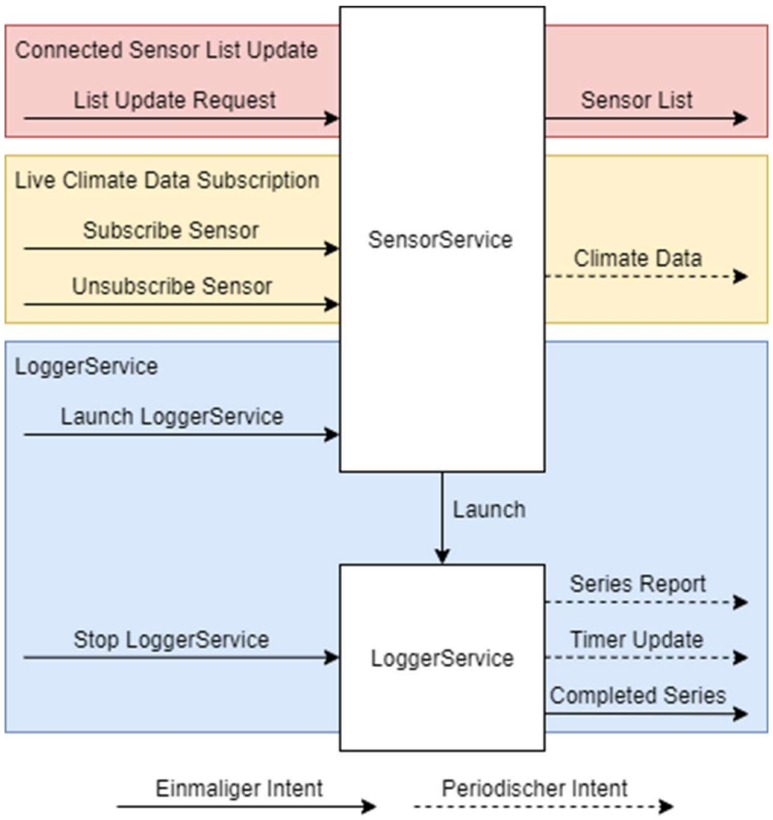


**Abbildung 11: Intent-Übersicht**

**Tabellarischer Überblick**

| Action-String | Extras | Return / Reaktion |
|---|---|---|
| **ActionSensorListRequest** | - | Intent mit Listenupdate (ActionSensorListUpdate) wird mit einem Broadcast übermittelt. |
| **ActionClimateSubscription** | Sensor-ID, Subscription-ID, (Intervall) | Neue Klimasubscription auf bestimmten Sensor wird mit Subscription-ID erstellt, Messintervall optional (Default: 1 Sek.). |
| **ActionClimateUnsubscription** | Subscription-ID | Klimasubscription mit bestimmter Subscription-ID wird gestoppt. |
| **ActionLaunchLoggerService** | Sensor-ID, (Intervall), Resultattypenliste, (Countdownzeit) | LoggerService wird im Hintergrund für bestimmten Sensor gestartet, erzeugt alle angegebenen Resultattypen. Intervall (Default: 5 Sek.) und Timer (Default: keine Begrenzung) optional. |
| **ActionSensorServiceStop** | - | SensorService wird gestoppt. |

**Tabelle 1: Action-Übersicht**

### 3.3.8 UI-Design

**Konzept**

Bei der Erstellung eines UI-Konzeptes ist es wichtig, Sinn und Zweck der Software sowie die Benutzergruppen zu berücksichtigen. Da es sich bei dieser App um ein Arbeitstool handelt, sollte die Oberfläche möglichst simpel und effizient gestaltet sein. Aus diesem Grundgedanken sind folgende Überlegungen entstanden:

- **Struktur:**
  Die Struktur sollte sich am Kernkonzept von ClimMate und dessen Aufbau orientieren. Ist das Konzept von ClimMate bekannt, sollte somit auch die Navigation durch die App intuitiv erfolgen.
- **Orientierung:**
  Bei der Benutzung sollte jederzeit ersichtlich sein, wo man sich in der Applikation befindet. Grundsätzlich hängt der Detaillierungsgrad von der Komplexität der Navigationsstruktur ab. Je mehr Ansichten vorhanden sind, desto genauer muss die Struktur angezeigt werden.
- **Navigation:**
  Die Navigation sollte ermöglichen, dass von jedem Ort in der App in die wichtigsten und meistverwendeten Ansichten gewechselt werden kann. So soll ein flüssiger Workflow garantiert werden.
- **Verschachtelungstiefe:**
  Ein hilfreiches Merkmal für die Ermittlung der Komplexität einer Applikation ist die Verschachtelungstiefe. Je mehr Ansichten durchlaufen werden müssen, um eine bestimmte Operation auszuführen, desto mehr Zeit wird für die Suche solcher Funktionen benötigt. Grundsätzlich gilt: Je weniger, desto einfacher.
- **Farbkonzept:**
  Farben können bei Benutzeroberflächen auf verschiedene Arten eingesetzt werden. Auf Webseiten werden Farben oft für eine ansprechende visuelle Präsentation gebraucht. Bei einem Arbeitstool führt dies jedoch oft zu Ablenkungen und verschleiert wichtige Funktionen. Um dies zu verhindern, können Farben auch funktionell für die schnelle Erkennung wichtiger Bestandteile in der Ansicht eingesetzt werden.
- **Layout:**
  Die richtige Anordnung der verschiedenen Funktionsbausteine in einer Ansicht trägt dazu bei, dass ein optimaler Workflow möglich ist. Wenn jedoch jede Ansicht ein komplett anderes Layout verwendet, erschwert dies das Zurechtfinden in der Applikation. Aus diesem Grund sollen möglichst ähnliche und gut gewählte Layouts wiederverwendet werden.

**Umsetzung**

Anhand der oben genannten Gedanken wurden für das UI-Design folgende Grundsätze umgesetzt:

- **Struktur:**
  Die App hat drei Hauptbereiche: Messen, Rechner und Archiv. Diese Bereiche entsprechen den Hauptfunktionen des ClimMate, weshalb diese auch jeweils eine eigene Hauptansicht im Hauptmenü erhalten. Für weiterführende Funktionen (Messung aufnehmen, Detailansicht, usw.) werden eigene, neue Ansichten erstellt.
- **Orientierung:**
  Aufgrund der niedrigen Verschachtelungstiefe wurde entschieden, dass für die Orientierung des Benutzers nicht immer der genaue Pfad angezeigt wird. Hierfür reicht es aus, wenn die jeweilige Hauptkategorie (Messen, Rechner oder Archiv) ersichtlich ist. Dies wird in der Menüleiste optisch hervorgehoben.
- **Navigation:**
  Anhand der Menüleiste kann in die wichtigsten Bereiche der App gewechselt werden. Weiterführende Funktionen benötigen nur eine weitere Ansicht. Somit kommt man mit nur einem Zurück-Klick ins Hauptmenü. Dies ermöglicht eine schnelle Navigation durch die App.
- **Verschachtelungstiefe:**
  Die Verschachtelungstiefe der Ansichten ist nicht mehr als drei Ebene tief, somit wird unnötige Komplexität verhindert.
- **Farbkonzept:**
  Die App verzichtet auf aufwendige und ablenkende Darstellungen. Die Benutzeroberfläche ist deshalb hauptsächlich mit Grautönen designt. Wichtige Informationen, sowie Hauptfunktionen werden mit wenigen Akzentfarben hervorgehoben. Hierbei wird das Magenta des Novasina-Logos für primäre Funktionen wie das Öffnen einer neuen Ansicht oder das Starten einer Messung gebraucht. Für sekundäre Funktionen wie das Anpassen von Parametern wird ein Seegras-Grün verwendet, welches gut mit dem Magenta kontrastiert.
- **Layout:**
  Listen und Details sehen bei den verschiedenen Messprozessen ungefähr gleich aus. Natürlich gibt es inhaltliche Unterschiede, jedoch ist die grobe Anordnung dieselbe geblieben und ähnliche Funktionen sind im gleichen Bereich zu finden.

**Wireframe**

Wireframes wurden benutzt, um einen frühen konzeptionellen Entwurf der Applikation darzustellen. Diese dienen unter anderem als Kommunikationsgrundlage mit dem Industriepartner. Hierbei geht es in erster Linie um das Konzept. So wird sichergestellt, dass alle verlangten Funktionen vorhanden sind und diese in einer sinnvollen Anordnung gegliedert werden. Nachfolgend sind als Beispiel zwei unterschiedliche Ansichten zu sehen. Alle erstellten Wireframes sind im Anhang C zu finden.

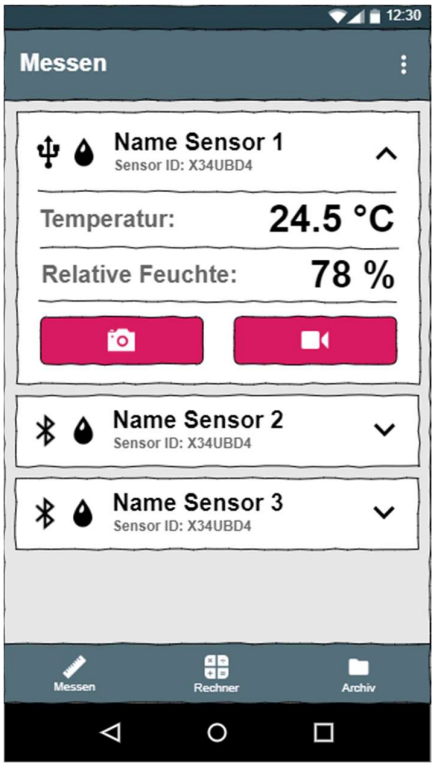


**Abbildung 12: Wireframe der Sensorübersicht**

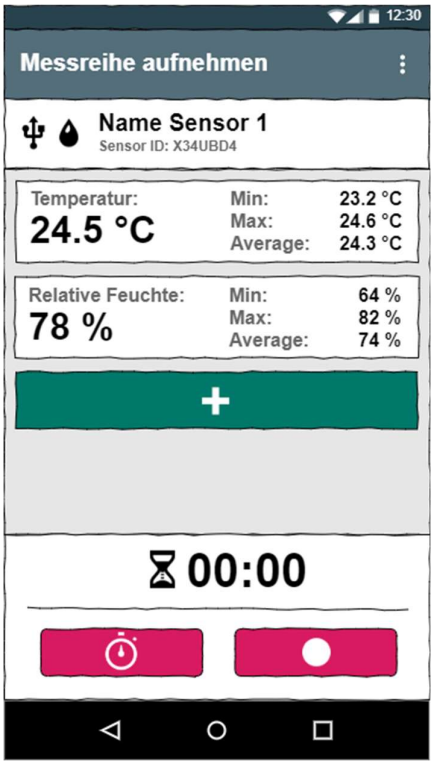


**Abbildung 13: Wireframe der Aufnahme-Ansicht**

Die in dieser Dokumentation gezeigten Wireframes entsprechen dem Planungsstand am Ende der Elaborationsphase. Während der Implementierung wurde aufgrund von Feedbacks und Kundenwünschen die Benutzeroberfläche laufend erweitert und angepasst. Abweichungen zwischen den Wireframes und dem Endprodukt wurden somit bewusst akzeptiert.

**Navigationsübersicht**

In Abbildung 14 ist eine Übersicht der Ansichten mit allen Navigationsmöglichkeiten dargestellt. Vereinzelt werden in einer Activity mehrere Fragmente gebraucht (z.B. Listen-Ansichten), welche jedoch keinen Einfluss auf die Navigation haben. Diese werden zugunsten übersichtlicherer Darstellung vereinfacht als Stapel angezeigt.

