

**Technisches Konzept**

**Projektname: M-Wetter**

**Version:** 0.1

**Status:** in Arbeit

**Datum:** 07.07.2017

**Autor:** Sonja Wasenmiller und Tobias Gilnhammer

Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc487196756)

[1. Einführung und Ziele / Fachliche Anforderungen 3](#_Toc487196757)

[1.1 Aufgabenstellung 3](#_Toc487196758)

[1.2 Qualitätsziele 3](#_Toc487196759)

[1.3 Stakeholder 3](#_Toc487196760)

[2. Randbedingungen 4](#_Toc487196761)

[2.1 Technische Randbedingungen 4](#_Toc487196762)

[3. Kontext 5](#_Toc487196763)

[3.1 Externe Schnittstellen 6](#_Toc487196764)

[3.1.1 Externe Schnittstelle 1 6](#_Toc487196765)

[3.1.2 Externe Schnittstelle 2 6](#_Toc487196766)

[3.1.4 Externe Schnittstelle 3 7](#_Toc487196767)

[4. Lösungsstrategie 8](#_Toc487196768)

[5. Bausteinsicht 9](#_Toc487196769)

[5.1 Ebene 1 9](#_Toc487196770)

[5.1.1 Bausteinname 1 (BlackBox-Beschreibung) 9](#_Toc487196771)

[5.2 Ebene 2 10](#_Toc487196772)

[5.2.1 Bausteinname 1 (Whitebox-Beschreibung) 10](#_Toc487196773)

[6. Laufzeitsicht 11](#_Toc487196774)

[7. Datensicht 12](#_Toc487196775)

[8. Verteilungssicht 13](#_Toc487196776)

[9. Konzepte 14](#_Toc487196777)

[9.1 Frameworks und Entwurfsmuster 14](#_Toc487196778)

[9.1.1 Frameworks 14](#_Toc487196779)

[9.1.2 Entwurfsmuster 14](#_Toc487196780)

[9.2 Persistenz 14](#_Toc487196781)

[9.3 Benutzungsoberfläche 14](#_Toc487196782)

[10. Qualitätsszenarien 15](#_Toc487196783)

[10.1 Bewertungsszenarien 15](#_Toc487196784)

[11. Glossar 16](#_Toc487196785)

# Einführung und Ziele / Fachliche Anforderungen

## Aufgabenstellung

Erstellung einer Wetterstation als prototypische Anwendungsentwicklung mit Java

Technologie.

## Qualitätsziele

* Entwicklung einer Wetterstation, die Wetterdaten wie Temperatur, Luftdruck, Helligkeit und Luftfeuchte misst und darstellt.
* Eine Verfügbarkeit von 24 Stunden am Tag, sieben Tage die Woche.
* Eine Schulungszeit von unter 10 Minuten die Benutzer benötigen, um eigenständig mit der Software arbeiten zu können
* Eine größtmögliche Testbarkeit ist zu gewährleisten um den Testaufwand so gering wie möglich zu halten.

## Stakeholder

|  |  |  |
| --- | --- | --- |
| **Name** | **Rolle** | **Definition** |
| Tobias Gilnhammer | Projektleiter | Koordination des Projektteammitglieder |
| Behnam Tajedini und Matthias Jeschke | Projektauftraggeber | Beauftragung und Unterstützung des Projektteams |
| Sinan Özturan | Designer | Designt die Masken am Desktop sowie am LCD-Bildschirm |
| Tarek Lutz | Entwickler | Entwickelt die Software für die Wetterstation |
| Tobias Gilnhammer, Tarek Lutz, Sinan Özturan und Sonja Wasenmiller | Tester | Die Tester testen die Software auf Fehler und geben Verbesserungsvorschläge ab. |
| Behnam Tajedini und Matthias Jeschke | Anwender | Nutzt die Anwendung |
| Tarek Lutz Tobias Gilnhammer Sinan Özturan Sonja Wasenmiller | Projektteammitglieder | Bearbeiten das Projekt, Erstellen die Anwendung |

# Randbedingungen

Im Wiki unter <https://github.com/matthjes/FISI-WS/wiki> sind alle Informationen eingetragen die das Projekt vereinfachen und an die sich grob gehalten werden soll. Eigeninitiativen der Entwickler zur Erfüllung der Anforderungen sind gerne gesehen.