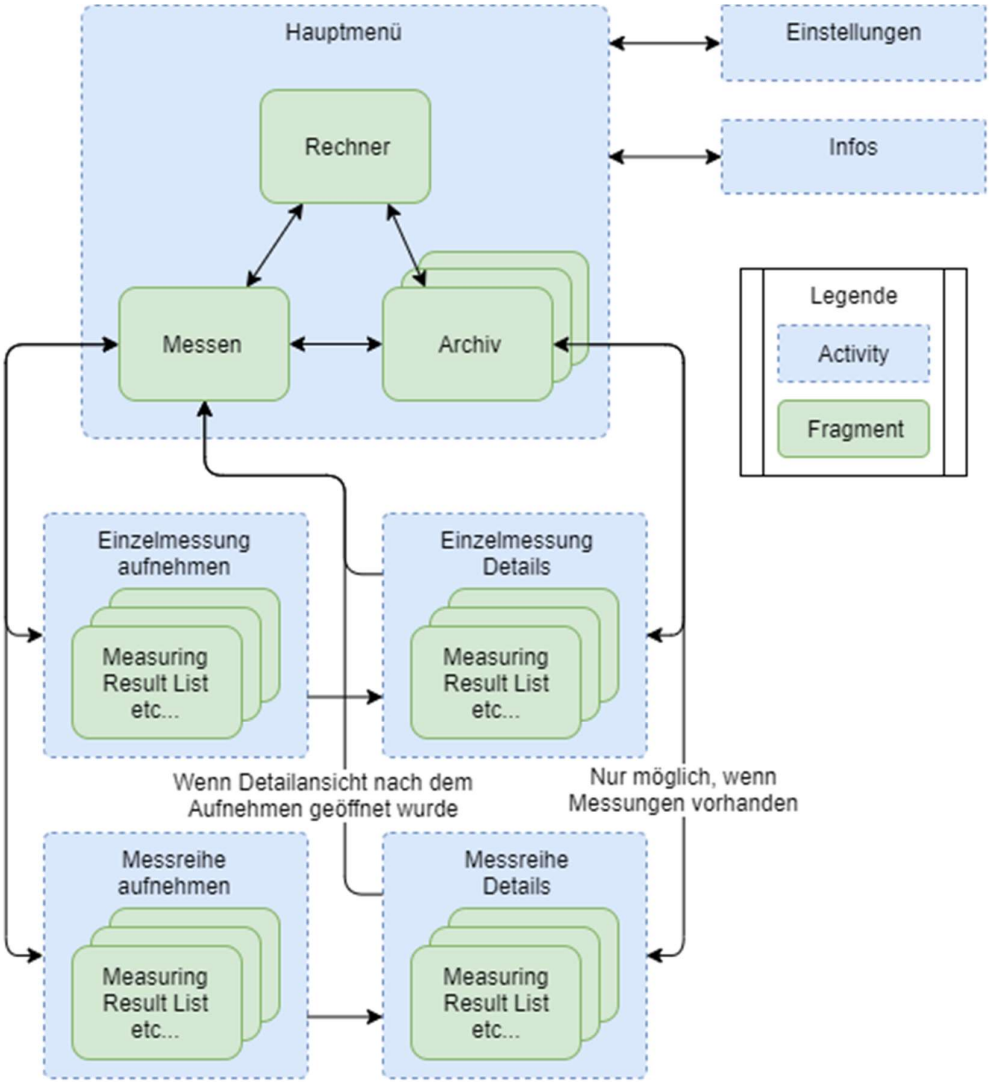


**Abbildung 14: Navigationsübersicht**

**UI-Pattern**

Android wurde ursprünglich nach dem Model-View-Controller (MVC) UI-Pattern [30] konzipiert. Erst zu einem späteren Zeitpunkt wurde Android mit den UI-Patterns Model-View-Presenter (MVP) [31] und Model-View-ViewModel (MVVM) [32] erweitert, um Kopplungen zu minimieren und die Testbarkeit zu verbessern. Somit wäre MVVM die optimale Wahl, um möglichst viel automatisiert testen zu können.

Dies ist die Situation, wenn nativ mit Java eine App für Android programmiert wird. Leider ist keine direkte MVVM-Umsetzung für C# in Xamarin Android Native verfügbar. Hierfür müsste auf weitere Frameworks zurückgegriffen werden (z.B. Xamvvm [33]).

Jedoch stellte sich heraus, dass selbst das MVVM-Pattern keine Verbesserung bezüglich der Testbarkeit für dieses Projekt mit sich bringt. Da gemäss den Anforderungen auch bei einem gesperrten Mobiltelefon Messungen gemacht werden können müssen, erfordert dies die Verwendung von Android-Services und deren asynchroner Kommunikation. Somit wäre selbst ein ViewModel so stark an das Android-System gebunden, dass ein vernünftiges und realisierbares Testing nicht zulassen würde. Aus diesem Grund und um eine schnellere Entwicklung zu ermöglichen, wurde das MVC-UI-Pattern für die App verwendet.

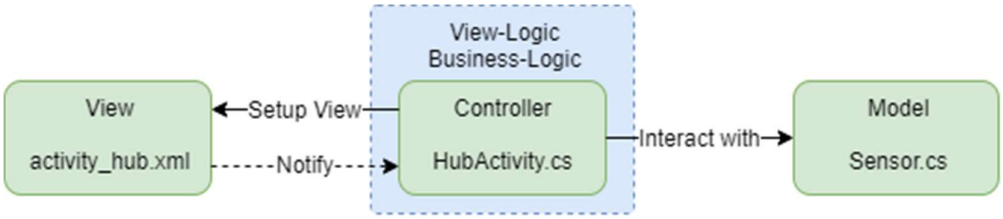


**Abbildung 15: Umsetzung des MVC-Patterns bei Android**

Wie in Abbildung 15 ersichtlich, verwaltet der Controller wie von Android vorgesehen beim MVC-Pattern die View-Logik und die Business-Logik. Mit View-Logik und Business-Logik ist folgende Definition zu verstehen:

- **View-Logik:**
  Die View-Logik umfasst die Initialisierung aller Ansichten und die Logik, wann und nach welcher Aktion welche Ansicht gezeigt werden soll.
- **Business-Logik:**
  Die Business-Logik beinhaltet die korrekte Bearbeitung und Verwaltung der Domain-Daten. Sie stellt Richtigkeit und Konsistenz der Daten sicher, verwaltet Zugriffe auf die Datenbank und regelt die Kommunikation mit anderen Services.

Um diese zwei Logiken sauber zu trennen und um zu viel Code in einer Klasse vorzubeugen, wurde, wo sinnvoll, zusätzlich zur Activity (übernimmt View-Logik) ein Controller für die Business-Logik implementiert.

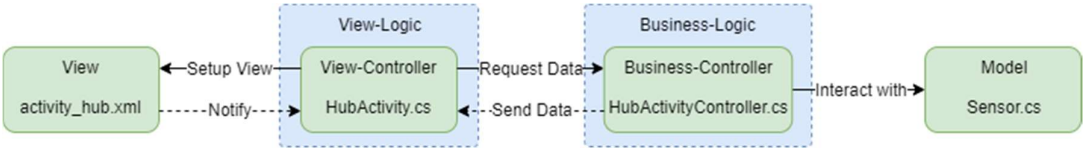


**Abbildung 16: Umsetzung des MVC-Patterns bei NovaMobile**

## 3.4 Implementation und Test

### 3.4.1 Implementation: Erläuterungen wichtiger konkreter Klassen

**SensorService**

Der SensorService ist die zentrale Stelle, an der die Kontrolle aller angeschlossenen Sensoren zusammenläuft. Er ist die höchste Instanz, was Informationshaushalt und Management der Sensoren angeht. Wie der Name schon andeutet, erbt er von der Android-Service-Klasse und läuft als Background Service. Dies bedeutet, dass er im Hintergrund lauffähig ist, ohne dass der Nutzer sich dessen bewusst ist, und dass er von anderer Stelle her über Intents erreicht werden kann.

Der SensorService erhält immer direkt von der Applikation (repräsentiert durch die Klasse NovaMobileApplication) selbst ein Startsignal, sobald eine zur Applikation gehörende Activity in den Vordergrund kommt. Dieses Startsignal hat allerdings nur eine Wirkung, wenn der SensorService gerade nicht läuft, ansonsten wird es ignoriert, da Android-Services immer Singletons sind. Das Signal wird trotzdem jedes Mal ausgelöst, weil der Ressourcen-Manager des Android-Betriebssystems Services gegebenenfalls unvermittelt beenden kann, wenn er eine Ressourcenknappheit (z.B. wenig Arbeitsspeicher) feststellt. Sollte dies eintreten, wenn der User gerade Funktionen in Anspruch nehmen will, die sich auf den Service stützen, wäre die Konsequenz Reaktionslosigkeit im besten und ein Absturz im schlimmsten Fall. Mit der Sicherstellung des Service-Starts bei Aufruf jeder Activity bewegt die Applikation sich auf der sicheren Seite, und diese Fehler können nicht auftreten. Wird die gesamte Applikation terminiert, entweder vom Nutzer oder vom Betriebssystem, stellt sie vor dem Abbau sicher, dass auch der SensorService ordnungsgemäss beendet ist.

Vom SensorService wird eine Liste aller MediumManager (pro am Gerät verfügbarem Medium einer) geführt, die er selbst konstruiert, wenn er gestartet wird. Die MediumManager in dieser Liste geben dem SensorService jeweils über ihre Methoden und Events Updates über die von ihnen verwalteten DeviceConnectors. Damit kann der SensorService ein vollständiges Dictionary mit einem Mapping aller verfügbaren Sensoren auf die dazugehörigen DeviceConnectors erstellen und so die Details der Hardware-Implementation der DeviceConnectors und MediumManager gegen oben hinter der sehr unkomplizierten Sensor-Klasse verstecken. Die tatsächliche Art der Verbindung spielt somit für alle Klassen, die vom SensorService abhängen, keine Rolle mehr.

Die Kommunikation mit dem SensorService erfolgt, wie Android dies vorsieht, ausschliesslich über Intents und die zugehörigen Actions, wozu ein SensorService-BroadcastReceiver nötig ist, der die empfangenen Signale an den SensorService weiterleitet. Die verfügbaren Actions sind die folgenden:

- Anfordern eines Updates der Liste mit verfügbaren Sensoren (beispielsweise zur Darstellung einer Sensorlist im GUI). Der Service versendet manchmal auch unaufgefordert Updates, und zwar immer dann, wenn sich etwas an der Liste der verfügbaren Sensoren ändert, also einer hinzukommt oder wegfällt. Dies dient dazu, potenzielle Interessenten auf dem neusten Stand zu halten. Proaktive Anfragen nach der Liste können trotzdem gestellt werden, denn wenn sich beispielsweise ein GUI-Element erstmalig aufbaut, hat es noch keinerlei Informationen zur Liste und muss sie erst einmal beziehen können.
- Starten einer Klimadaten-Subscription auf einem bestimmten Sensor: Hat ein Element Interesse an den Klimadaten eines bestimmten Sensors, so kann es eine Anfrage für eine Klimadaten-Subscription starten. Dabei kann es die Sensor-ID des fraglichen Sensors, ein Abfrageintervall und die einzigartige ID der Subscription angeben. Der SensorService erstellt dann aus diesen Informationen einen SensorUpdateRunner mit einem eigenständig laufenden Thread, der den

DeviceConnector des jeweiligen Sensors im angegebenen Abfrageintervall um ein Measurement abfragt und dieses dann an die Subscription-ID broadcastet, wo es weiterverarbeitet werden kann.

- Stoppen einer Klimadaten-Subscription: Wird eine Klimadaten-Subscription nicht mehr benötigt, so kann der Aufrufer sie auch wieder terminieren. Dazu muss lediglich die einzigartige Subscription-ID angegeben werden. Der SensorService wird dann den damit assoziierten SensorUpdateRunner stoppen.
- Starten des LoggerService: Der SensorService ist auch für den Launch des LoggerService verantwortlich (siehe unten). Hierzu müssen eine Sensor-ID, eine Liste von Typen zur Speicherung der erwünschten Messresultate und optional ein Messintervall mitgegeben werden.

Durch diese Mechaniken können Interessenten Sensor-Updates erhalten, ohne jemals eine direkte Referenz auf den Sensor selbst (beziehungsweise seinen DeviceConnector) halten zu müssen.

**DatabaseAccess**

Durch die DatabaseAccess-Klasse sind Methoden zum Zugriff auf die der Applikation zu Grunde liegende SQLite.NET-Datenbank verfügbar. Wird an irgendeiner Stelle im Programm Zugriff auf die Datenbank benötigt, so muss zuvor eine neue Instanz der DatabaseAccess-Klasse erstellt werden. Diese öffnet dann während der Konstruktion eine Verbindung zum Datenbank-File und ermöglicht danach alle nötigen Zugriffe. Nach der Tätigung aller Abfragen muss die DatabaseAccess-Instanz wieder geschlossen werden.

Die bei der Implementation verwendete Philosophie lautet, dass im aufrufenden Programmfluss sehr wenig, idealerweise gar keine Datenbanklogik implementiert werden soll, und dass die Anfragen für den Aufrufenden möglichst simpel sein sollen. Sämtliche eventuell anfallende Komplexität wird hinter der DatabaseAccess-Front versteckt (siehe unten, EntityConverter).

Die DatabaseAccess-Klasse offeriert eine ausreichende Bandbreite an Methoden, um alle Bedürfnisse der restlichen Applikation abzudecken. Dazu zählen vor allem CRUD-Methoden für Sensoren, MeasuringProcesses und VisibilityPreferences sowie Methoden zur Abfrage von Metadaten.

**EntityConverter**

Für aussenstehende Aufrufer unsichtbar läuft vor den Datenbank-Zugriffen der DatabaseAccess-Klasse noch der zwischengeschaltete, statische EntityConverter. Domain-Objekte (POCOs, Plain Old C# Objects) werden nämlich nicht direkt in die Datenbank abgelegt, sondern zunächst in dem jeweiligen Typ zugehörige Entitäten umgewandelt.

Die Entscheidung, diese Konversion zu implementieren, wurde aus mehreren Gründen gefällt. Erstens sind die Informationen, die bei der Abfrage verfügbar sein müssen, zum Teil in den POCOs verschachtelt. Die Umwandlung in Entitäten erlaubt es, diese direkt nach oben zu ziehen und aus der DB abfragbar zu machen. Zweitens verletzt es die Separation of Concerns, Datenbankinformationen wie IDs gegen aussen weiterzureichen, wenn dies nicht unbedingt nötig ist. Drittens sollte POCO-Code nicht unnötig mit von der Datenbank benötigten Annotationen versehen werden. Und schlussendlich erlaubt das Durchlaufen des Entity Converters einen Integritätscheck der Daten, bevor sie in die DB geschrieben oder ausgelesen werden, was sicherstellt, dass sie plausibel sind.

Konkret bietet der EntityConverter die folgenden statischen Methoden für die Konversion von Sensoren, MeasuringProcesses (hier wird zwischen MeasuringSnapshot und MeasuringSeries via Overload unterschieden) und VisibilityPreferences an:

- ObjectToEntity wandelt ein POCO in eine Entität um, die danach direkt in die Datenbank geschrieben werden kann.

- EntityToObject wandelt eine Entität aus der Datenbank wieder in das vom Rest der Applikation verwendbare Objekt um.
- Im Fall des MeasuringProcess kommt zusätzlich die Methode EntityToMetadata hinzu, die die zugehörigen Metadaten aus einer MeasuringProcessEntity in ein Objekt extrahiert.

**LoggerService**

Die Aufgabe des LoggerService ist es, sich um das Erfassen und Abspeichern von Messreihen (auch als Logs bezeichnet) zu kümmern, während die App im Hintergrund läuft (weil der User ins Android-Hauptmenü zurückgekehrt ist oder die Tabs gewechselt hat) oder sogar beendet wurde. Dazu sind einige spezielle Punkte zu beachten.

Beim LoggerService handelt es sich um einen sogenannten Foreground Service. Dies bedeutet, dass er, wie der SensorService auch, von der Service-Klasse erbt. Anders als letzterer wird der LoggerService allerdings im Vordergrund gestartet und muss dem Nutzer deshalb eine Notification in der Android-Statusleiste anzeigen. In unserem Falle handelt es sich dabei um eine kleine Statusanzeige mit der ID des momentan loggenden Sensors, die aufgezogen werden kann und dann Live-Messwerte bietet.

Mit der Verpflichtung, eine solche Notification anzeigen zu müssen, geht allerdings ein System-Privileg einher: Normalerweise können Services, die im Hintergrund ablaufen, vom Android-Betriebssystem ohne Vorwarnung abgeräumt werden, um Systemressourcen und Speicher zu sparen. Das gilt beispielsweise auch für den SensorService, weshalb dieser jeweils manuell neu gestartet wird, damit das nicht passieren kann. Dies bedeutet aber, dass ein solcher Service sich nicht dafür eignet, im Hintergrund längerfristig Logs zu erfassen. Für den Foreground-Service gilt diese Beschränkung nicht. Da dieser dem User direkt angezeigt wird, kommt ihm von Android die höchste Priorisierung entgegen und er wird nur im Notfall abgeräumt. Genau diesen Effekt nutzt der LoggerService aus, um längerfristiges Erfassen von Messreihen zu garantieren.

Trotzdem bleiben noch einige Probleme. NovaMobile kommuniziert mit den Sensoren über DeviceConnectors. Da der SensorService aber beim Verlassen der App mitsamt den DeviceConnectors abgeräumt wird, muss der LoggerService eine Referenz auf den DeviceConnector behalten, den er zum Loggen braucht. Dies ist unter der Android-Designphilosophie nicht problemlos machbar, da über Intents keine Referenzen übergeben werden können.

Aus diesem Grund existiert die Hilfsklasse LoggerLauncher. Sie erstellt einen LoggerService als BoundService [28], der via einer ServiceConnection direkt angesprochen werden kann. So können die benötigten Referenzen übergeben werden. Der LoggerService wird danach unbound und ist eigenständig lauffähig.

### 3.4.2 Automatische Testverfahren

**Problematik**

Ein Grossteil der implementierten Klassen stützt sich direkt oder indirekt auf vom Xamarin-Framework angebotene Android-Systemklassen. Dabei handelt es sich entweder um Hintergrund- und Hilfsklassen, wie beispielsweise die vom UsbDeviceConnector verwendete Klasse UsbDevice oder die Service-Klasse oder um GUI-Klassen wie Fragment und Activity. Diese Abhängigkeiten ziehen sich also von oben bis unten durch das Architekturmodell.

Dies stellt das automatische Testing vor gewisse Herausforderungen. Es ist nicht ohne massiven Mehraufwand möglich, diese Klassen zu mocken, um sie danach in Unit Tests substituieren zu können. Im Falle der GUI-Klassen wäre ein solches Mocking auch gar nicht sinnvoll.

**Grundsätze**

Ein Testing-Framework für diese Abhängigkeiten zu entwerfen und zu implementieren, würde den Rahmen dieses Projektes sprengen. Aus diesem Grund wurde die Entscheidung gefällt, sich bei allen von Android-Systemklassen abhängigen Programmteilen auf manuelle Testverfahren (s. Kapitel 3.5.3) zu beschränken.

Um trotzdem einen Teil der Qualitätssicherung automatisieren zu können, werden für alle Teile des Programms, bei denen keine solche Abhängigkeit besteht, automatische Tests eingesetzt. Darunter fallen insbesondere die Projekte «Common» und «Database», da diese komplett plattformunabhängig sind.

Für alle so getesteten Teile wird eine Test Coverage von mindestens 60% angestrebt. Als ideale Abdeckung wird eine Coverage von 80% oder mehr angesehen. Dabei liegt der Schwerpunkt auf dem Testen der wichtigen Programmlogik, das heisst, reine Datenhalterobjekte ohne Logik, Exceptions oder Konstruktoraufrufe erhalten nur zweite Priorität und obliegen nicht dem oben angegebenen Minimum von 60% Abdeckung. Ausserdem werden mögliche geworfene Exceptions ebenfalls geprüft.

**Technisches**

Für die Implementation der Unit-Tests wurde die NuGet-Library NUnit V3.12.0 [24] sowie die Hilfslibrary FluentAssertions V5.9.0 [25] verwendet. Letztere ermöglicht das Ausformulieren von Testbedingungen über Extension Methods und Call Chaining und offeriert eine praktische Äquivalenzprüfungsmethode.

Zum Zweck der sauberen Gliederung der Tests wurde ein C#-Projekt «Testing» erstellt. In diesem finden sich für alle anderen C#-Projekte, für welche automatische Tests geschrieben wurden, ein Unterordner, der alle diese Tests beinhaltet. So können jeweils ganze Projekte durchgetestet werden.

**Resultate**

Die Abbildung 17 zeigt die Testcoverage am Ende des Projektes. Wie klar ersichtlich ist, laufen alle Tests erfolgreich ab und die abgesteckten Coverage-Ziele wurden eingehalten.

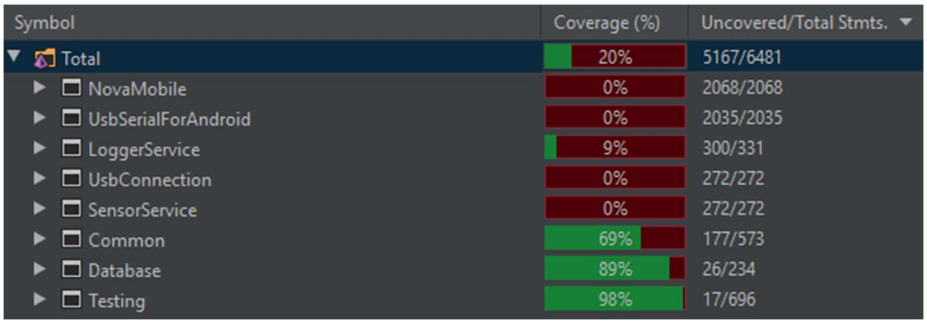


**Abbildung 17: Test-Coverage Gesamtprojekt**

In der Abbildung 18 ist die Testabdeckung in den beiden gut testbaren, weil von Android nur sehr schwach abhängigen Projekten «Common» und «Database» genauer ersichtlich.