## Technische Randbedingungen

|  |  |
| --- | --- |
| Hardware-Vorgaben | |
| Wetterstation von Tinkerforge | vollwertige Open Source Wetterstation. Sie misst Temperatur, Luftfeuchtigkeit, Luftdruck und Helligkeit mit hoher Präzision. |
| Raspberry Pi | Mini Computer auf dem ein Deamon zum Abgreifen der Wetterdaten von der Wetterstation läuft. |
| Windows/ OSX Rechner | Betrieb der Lösung auf einem Marktüblichen Standardrechner mit Microsoft Windows oder Apple OSX als Betriebssystem. |
| Software-Vorgaben | |
| Fremdsoftware frei verfügbar | Falls zur Lösung Fremdsoftware hinzugezogen wird (z.B. grafisches Frontend), sollte diese idealerweise frei verfügbar und kostenlos sein. |
| Konfigurations- und Versionsverwaltung | Konfigurations- und Versionsverwaltung über Git und Github. |
| Programmiervorgaben | |
| Backend | Erstellung von Java-Quelltexten in der IDE IntelliJ. |
| Frontend | Erstellung von Masken mittels JavaFX und SceneBuilder |

1. Absprache mit Entwickler

|  |  |
| --- | --- |
| Analyse- und Entwurfsmethoden | Objektorientierte oder strukturierte Methoden? |
| Datenstrukturen | Vorgaben für bestimmte Datenstrukturen, Schnittstellen zu bestehenden Datenbanken oder Dateien |
| Programmierschnittstellen | Schnittstellen zu bestehenden Programmen |
| Programmiervorgaben | Programmierkonventionen, fester Programmaufbau |
| Technische Kommunikation | Synchron oder asynchron, Protokolle |

# Kontext

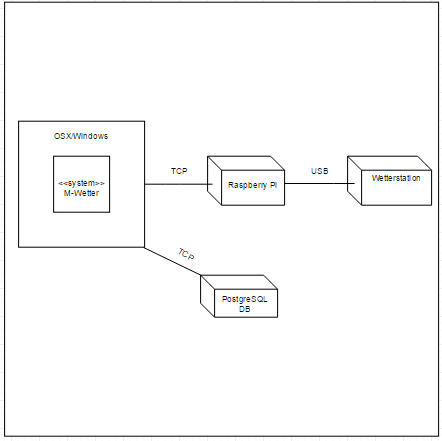


Bild: Technische Interaktion des Systems mit den anderen Komponenten.

Die Wetterstation ist mittels USB mit einem Raspberry Pi verbunden. Auf diesem läuft ein   
Deamon mit dem die Wetterdaten von der Wetterstation abgegriffen werden. Diese Wetterdaten werden lokal auf einer PostgreSQL Datenbank gespeichert und werden von der Anwendung aus der Datenbank ausgelesen. Anschließend werden die Wetterdaten auf dem Desktop und auf dem LCD Bildschirm der Wetterstation angezeigt.

## Externe Schnittstellen

### Externe Schnittstelle 1

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | RaspWetter |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Wetterdaten werden von der Wetterstation auf den Raspberry übertragen |

### Externe Schnittstelle 2

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | <Name der Schnittstelle> |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Wetterdaten werden von einem Brick-Deamon auf dem Raspberry zur Verfügung gestellt und von einem Rechner abgerufen. |

### Externe Schnittstelle 3

#### Identifikation der Schnittstelle

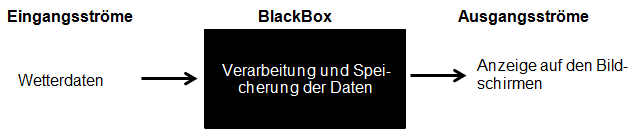
|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | DB |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Datenaustausch zwischen Datenbank und Client |

# Lösungsstrategie

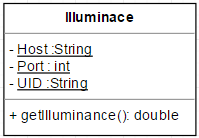
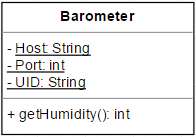
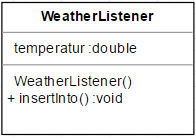
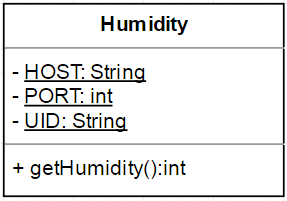
Es gibt 2 Anwendungen. Die erste Anwendung dient dazu, die Wetterdaten auszulesen und in eine Datenbank zu speichern. Die Anwendung wird gestartet und verbindet sich mit dem Deamon der auf dem Raspberry läuft und ruft die Daten der Wetterstation ab. Der Deamon speichert diese Wetterdaten zwischen, verbindet sich mit der PostgreSQl Datenbank und schreibt die Daten in die Datenbank.