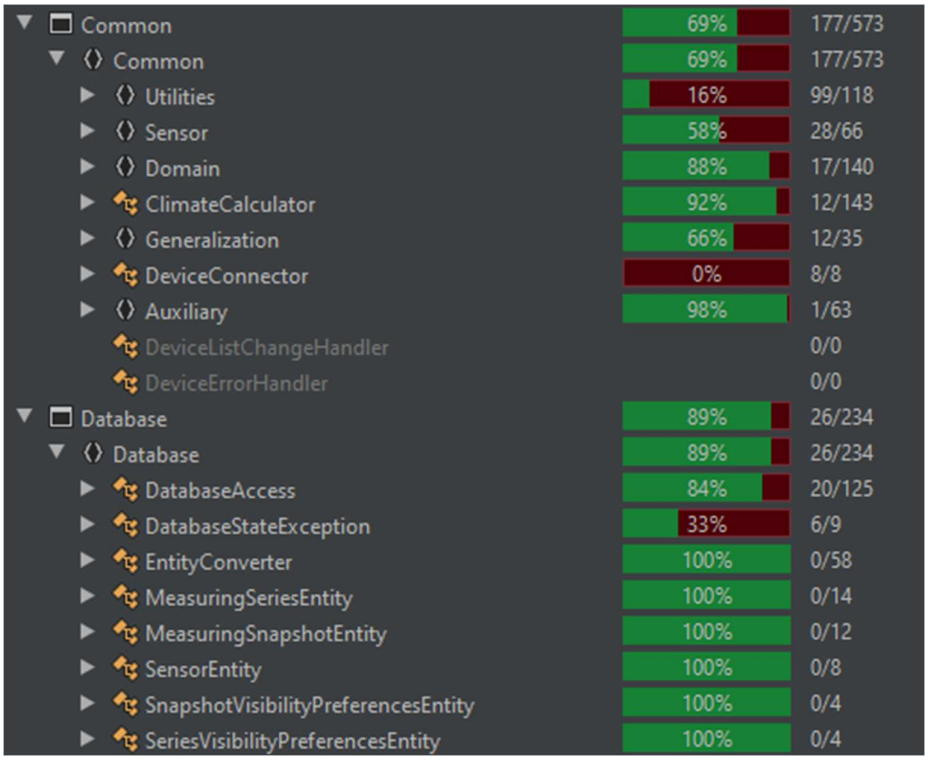


**Abbildung 18: Test-Coverage nicht androidabhängiger Klassen**

### 3.4.3 Manuelle Testverfahren

**Anwendung**

Wie bereits im Kapitel 3.4.2 beschrieben, wurde die Entscheidung gefällt, sich für den Grossteil der von Android direkt oder indirekt abhängigen Klassen auf manuelles Testing zu beschränken.
Das manuelle Testing für das NovaMobile-Projekt besteht somit aus zwei Teilen. Einerseits wurden Integrations- und Systemtests manuell durchgeführt, andererseits wurden zwecks Testing der Anwendbarkeit Usability Tests durchgeführt. Für die Usability-Tests wurden Testprotokolle erstellt, die den Inhalt des Tests und die Erfolgskriterien bestimmen.

**Manuelle Tests**

Manuelle Tests sollen ergänzend zu den automatischen Testverfahren möglichst systemübergreifend angewendet werden, damit Interfaces und schwierig testbare Bereiche kontrolliert werden können. Hierfür wurde die Benutzeroberfläche während der Implementierung fortlaufend von den Entwicklern selbst manuell getestet. Für die Tests wurden Szenarien aus den Use Cases durchgespielt. Die gravierendsten hierbei aufgefallenen Mängel waren:

1. Der Sensor-Service war nach längerem Gebrauch plötzlich nicht mehr verfügbar. Es konnten keine Abfragen zu den Sensoren mehr gemacht werden. Wie sich herausstellte, wurde der Service von Android wegen RAM-Knappheit abgebaut.
2. Je nachdem wie die Aufnahme einer Messreihe gestoppt wurde, blieb die Notification erhalten und wurde nicht geschlossen.
3. Während der Aufnahme einer Messreihe wurde in der App und bei der Notification ein anderer Timer angezeigt. Dieser wurde nicht zentral verwaltet und konnte je nach Laufzeit einen Delay entwickeln.
4. Bei einigen Ansichten wurde nicht auf unvorhergesehen Ausstecken des Sensors reagiert.
5. Durch einen Layoutfehler wurde nur ein Sensor angezeigt, obwohl mehrere angeschlossen waren.
6. Je nachdem wie der Löschvorgang eines Archiv-Objektes abgebrochen wurde, aktualisierte sich die Liste nicht richtig.
7. Die Liste von den auswählbaren Messresultaten wurde nicht richtig aktualisiert. Dadurch war es möglich, Messresultate mehrfach auszuwählen und dupliziert darzustellen.
8. Obwohl alle möglichen Messresultate bereits in der Liste angezeigt wurden, wurde weiterhin ein Plus-Knopf angezeigt.
9. Durch länderspezifische Formatierungsoptionen von Zahlen kam es beim Klimarechner zu Parse-Fehler, weswegen Zahlen mit Nachkommastellen nicht berechnet werden konnten.
10. Beim CSV-Export wurden immer alle Messwerte exportiert, nicht nur die ausgewählten.

Alle diese Punkte wurden fortlaufend während dem Entwicklungsprozess korrigiert und überarbeitet.

**Usability Tests**

Zumal eines der erklärten Primärziele des Projektes die Verbesserung der Usability über die der bestehenden Lösungen war, wurden von uns vor allem in der Refactoring-Phase der Implementation Usability-Tests durchgeführt. Dazu wurden vier Testfälle erstellt, die sich jeweils mit einem Probanden durchspielen lassen, um festzustellen, wie einfach die betreffende Person ans Ziel kam beziehungsweise wo Schwierigkeiten auftraten.

Diese Tests orientieren sich erheblich an den von uns erarbeiteten Use Cases, kombinieren sie aber zum Teil auch, da das Testen auf nur einen einzigen Use Case praxisfremd wäre.

Die Usability Tests sind alle nach der gleichen Struktur aufgebaut. Zuerst wurde definiert, welche Use Cases (und ggf. Extensions davon) der Test überprüfen soll und wann er als erfolgreich gilt. Dann wurde ein kleines Rollenspiel-Szenario erstellt, in das sich der Proband hineinversetzen soll, um den Test zu absolvieren. Diese Szenarien sowie die genauen Tests können im Anhang E eingesehen werden.

Die Usability-Tests wurden von sechs Probanden durchgeführt, da sich bei dieser Anzahl das beste Nutzungs-/Aufwandverhältnis ergibt [34]. Vor dem Usability-Test wurde dem Probanden erklärt, wobei es sich um NovaMobile handelt und was dessen Hauptfunktionen sind.Bei dieser Erklärung wurde absichtlich darauf verzichtet, das von den Entwicklern verwendete Vokabular zu verwenden, da dies einem Nutzer in der Praxis ebenfalls nicht vertraut ist. Für die Erläuterungen und Aufgabenstellung wurden keine Anglizismen verwendet, wie sie im Code üblich sind, damit User nicht gezielt nach genannten Schlüsselwörtern suchen, sondern Eigeninitiative beim Ausprobieren zeigen. Zusätzlich sind alle Probanden gebeten worden, während den Tests bei unbekannten Begriffen nachzufragen.

Sämtliche Teilnehmer haben die Aufgaben erfolgreich und in einem vernünftigen Zeitrahmen ausgeführt. Auf Nachfrage wurden folgende Verbesserungsvorschläge angegeben:

1. Da während des Messvorganges einer Messreihe die Ansicht verlassen werden kann, wurde wegen fehlender Signalisierung vom Probanden angenommen, dass die Messung beim Verlassen der Ansicht beendet wird. Als Verbesserung wurde in der Sensorenübersicht ein zusätzlicher visueller Effekt vorgeschlagen, welcher auf die Aufnahme hindeutet.
2. Da die Einstellungen in einer separaten Ansicht organisiert werden, wurde empfohlen, die aktuell eingestellten Werte in den Aufnahme-Ansichten anzuzeigen. Somit kann man überprüfen, ob die richtigen Einstellungen getätigt sind und muss deshalb nicht jedes Mal in die Einstellungsansicht wechseln.
3. Bei wenigen Probanden war nicht ersichtlich, wie ein Element aus einer Liste gelöscht werden kann. Die seitliche Wischbewegung war ihnen nicht bekannt. Hier wurde eine kleine Einführung beim erstmaligen Öffnen der App vorgeschlagen.
4. Bei der Detail-Ansicht einer Messreihe ist nicht ersichtlich, wie lange die Messung dauerte. Es wird lediglich der Startzeitpunkt der Messung angezeigt. Hier wurde vom Probanden gewünscht, dass die Zeitdauer auch dargestellt werden soll.
5. Beim Aufnehmen von Messreihen kann ein Timer gewählt werden. Die Auswahl hierfür ist vorgegeben. Vom Probanden wurde gewünscht, dass dieser Timer mit Stunden, Minuten und Sekunden selbst ausgewählt werden kann und man nicht auf die Auswahl eingeschränkt ist.
6. Die Analogie der Bilder mit der Fotokamera und des Camcorders wurde missinterpretiert. Oftmals haben die Probanden erwartet, dass die Handykamera geöffnet wird. Als Verbesserungsvorschlag wurde hier angegeben, dass zusätzlich zum Icon eine Beschriftung beim Button angezeigt wird.

Anhand der Feedbacks aus den Usability-Tests wurden folgende Entscheidungen getroffen oder Änderungen vorgenommen:

- **Zu Punkt 1:** Da ohne zusätzliche Anpassungen für diesen Vorschlag auf alle nötigen Informationen zugegriffen werden konnte, wurde dieses Feature noch implementiert. In der Liste der verfügbaren Sensoren wird nun bei einer Aufnahme beim entsprechenden Sensor ein roter blinkender Punkt angezeigt. Sobald die Aufnahme beendet ist, verschwindet dieser.
- **Zu Punkt 2:** Da dieser Vorschlag bereits frühzeitig noch während der Feature-Implementierung kam, wurde das Layout noch angepasst und die zusätzlichen Informationen in der Aufnahme-Ansicht integriert.
- **Zu Punkt 3:** Die Implementierung einer Einführung in die App ist aufwändig und wurde wegen niedriger Priorisierung nicht realisiert. Zusätzlich ist zu beachten, dass seitliche Wischbewegungen bei Listen ein weitverbreitetes Feature sind, weshalb man davon ausgehen kann, dass diese Funktion von den meisten Benutzern bereits intuitiv angewendet wird.
- **Zu Punkt 4:** Da dieser Vorschlag erst nach Feature-Freeze eingereicht wurde, ist diese Verbesserung nicht implementiert worden. Dieser Vorschlag kann jedoch ohne allzu grossen Aufwand realisiert werden. Bei der weiterführenden Entwicklung wird empfohlen, dies umzusetzen.
- **Zu Punkt 5:** Dass der Timer individuell nach Bedarf auf die Sekunde genau selbst definiert werden kann, wurde bereits bei der Konzeptionierung so vorgesehen. Jedoch wurde beim MVP in Absprache mit der Novasina AG bewusst nur eine vordefinierte Auswahl von Zeiten für den Timer implementiert. Dieser Entscheid kommt daher, dass eine Liste als Auswahl einfacher und schneller zu implementieren ist und für die meisten Anwendungszwecke völlig ausreicht.
- **Zu Punkt 6:** Wegen Platzknappheit und wegen der Unterstützung verschiedener Sprachen muss die Beschriftung im Button mehrzeilig sein können. Hierfür hätte das Layout überarbeitet werden müssen, was wegen Zeitgründen nicht mehr realisiert wurde. Jedoch ist zu bemerken, dass diese Art von Verwirrung nur beim erstmaligen Gebrauch auftritt. Wenn der Benutzer die Funktion das erste Mal gebraucht und somit die Analogie begriffen hat, sollte dies kein Problem mehr sein. Bei dieser Problematik wird somit ein hoher Lerneffekt erwartet.

## 3.5 Resultate und Weiterentwicklung

### 3.5.1 Resultate

**Zusammenfassung**

Das in Zusammenarbeit mit der Novasina AG erarbeitete Minimum Viable Product (siehe Kapitel 2.4) wurde zum Grossteil erreicht. Gewisse Vorbehalte, was Features betrifft, mussten dabei bedingt durch die zeitsensitive Natur dieser Arbeit allerdings gemacht werden.

Trotz der anfänglich aufgetretenen Schwierigkeiten bei der Kommunikation mit dem Sensor via Gerätetreiber sowie mangelnder Erfahrung im Umgang mit dem Xamarin-Framework wäre die aus dieser Arbeit resultierende Android-Applikation nach einigen minimalen, grösstenteils administrativen Änderungen bereit zum Deployment und Verwendung im praktischen Einsatz.

**Rebranding**

Der Entwicklungstitel dieser Arbeit sowie der daraus entstehenden App lautete «NovaMobile». Dieser Titel, wurde zu Beginn provisorisch definiert. Zu Beginn der Transition-Phase wurde allerdings von der Novasina AG der Wunsch geäussert, die App auf «nSens-Mobile» umzubenennen, da dies besser in ihr Produktportfolio passt.

Aus diesem Grund wurde die App kurz vor Deployment noch umbenannt, was einen sehr kleinen kosmetischen Eingriff darstellt. Der Titel des Projektes lautet allerdings weiterhin «NovaMobile», und auch im Code selbst wird die App weiterhin als NovaMobile geführt.

**Zielerreichung**

Zu den im MVP definierten, erreichten Zielen gehören insbesondere die folgenden:

- Sensoren können via USB-Kabel angeschlossen werden, weitere Anschlussmedien sind implementierbar.
- Live-Daten können von einem Sensor ausgelesen und vom Nutzer eingesehen werden.
- Schnappschuss-Aufnahmen der momentanen Klimadaten eines Sensors können erstellt und im Gerätespeicher abgelegt werden.
- Messreihen können erstellt und im Gerätespeicher abgelegt werden.
- Mehrere Sensoren können gleichzeitig verbunden und ausgelesen werden (mangels Verbindungsmedien erfordert dies einen USB-Hub).
- Fiktive Klimadaten können über den Klimarechner anhand manuell eingegebener Inputparameter errechnet werden.
- Sensoren können angepasste Labels erhalten.
- Gespeicherte Messprozesse können angepasste Labels erhalten.
- Im Gerätespeicher hinterlegte Messprozesse können tabellarisch eingesehen, verwaltet und gelöscht werden.
- Aus gespeicherten Messprozessen kann ein CSV-File erstellt werden, welches dann über die Android-Exportfunktion geteilt werden kann.
- Der Referenzdruck für die Messungen kann individuell eingestellt werden.
- Das Speicherintervall für Messreihen kann individuell eingestellt werden.
- Die Applikation kann in Deutsch oder Englisch angezeigt werden (richtet sich nach Systemeinstellungen), weitere Sprachpakete sind problemlos und ohne Veränderungen am Code hinzufügbar.

Durch die engen Zeitvorgaben mussten allerdings einige weniger prioritäre Ziele ausgelassen werden, darunter die folgenden:

- Die Anzeige der Messreihen erfolgt tabellarisch (Minimum, Maximum, Durchschnitt), jedoch nicht grafisch oder in einem Diagramm. Aus Zeitgründen konnte diese grafische Darstellung nicht mehr implementiert werden.
- Mess- und Speicherintervall werden nicht differenziert. In demselben Intervall, in dem gemessen wird, wird auch gespeichert.
- Die Suche in der Archivansicht ist nicht möglich. Die notwendige Backend-Methode ist zwar implementiert, doch die Elemente des grafischen User Interface, welche den Zugriff auf sie erlauben würden, konnten nicht mehr vor Code Freeze implementiert werden.
- Das Umschalten der Messeinheiten (z.B. Celsius zu Kelvin) ist nicht möglich.
- Das Importieren von Datenbanken eines anderen Gerätes ist nicht möglich

**Ausblick**

Gemäss den Spezialisten der Novasina AG ist die App im Zustand bei Abgabe dieser Arbeit schon beinahe bereit für einen Testlauf im praktischen Gebrauch. Die Feature Completeness hierzu ist schon gegeben, die zwei wichtigsten fehlende Features bleiben die grafische Darstellung der Messreihen (was allerdings durch den CSV-Export und die Verwendung einer Tabellenhaltungssoftware erreicht werden kann) sowie die mangelnde Suche in der Archivansicht, welche mit vergleichsweise mässigem Aufwand noch implementiert werden könnte.

**Signierung**

Ebenfalls notwendig, um die App tatsächlich releasen zu können, wäre es, das APK-File, welches zum Installieren auf einem Android-Gerät vonnöten ist, als Firma mit einem internen Zertifikat zu signieren [35]. Dieser Schritt wurde bewusst nicht durchgeführt, da er beim eigentlichen Publisher der App geschehen sollte. Bei allen bisher erstellten APKs handelt es sich um Debug-Versionen zu Testzwecken, die aus Sicherheitsgründen nicht an die Öffentlichkeit gelangen sollten, da sie nicht von der Novasina AG signiert sind. Nach erfolgter Signierung könnte die App über den offiziellen Google Play Store zu den Bedingungen der Novasina AG verfügbar gemacht werden.

**Neue Technologien**

Darüber hinaus besteht, wie bereits erwähnt, die Möglichkeit, zukünftige Neuentwicklungen der Novasina AG, wie beispielsweise neue Verbindungsmedien und Sensortypen, in die Applikation zu integrieren, um sie langfristig für die Novasina AG tragbar zu machen. Zu den Details dieser Anbindungen siehe Kapitel 3.5.

### 3.5.2 Möglichkeiten der Weiterentwicklung

**Neue Sensortypen**

Wie bereits anfangs erwähnt, war eines der wichtigsten Ziele beim Architekturentwurf und der Implementierung von NovaMobile das einfache Implementieren von neuen Sensortypen. Dementsprechend ist dies eine der wichtigsten Möglichkeiten, wie NovaMobile weiterentwickelt werden kann.

**Neue Verbindungsmedien**

Der zweite grosse erweiterbare Bereich, der von Anfang an feststand, war das Anknüpfen von Sensoren über neue Medien. Insbesondere Bluetooth stand zur Debatte, da die Novasina AG dies im Moment auf Machbarkeit hin untersucht.

**CRC-Check für Sensorantworten**

Die nSens-Fühler der Novasina AG bieten einen integrierten CRC-Check, der mit jedem Sensorresultat in den letzten zwei Bytes mitgeschickt wird und sich auf den von der Novasina AG verwendeten CRC-Algorithmus [36] stützt. Es bestand ein Versuch, diesen zu implementieren, welcher allerdings fehlschlug.

Mangels Zeit und angesichts Features mit höherer Priorität wurde dieser CRC-Check deshalb bis zum Ende der Construction-Phase nicht mehr implementiert, zumal die Tests trotzdem korrekte Werte lieferten. Um sensorseitige Fehler erkennen zu können, sollte er aber noch ergänzt werden.

**Grafische Darstellung von Messreihen**

In der GUI-Konzeption war von Beginn an eingeplant, dass Messreihen als bildlich dargestellter Graph angezeigt werden können. Aus Zeitmangel konnte dies allerdings nicht bis zum Ende des Projektes realisiert werden.

**Archiv-Sortierfunktion**

Gemäss Konzept soll es im Archiv möglich sein, anhand eines Lupen-Buttons gezielt nach Messprozessen zu suchen. Dieses Feature hat es nicht in die zu Ende des Entwicklungsprozesses verfügbare Version geschafft. In der Datenbank sind die benötigten Funktionen bereits vorhanden. Jedoch müsste noch die gesamte Benutzeroberfläche hierfür implementiert werden.

**Verbesserungen gemäss den Usability-Tests**

Anhand der Usability-Test sind folgende Verbesserungsvorschläge für die Weiterentwicklung gemacht worden:

- In der Detailansicht von Messreihen die Zeitdauer anzeigen statt nur den Startzeitpunkt.
- Den Timer beim Aufnehmen von Messreihen individuell auf die Sekunde genau einstellbar machen.
- Kamera- und Camcorder-Button mit einer zusätzlichen Beschriftung versehen.

**Treiber-Verbesserung**

Der Code des von uns verwendeten USB-Gerätetreibers, UsbSerialForAndroid, entspricht nicht den in dieser Arbeit definierten Code-Guidelines und beinhaltet im momentanen Zustand viele Komponenten, die von der Applikation gar nicht verwendet werden. Es empfiehlt sich deshalb, längerfristig eine Überarbeitung dieses Treibers ins Auge zu fassen. Mangels Vertiefung in den Treiber-Code und Zeit existieren dafür allerdings keine konkreten Konzepte.

**Capability-Check**

Die Sensoren der Novasina AG bieten ein sogenanntes Capability-Level an, welches den aktuellen Stand der Firmware widerspiegelt. Sollte es seitens der Novasina AG je zu nicht rückwärtskompatiblen Änderungen kommen, so würden die Sensoren nicht mehr mit dieser Applikation zusammen funktionieren. Aus diesem Grund könnte ein Capability-Check implementiert werden, der im Falle eines Capability-Mismatches den User benachrichtigt und darauf hinweist, ein Update der Applikation oder der Sensor-Firmware durchzuführen.

**Wetterservice-Anbindung**

Während der Elaborationsphase wurde kurzzeitig die Möglichkeit der Anbindung an einen Online-Wetterservice besprochen, um jeweils den aktuellen lokalen Druck abzufragen. Diesem Feature wurde aber so wenig Priorität zugeordnet, dass dessen Implementierung in der Konstruktionsphase nie ein Thema war. Dennoch wäre eine solche Anbindung denkbar, mangels Ausarbeitung des Vorschlags werden unten aber keine konkreten Vorschläge gemacht.

### 3.5.3 Konkretes Vorgehen bei der Weiterentwicklung

**Neue Sensortypen**

Zur konkreten Implementation eines neuen Sensortyps wären konkret die folgenden Schritte notwendig:

- Ein neuer Eintrag für einen Wert im SensorType-Enum muss angelegt werden, der den neuen Sensortyp beschreibt.
- Neue Resultattypen (z.B. CO2-Gehalt und alle daraus errechenbaren Werte) müssen deklariert werden, die von MeasuringResult erben.
- Die Klasse SensorUtilities muss um Einträge für die Sensortyp-Bytesequenz und die primären und sekundären Resultattypen ergänzt werden, damit bei der Abfrage beim Anschluss des neuen Sensors klar wird, welchen Typ er hat und was er liefern kann.
- Neue Stringressourcen für die neuen Resultattypen müssen in den Resource-XML-Files erstellt und im UnitToStringConverter eingebaut werden, damit sie im UI korrekt dargestellt werden können.
- Eine neue MeasurementStrategy muss angelegt werden, um die Resultat-Bytestreams von Measurement-Abfragen auf den neuen Sensor korrekt interpretierten und durch die SensorUtilities in MeasuringResults umwandeln zu können.
- Ein neuer TypeFiller muss angelegt werden, um nachträglich Resultate zu MeasuringProcesses hinzufügen zu können.
- Der ClimateCalculator muss um die neu berechenbaren Werte ergänzt werden.

**Neue Verbindungsmedien**

Um Sensoren über ein neues Medium erreichen zu können, wäre das Folgende notwendig:

- Ein neues Projekt «MediumConnection» (z.B. BluetoothConnection) sollte angelegt werden, da die Medien jeweils in ein eigenes Projekt separiert werden.
- Darin muss eine neue MediumManager-Klasse für das jeweilige Medium implementiert werden, die das Interface IMediumManager umsetzt.
- Ein von der abstrakten DeviceConnector-Klasse ableitender DeviceConnector muss implementiert werden, der mit Sensoren über das neue Medium kommunizieren kann.
- Ein Monitor muss erstellt werden, der das Medium auf neue und bestehende Verbindungen hin prüfen kann und diese dem MediumManager meldet.
- Im SensorService muss das neue Medium in die Liste der zu prüfenden Medien aufgenommen werden, indem bei der Konstruktion des Service ein entsprechender MediumManager erstellt wird.