Die zweite Anwendung ist für den Benutzer gedacht. Der Anwender startet die Anwendung, welche sich dann mit der Datenbank verbindet und die erforderlichen Daten abruft. Anschließend werden die Daten in einer GUI ausgegeben.

# Bausteinsicht



## C:\Users\gilnhammer.tobias\Desktop\Masken\Klassendiagramm_LcdAnzeige.PNGEbene 1



### LcdAnzeige

Setzt Buttonlistener und ruft durch Betätigung der Buttons Methoden auf die dann am LCD die entsprechenden Ausgaben liefert.

Die Daten die Ausgegeben werden kommen aus der DB

|  |  |
| --- | --- |
| setLcd() | „Initialisierungsmethode“,mit welcher eine Verbindung mit dem Raspberry Deamon hergestellt wird und Buttonlistener gesetzt werden. |
| getData(int i) | Verbindung zur Datenbank , Daten in Arraylisten speichern |
| button1Lcd() | Baut Verbindung zum Deamon auf und gibt Wetterdaten auf dem LCD Bildschirm aus |
| button2Lcd() | Gibt bei Betätigung die Luftfeuchtigkeit aus |
| button3Lcd() | Gibt bei Betätigung aktuelle Infos zu Zeit und Datum aus. |

### Humidity

|  |  |
| --- | --- |
| public int getHumidity() | Verbindung zum Deamon auf, gibt Daten zur Luftfeuchtigkeit aus. |

### Barometer

|  |  |
| --- | --- |
| getAirPressure() | Verbindung Deamon und gibt Daten zum Luftdruck aus. |
| getTemperature() | Verbindung Deamon und gibt Daten zur Temperatur aus. |

### Illuminance

|  |  |
| --- | --- |
| getIllumiance() | Verbindung Deamon und gibt Daten zur Helligkeit aus. |

### WeatherListener

|  |  |
| --- | --- |
| public static void main(String[] args) | Erzeugt Objekte von anderen Klassen. Ruft Methode lcdAnzeige.setlcd() auf und setzt einen Timer dass alle 5 Minuten weatherListener.insertInfo() aufgerufen wird. |
| insertInto() | Baut Verbindung zur DB auf und schreibt die aktuellen Wetterdaten in die DB. |

# 

# Laufzeitsicht

Inhalt

Diese Sicht beschreibt, wie sich die Bausteine des Systems als Laufzeitelemente (Prozesse, Tasks, Activities, Threads, ...) verhalten und wie sie zusammenarbeiten.

Anmerkung: Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren *Architekturrelevanz.* Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

Kandidaten sind:

1. Die wichtigsten 3-5 Anwendungsfälle
2. Systemstart
3. Das Verhalten an den wichtigsten externen Schnittstellen
4. Das Verhalten in den wichtigsten Fehlerfällen