**CRC-Check für Sensorantworten**

Der CRC-Check wurde als statische Methode in der SensorUtilities-Klasse implementiert und wird auch bereits aufgerufen, wo dies notwendig ist. Mangels Zeit und Priorität zur Implementierung gibt die Methode zur Zeit der Abgabe allerdings immer true zurück und wurde deshalb mit einem To-Do-Tag versehen.

Die Implementierung des CRC-Checks müsste gemäss dem von der Novasina AG verwendeten CRC-Algorithmus [29] [36] erfolgen. Dies würde die Erkennung von Fehlern im Sensor und die Behandlung auf der App-Seite ermöglichen.

# 4 Softwaredokumentation

## 4.1 Installation

**APK**

Die NovaMobile-App wird über ein gewöhnliches .apk-File, ein Android Application Package, auf einem Gerät installiert. Hierzu muss die Datei auf das Gerät geladen werden, beispielsweise via USB-Transfer, Messenger-App oder E-Mail. Dann kann sie geöffnet und somit installiert werden.

**Voraussetzungen**

Die App wurde auf der Mindest-API-Version 26 entwickelt. Ziel-API-Version ist 28. Dies bedeutet, dass ein Android-Mobilgerät mindestens die Android-Version Oreo (8.0.0, API 26) installiert haben muss, um die App überhaupt starten zu können. Für eine ideale Laufzeit-Umgebung wird die Version Pie (9.0, API 28) empfohlen.

**Sicherheit**

Da es sich bei der vorliegenden Version der App noch nicht um einen offiziellen Release handelt, kann es beim Installiervorgang unter Umständen zu einer Warnung kommen. Dies liegt daran, dass das AKP nur mit einem temporären Zertifikat signiert wurde und nicht aus dem offiziellen Google Play Store stammt. Diese Warnung kann in den Geräteeinstellungen unter «Sicherheit» ausgeschalten werden.

## 4.2 Tutorial / Benutzerhandbuch

**Hub**

Nach dem Start der App wird das Hauptmenü angezeigt. Hier kann über drei Tabs am unteren Bildschirmrand zwischen den wichtigsten Funktionsgruppen der App navigiert werden:

- Messen: In der Messen-Ansicht wird eine Übersicht aller verfügbaren Sensoren angezeigt. Diese Ansicht wird beim Start der App automatisch selektiert.
- Rechner: In der Klimarechner-Ansicht können fiktive Klimadaten eingegeben und sekundäre Werte errechnet werden.
- Archiv: Die Archiv-Ansicht bietet einen Überblick über alle abgespeicherten Messdaten.

Diese drei Ansichten werden nachfolgend im Detail erklärt.
Darüber hinaus können über das Dropdown-Menü in der oberen rechten Bildschirmecke aller Hub-Ansichten die Einstellungen sowie Informationen über die vorliegende App angezeigt werden.

**Messen**

Die Messen-Ansicht erlaubt es, alle momentan angeschlossenen Sensoren mit Seriennummer und Label einzusehen sowie Live-Daten abzurufen, die gerade gemessen werden.

Durch Antippen des Pfeil-Icons neben einem Sensoren-Eintrag in der Liste klappt dieser auf und zeigt seine Live-Daten an. Es werden all jene Grössen angezeigt, die vom Sensor direkt gemessen werden, aber keine Sekundärgrössen. Durch ein weiteres Antippen des Pfeil-Icons lässt sich der Eintrag wieder schliessen.

Im ausgeklappten Eintrag erscheinen ausserdem zwei weitere Buttons. Durch Antippen der Fotokamera wird die Schnappschuss-Ansicht zum betreffenden Sensor aufgerufen, durch Antippen der Videokamera gelangt man in die Messreihen-Ansicht. Hierbei wird die Analogie gezogen, dass das Aufnehmen eines Schnappschusses einem Foto entspricht, das Aufnehmen einer Messreihe einem Video.

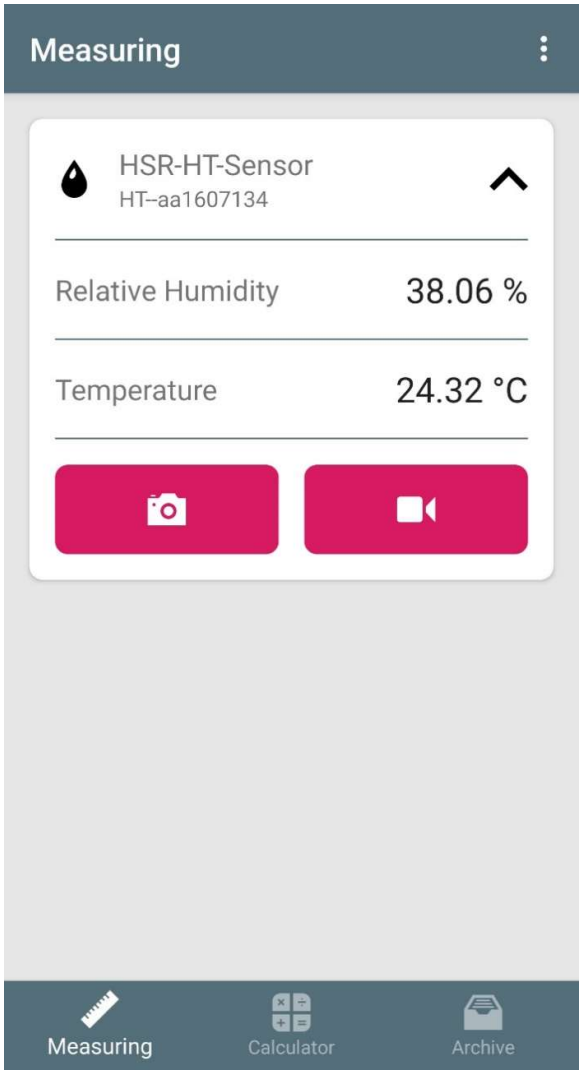


**Abbildung 19: Messen-Ansicht**

**Rechner**

Die Rechner-Ansicht erlaubt das Nutzen der App als Klimarechner. Es können fiktive Messdaten eingegeben werden, um daraus sekundäre Werte auszurechnen.

Durch Antippen der blaugrünen Schaltfläche «Erwünschter Wert» kann der zu errechnende Wert aus einer Popup-Liste ausgewählt werden. Im Feld darunter werden dann alle Werte angezeigt, die bekannt sein müssen, um den erwünschten Wert zu berechnen. Sind alle notwendigen Angaben ausgefüllt, wird das Resultat an unterster Stelle angezeigt.

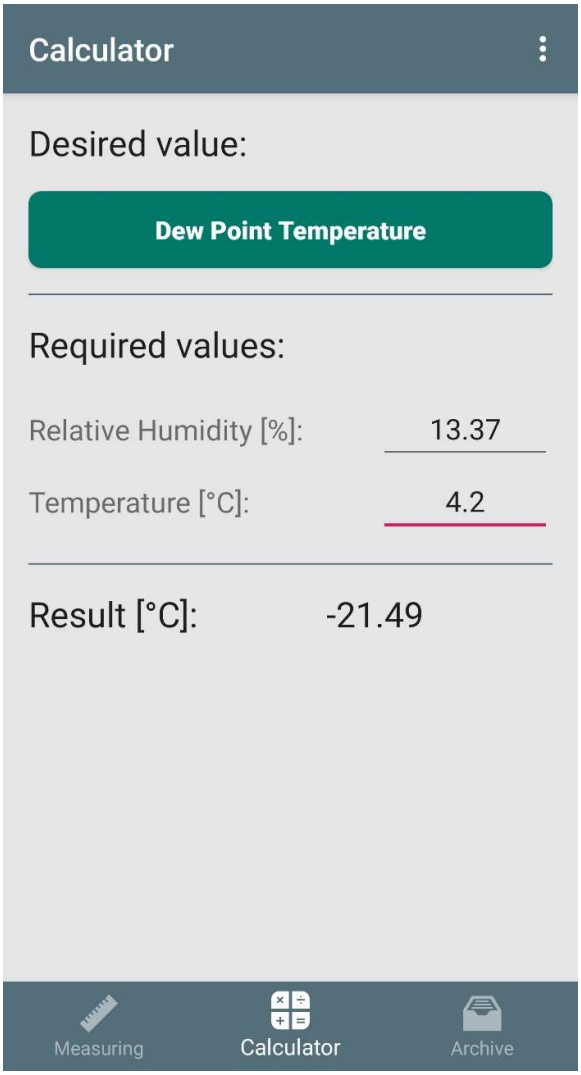


**Abbildung 20: Klimarechner-Ansicht**

**Archiv**

Die Archiv-Ansicht gewährt Einsicht in sämtliche im Gerätespeicher hinterlegten Messprozesse. Jeder davon wird in der Liste aufgeführt, sortiert nach Aufnahmedatum (neueste Aufnahmen zuoberst).

Pro Eintrag angezeigt werden das Label, mit dem die Aufnahme versehen wurde, die ID des Sensors, von dem sie stammt, die Aufnahmezeit sowie der Aufnahme-Typ (Schnappschuss oder Messreihe) und Sensortyp (zur Zeit der Abgabe nur Feuchtigkeit/Temperatur) als Icons.

Durch Antippen eines Eintrages kann der dazugehörige Detailansichts-Bildschirm aufgerufen werden, wo genauere Informationen ersichtlich sind.

Einträge lassen sich durch eine Wisch-Geste nach links oder rechts aus der Liste und somit dem Gerätespeicher löschen.

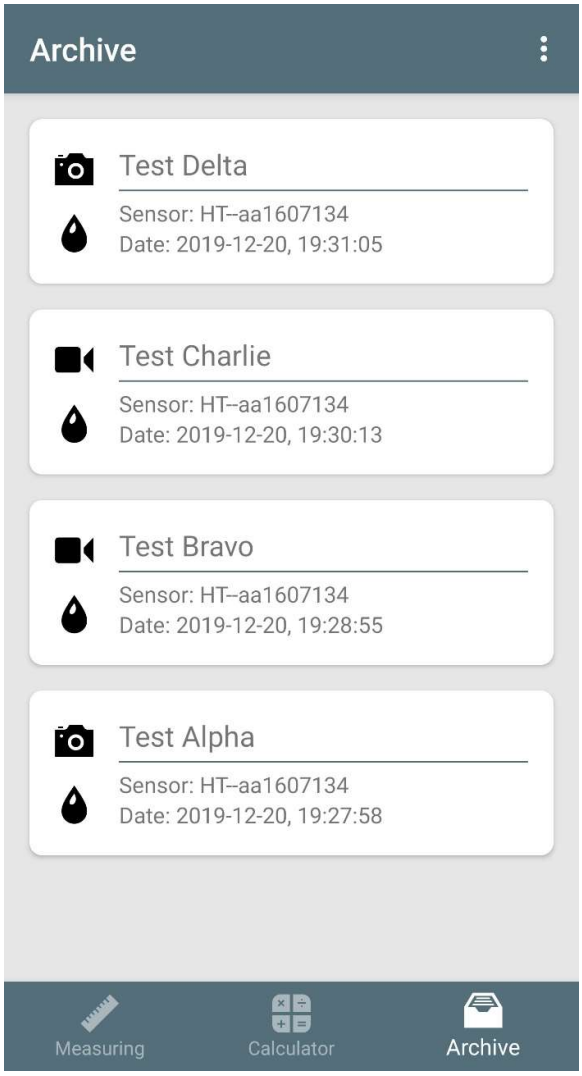


**Abbildung 21: Archiv-Ansicht**

**Schnappschuss-Ansicht**

Die Schnappschuss-Ansicht erlaubt es, einen Schnappschuss, also eine Momentaufnahme der aktuellen Klimabedingungen, aus den Daten eines bestimmten Sensors zu erstellen. Zuoberst in dieser Ansicht erscheint der Sensor, mit dem gemessen wird, mit Seriennummer (ID) und Label. Letzteres lässt sich durch Antippen des Stift-Icons bearbeiten. Ausserdem wird der Referenzdruck, der für den Schnappschuss verwendet wird, angegeben.

Darunter wird eine Liste aller Messgrössen angezeigt, die als Teil des Schnappschusses abgespeichert werden. Standardmässig erscheinen in dieser Liste nur die Primärwerte, also diejenigen Grössen, die der Sensor direkt liefert (für einen HT-Sensor z.B. Feuchtigkeit und Temperatur). Durch Antippen des blaugrünen Plus-Buttons lassen sich aber Sekundärwerte aus einem Popup auswählen, die dann berechnet werden, sobald der Schnappschuss erstellt wurde. Wurden versehentlich zu viele Werte hinzugefügt, lassen sie sich durch Wischen nach links oder rechts wieder entfernen.

Tatsächlich erstellen lässt sich der Schnappschuss durch das Antippen des magentafarbenen Kamera-Buttons am unteren Bildschirmrand. Sobald dieser gedrückt wird, wird der Schnappschuss erstellt, ins Archiv abgelegt und direkt die Detailansicht dazu aufgerufen, damit die gespeicherten Werte eingesehen werden können.

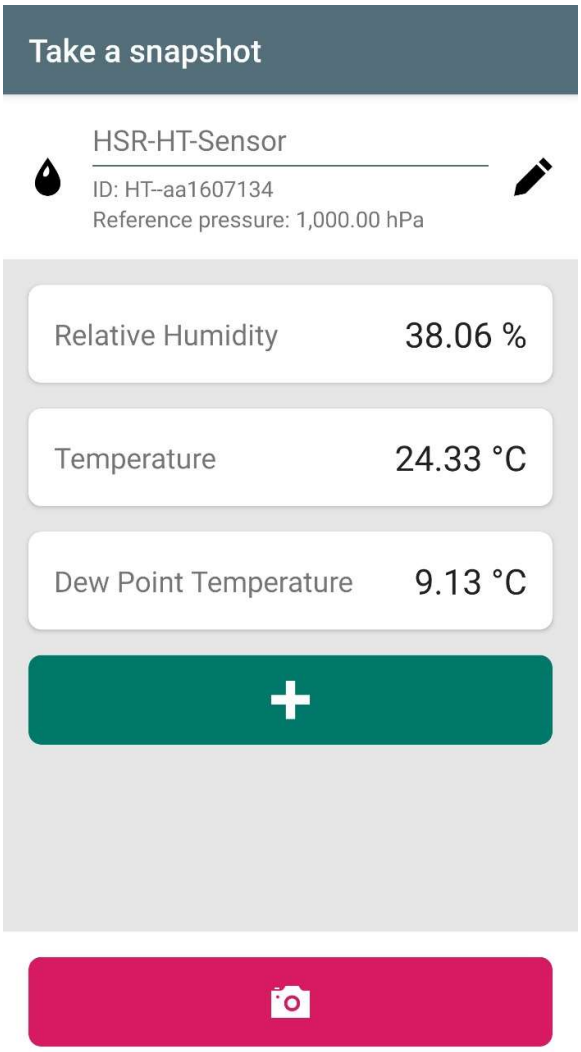


**Abbildung 22: Schnappschuss-Ansicht**

**Messreihen-Ansicht**

Die Messreihen-Ansicht ermöglicht es, von einem Sensor eine Messreihe aufzunehmen. Wie bereits in der Schnappschuss-Ansicht werden am oberen Bildschirmrand die Informationen zum aktuellen Sensor angezeigt. In dieser Ansicht kommt dazu noch die Anzeige des Speicherintervalls, in dem Messungen zur Messreihe hinzugefügt werden.

Genau wie in der Schnappschuss-Ansicht werden darunter die zur Erfassung ausgewählten Messgrössen angezeigt, und es können neue hinzugefügt und bestehende entfernt werden. Neu ist die Anzeige der statistischen Werte Minimum, Durchschnitt und Maximum, die zu jeder ausgewählten Messgrösse ausgerechnet und angezeigt werden, sobald eine Messreihe gestartet wird.

Am unteren Bildschirmrand wird das Messreihen-Kontrollpanel angezeigt. Hier kann über die Stoppuhren-Schaltfläche das Timer-Menü aufgerufen werden, in welchem eine festgelegte Zeitdauer für die Messreihe ausgewählt werden kann. Wird der Eintrag «Kein Timer» ausgewählt, so läuft die Messung, bis sie manuell unterbrochen wird.

Mit dem Aufnehmen-Knopf unten rechts im Kontrollpanel kann eine Messreihe gestartet werden. Dies führt dazu, dass gewisse Auswahlen, wie z.B. das Hinzufügen von neuen Messwerten oder das Verändern des Timers, ausgegraut werden und nicht mehr verfügbar sind, solange die Messreihen-Aufnahme läuft. Beim Beenden einer Messreihe wird, wie beim Aufnehmen eines Schnappschusses, direkt die Detailansicht zur neu erstellten Messreihe angezeigt.

Während eine Messreihe läuft, kann die Messreihen-Ansicht verlassen und die App wie üblich navigiert oder sogar über den Home-Button minimiert werden. Die Messreihe wird dann im Hintergrund weiter aufgenommen. Dass gerade eine Messreihe läuft, ist dadurch ersichtlich, dass ein Icon in der Android-Taskleiste angezeigt wird. Wird das Taskmenü durch Herunterziehen geöffnet, erscheint dort eine Notifikation, in der der Sensor sowie der Timer angezeigt werden. Durch Ziehen an der Notifikation kann sie ausgeklappt werden und zeigt dann eine Liste der erfassten Live-Messdaten an. Darüber hinaus kann die Messung durch den Stop-Button zuunterst in der Notifikation beendet werden, wodurch sie automatisch abgespeichert wird.

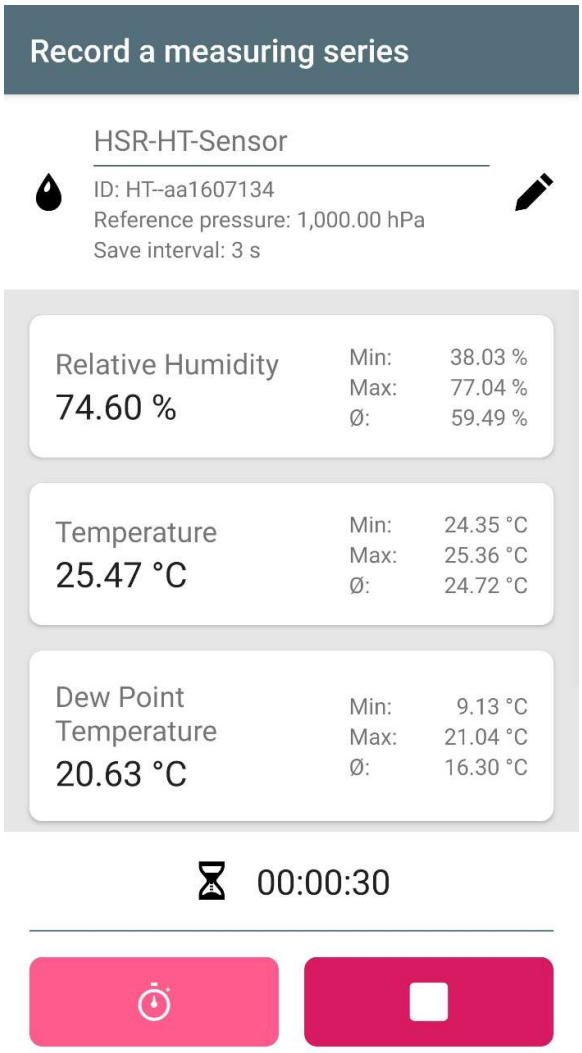


**Abbildung 23: Messreihen-Ansicht**

**Detailansicht**

Die Detailansicht erlaubt das Einsehen genauerer Informationen zu abgespeicherten Schnappschüssen oder Messreihen. Am oberen Bildschirmrand erscheinen die wichtigsten Informationen zum Messprozess, darunter das Label, welches durch Antippen des Stift-Icons bearbeitet werden kann, der Sensor, mit dem die Messung gemacht wurde, das Aufnahmedatum sowie der Referenzdruck.

Darunter erscheint die bereits aus der Schnappschuss- oder Messreihenansicht bekannte Liste der Messgrössen. Auch in dieser Ansicht lassen sich noch neue Grössen hinzufügen, die aus den bereits abgespeicherten Primärgrössen errechnet werden können, oder unerwünschte Einträge durch eine Wischgeste aus der Liste streichen.

Zu beachten ist, dass Daten nicht durch Wischen aus der Liste permanent vom Gerätespeicher gelöscht werden können, da es jederzeit möglich ist, alle entfernten Einträge wieder hinzuzufügen. Es können also in dieser Ansicht keine Daten permanent verloren gehen, lediglich versteckt werden.

Am unteren Bildschirmrand findet sich der Export-Button. Er erlaubt es, den Messprozess als CSV-Datei (Comma Separated Values) zu exportieren und zu teilen. Ins CSV aufgenommen werden dabei all jene Werte, die auch in der Liste angezeigt werden. Zum Teilen stehen die von Android gegebenen Optionen, wie z.B. Mail, zur Verfügung.

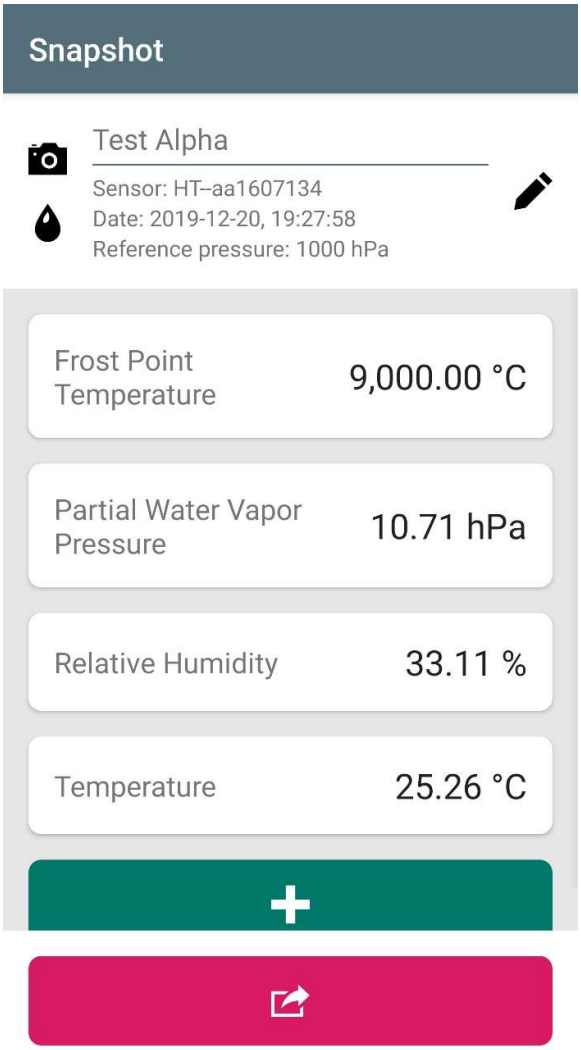


**Abbildung 24: Detail-Ansicht**

**Einstellungen**

Im Einstellungsmenü können benutzerdefinierte Parameter für die gesamte App angepasst werden. Zum Zeitpunkt der Abgabe dieser Arbeit sind dies der Referenzdruck in Hektopascal und das Messintervall in Sekunden. Der Referenzdruck wird jeweils mit der Messung abgespeichert und ist notwendig, um nachträglich sekundäre Werte ausrechnen zu können. Das Messintervall definiert, wie viele Sekunden zwischen den einzelnen Messpunkten gewartet werden soll.