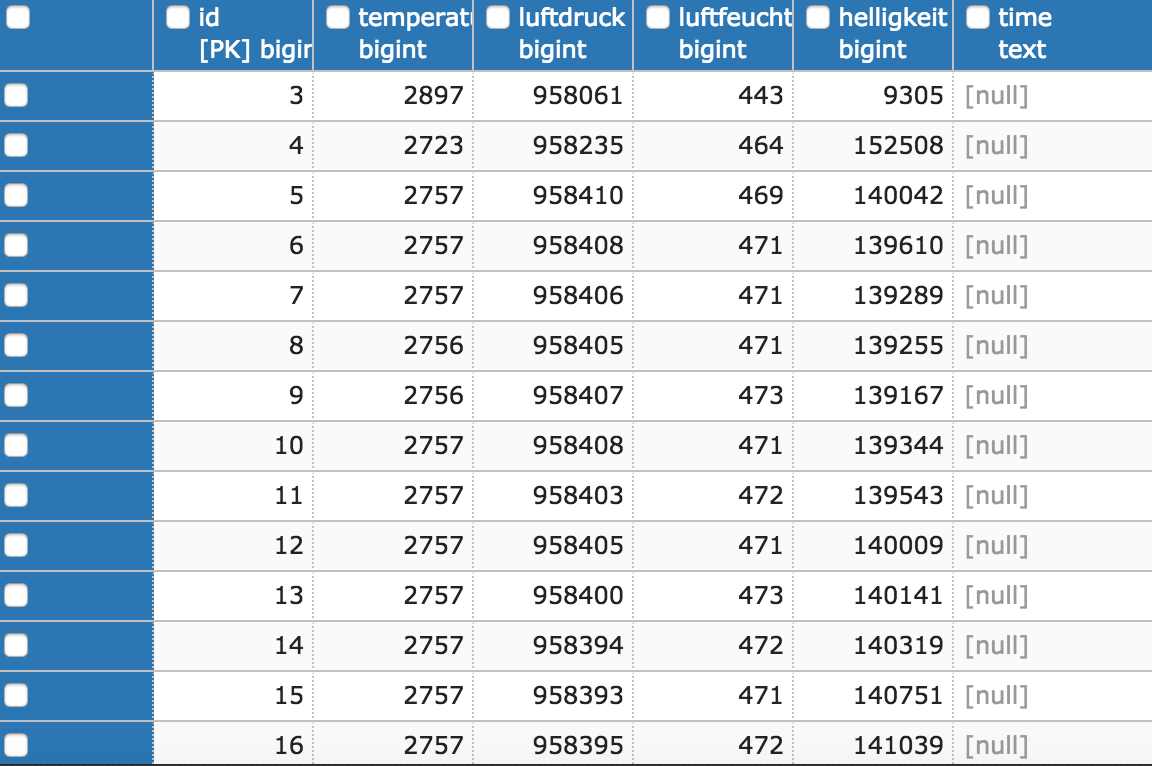
Form

Dokumentieren Sie die ausgesuchten Laufzeitszenarien mit UML-Sequenz-, Aktivitäts-, oder Kommunikationsdiagrammen.

# Datensicht

Folgende Wetterdaten müssen gespeichert werden:

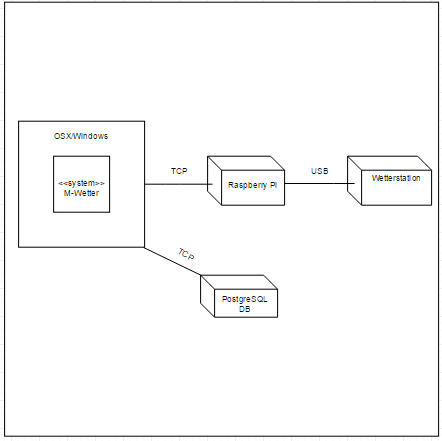
* Temperatur (Minimal-, Maximal- und Mittelwerte)
* Luftdruck (Minimal-, Maximal- und Mittelwerte)
* Luftfeuchtigkeit (Minimal-, Maximal- und Mittelwerte)
* Helligkeit (Minimal-, Maximal- und Mittelwerte)

Die Daten werden wie auf dem Screenshot abgebildet, auf einer PostgreSQL Datenbank gespeichert:

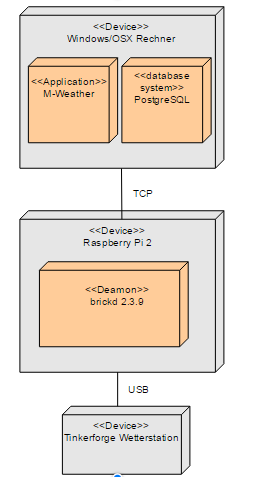
# 

# Verteilungssicht

## Oberstes Deployment-Diagramm



### Struktur der Hardwarekomponenten



# Konzepte

## Frameworks

Mit dem Java-Framework [JUnit](http://junit.org/junit4/) können sogenannte Unittests durchgeführt werden. Eine *Unit* (Einheit) kann man sich als kleinsten testbaren Teil einer Anwendung vorstellen. Hierbei kann es sich z.B. um eine ganze Klasse oder auch nur um eine einzelne Methode handeln.

Jeder Unittest sollte einen konkreten Testfall umsetzen. Ein Testfall definiert die Vorbedingungen eines Tests, die Durchführung und die nach Ausführung geltenden Nachbedingungen; sind alle Nachbedingungen erfüllt, so gilt der Test als bestanden.

## Benutzungsoberfläche

Es ist eine durch JAVAFX realisierte Benutzeroberfläche die durch wechseln der verschiedenen Tabs (Textanzeige, Graph) verändert werden kann.

# Bewertungsszenarien

* Ein Interessierter mit Grundkenntnissen in UML möchte einen Einstieg in die Architektur von M-Wetter finden. Lösungsstrategie und Entwurf erschließen sich ihm innerhalb von 15 Minuten.
* Ein Anwender will vom Tab “Luftfeuchtigkeit” zum Tab “Temperatur” wechseln. Die Anwendung wechselt das Tab mit einer Verzögerung von unter einer Sekunde.

# 

# Glossar

|  |  |
| --- | --- |
| Begriff | Definition |
| IDE | Integrated Development Envirement (Integrierte Entwicklungsumgebung) |
|  |  |