Darüber hinaus lässt sich mit dem Knopf «Alle Daten löschen» die komplette Datenbank der App leeren und auf den Ausganszustand zurücksetzen. Dazu gehören bekannte Sensoren und aufgenommene Schnappschüsse und Messreihen. Einmal gelöscht, kann dieser Vorgang nicht mehr rückgängig gemacht werden.

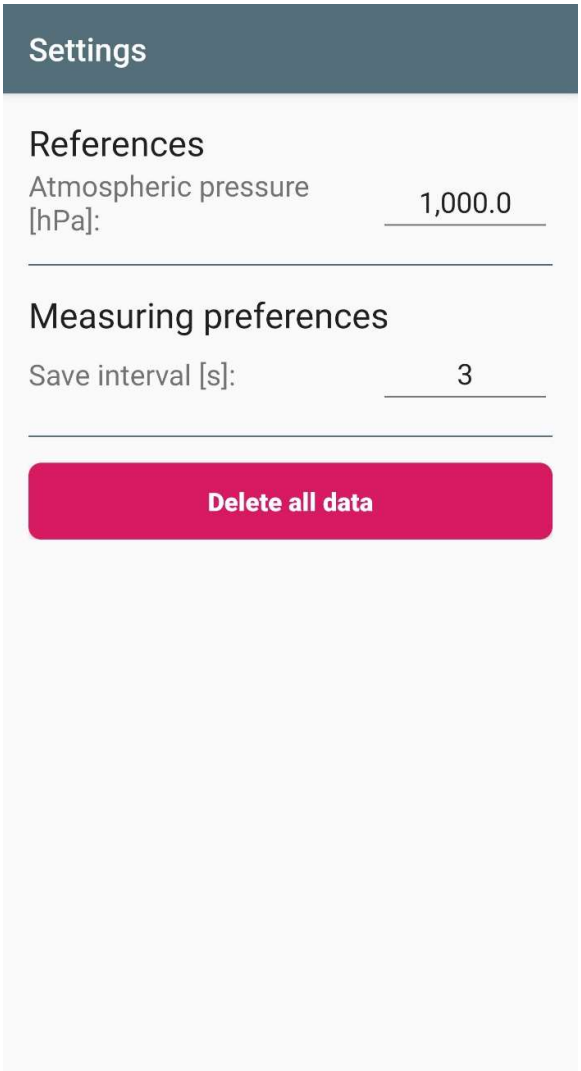


**Abbildung 25: Einstellungen-Ansicht**

# 5 Literatur und Quellenverzeichnis

**Herkunft der Vorlage**

Das Dokument wurde auf der Basis einer Vorlage für Technische Berichte erstellt. Die Vorlage ist ein Element des „Werkzeugkastens Technische Berichte“ der Hochschule für Technik Rapperswil. Sie orientiert sich an Prinzipien des Strukturierten Schreibens.

[1] Novasina, «nSens-HT-ENS,» Novasina AG, [Online]. Available: https://www.novasina.ch/produkt/nsens-ht-ens/. [Zugriff am 3 Januar 2020].

[2] Novasina, "ClimMate," Novasina AG, [Online]. Available: https://www.novasina.ch/produkt/climmate/. [Accessed 3 Januar 2020].

[3] C. Furrer, "Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik," HSR Hochschule für Technik Rapperswil, Rapperswil, 2019.

[4] Novasina, "QuantaDat / nSens," Novasina AG, [Online]. Available: https://www.novasina.ch/produkt/quantadat-nsens/. [Accessed 3 Januar 2020].

[5] Hostingtribunal, "Mobile and Desktop Operating Systems Market Share," [Online]. Available: https://hostingtribunal.com/blog/operating-systems-market-share/. [Accessed 3 Januar 2020].

[6] A. Holst, "Mobile operating systems' market share worldwide from January 2012 to July 2019," Statista Inc., 13 September 2019. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. [Accessed 3 Januar 2020].

[7] Google Developers, "Documentation for app developers," Google LLC, [Online]. Available: https://developer.android.com/docs. [Accessed 3 Januar 2020].

[8] Apple, "Apple Developer Documentation," Apple Inc., [Online]. Available: https://developer.apple.com/documentation/. [Accessed 3 Januar 2020].

[9] Statista Research Department, "Apple iPhone and Android smartphone average selling price from 2008 to 2016," Statista Inc., 24 Mai 2018. [Online]. Available: https://www.statista.com/statistics/612937/smartphone-average-selling-price-iphone-and-android/. [Accessed 3 Januar 2020].

[10] Stack Overflow, "Developer Survey Results 2017," Stack Exchange Inc., [Online]. Available: https://insights.stackoverflow.com/survey/2017#technology-_-most-loved-dreaded-and-wanted-frameworks-libraries-and-other-technologies. [Accessed 3 Januar 2020].

[11] Stack Overflow, "Developer Survey Results 2018," Stack Exchange Inc., [Online]. Available: https://insights.stackoverflow.com/survey/2018#technology-_-most-loved-dreaded-and-wanted-frameworks-libraries-and-tools. [Accessed 3 Januar 2020].

[12] Stack Overflow, "Developer Survey Results 2019," Stack Exchange Inc., [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology-_-other-frameworks-libraries-and-tools. [Accessed 3 Januar 2020].

[13] Unity, "Unity Real-Time Development Platform," Unity Technologies, [Online]. Available: https://unity.com/. [Accessed 3 Januar 2020].

[14] Xamarin, "Xamarin | Open-source mobile app platform for .NET," Xamarin Inc., [Online]. Available: https://dotnet.microsoft.com/apps/xamarin. [Accessed 3 Januar 2020].

[15] React Native, "React Native · A framework for building native apps using React," Facebook Inc., [Online]. Available: https://facebook.github.io/react-native/. [Accessed 3 Januar 2020].

[16] Flutter, "Flutter - Beautiful native apps in record time," Google LLC, [Online]. Available: https://flutter.dev/. [Accessed 3 Januar 2020].

[17] GitHub, "Electron," GitHub Inc., [Online]. Available: https://electronjs.org/. [Accessed 3 Januar 2020].

[18] Future Technology Devices International Ltd., "FTDI Chip," 24 Juni 2019. [Online]. Available: https://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf. [Accessed 3 Januar 2020].

[19] C. Miller, "UsbSerialForAndroid," [Online]. Available: https://github.com/anotherlab/UsbSerialForAndroid. [Accessed 3 Januar 2020].

[20] K. Morich, "usb-serial-for-android," [Online]. Available: https://github.com/mik3y/usb-serial-for-android. [Accessed 3 Januar 2020].

[21] Google Developers, "Intents and Intent Filters," Google LLC, [Online]. Available: https://developer.android.com/guide/components/intents-filters. [Accessed 3 Januar 2020].

[22] J. Newton-King, "Newtonsoft.Json," Newtonsoft, [Online]. Available: https://www.nuget.org/packages/Newtonsoft.Json/. [Accessed 3 Januar 2020].

[23] F. Krueger, "sqlite-net," [Online]. Available: https://www.nuget.org/packages/sqlite-net/. [Accessed 3 Januar 2020].

[24] C. Poole and R. Prouse, "NUnit," [Online]. Available: https://www.nuget.org/packages/NUnit/. [Accessed 3 Januar 2020].

[25] D. Doomen and J. Nyrup, "FluentAssertions," [Online]. Available: https://www.nuget.org/packages/FluentAssertions/. [Accessed 3 Januar 2020].

[26] Google Developers, "Introduction to Activities," Google LLC, [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities. [Accessed 3 Januar 2020].

[27] Google Developers, "Fragments," Google LLC, [Online]. Available: https://developer.android.com/guide/components/fragments. [Accessed 3 Januar 2020].

[28] Google Developers, "Services overview," Google LLC, [Online]. Available: https://developer.android.com/guide/components/services. [Accessed 3 Januar 2020].

[29] D. Schnyder and T. Buchli, "Simple Protocol - nSens OEM nBus Extension," Novasina AG, Lachen, 2017.

[30] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[31] M. Potel, 1996. [Online]. Available: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf. [Accessed 3 Januar 2020].

[32] J. Smith, "Patterns - WPF Apps With The Model-View-ViewModel Design Pattern," Microsoft Co., Februar 2009. [Online]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern. [Accessed 3 Januar 2020].

[33] D. Luberda, "Simple, fast and lightweight MVVM Framework for Xamarin.Forms with fluent API," [Online]. Available: https://github.com/xamvvm/xamvvm. [Accessed 3 Januar 2020].

[34] J. Nielsen und T. Landauer, «A mathematical model of the finding of usability problems,» in *INTERCHI '93 Conference on Human Factors in Computing Systems*, Morristown, NJ 07962-1920, USA, 1993.

[35] Google Developers, "Sign your app," Google LLC, [Online]. Available: https://developer.android.com/studio/publish/app-signing. [Accessed 3 Januar 2020].

[36] R. O'Leary, "Cyclic Redundancy Check," 10 Oktober 1996. [Online]. Available: https://pdc.ro.nu/crc.html. [Accessed 3 Januar 2020].

[37] D. Keller, «Software Engineering 1 v1.5,» HSR Hochschule für Technik Rapperswil, Rapperswil, 2017.

[38] «Richtlinien des Hochschulrates für die koordinierte Erneuerung der Lehre an den universitären Hochschulen der Schweiz im Rahmen des Bologna-Prozesses,» 28 Mai 2015. [Online]. Available: https://www.admin.ch/opc/de/classified-compilation/20150869/index.html. [Zugriff am 22 September 2019].

[39] Clockyfy, "Clockify - 100% Free Time Tracking Software," COING Inc., [Online]. Available: https://clockify.me/. [Accessed 3 Januar 2020].

[40] Google, "Google Java Style Guide," Google LLC, [Online]. Available: https://google.github.io/styleguide/javaguide.html. [Accessed 3 Januar 2020].

[41] Google, "Google JavaScript Style Guide," Google LLC, [Online]. Available: https://google.github.io/styleguide/jsguide.html. [Accessed 3 Januar 2020].

[42] D. Doomen, "C# Coding Guidelines," Aviva Solutions, [Online]. Available: https://csharpcodingguidelines.com/. [Accessed 3 Januar 2020].

[43] K. Kawaguchi, "Jenkins, Build great things at any scale," [Online]. Available: https://jenkins.io/. [Accessed 3 Januar 2020].

[44] Bamboo, "Bamboo Continuous Integration and Deployment Build Server," Atlassian Corporation Plc, [Online]. Available: https://www.atlassian.com/software/bamboo. [Accessed 3 Januar 2020].

[45] TeamCity, "TeamCity: the Hassle-Free CI and CD Server by JetBrains," JetBrains s.r.o., [Online]. Available: https://www.jetbrains.com/teamcity/. [Accessed 3 Januar 2020].

[46] Travis CI, "Travis CI - Test and Deploy Your Code with Confidence," Idera Inc., [Online]. Available: https://travis-ci.org/. [Accessed 3 Januar 2020].