Nimrag
Use-Case-Realization Specification: Wetter Modul

Version <1.0>

Revision History

Date	Version	Description	Author
26.10.25	1.0	Erste Basisversion Wetter-Modul	Louis Reißer

Table of Contents

Use-Case-Realization Specification: Wetter Modul

1. Introduction

1.1 Purpose

Diese Use-Case-Realization Specification definiert die technische Realisierung des Use Cases "Wetteranzeige" für das Smart Mirror System Nimrag. Das Dokument beschreibt, wie die Funktionalität zur automatischen Anzeige von Wetterinformationen durch kollaborierende Objekte und Komponenten implementiert wird.

1.2 Scope

Diese Spezifikation bezieht sich auf den Use Case "Wetteranzeige" aus dem Smart Mirror Nimrag Use-Case-Modell. Sie umfasst die Interaktion zwischen Bewegungssensor, Smart Mirror Core System, Cloud Service, Weather API und Display-Komponente. Außerdem werden die dazugehörigen UML-Diagramme (Use Case, Activity, Sequence, Class Diagram) referenziert und deren Zusammenhang erläutert.

1.3 Definitions, Acronyms, and Abbreviations

- API: Application Programming Interface
- PIR: Passive Infrared (Bewegungssensor)
- REST: Representational State Transfer
- JSON: JavaScript Object Notation
- HTTP: HyperText Transfer Protocol
- Vue 3: Progressive JavaScript Framework für User Interfaces
- FastAPI: Modern, fast web framework für Python APIs
- Raspberry Pi: Single-board Computer als Hardware-Basis
- MQTT: Message Queuing Telemetry Transport Protocol
- WeatherData: Datenobjekt mit Temperatur, Luftfeuchtigkeit, Wetterbedingungen und Icons

1.4 References

1. Smart Mirror Nimrag - Software Requirements Specification Version 1.0
2. OpenWeatherMap API Documentation
3. Raspberry Pi GPIO Documentation
4. Vue 3 Official Documentation
5. FastAPI Official Documentation

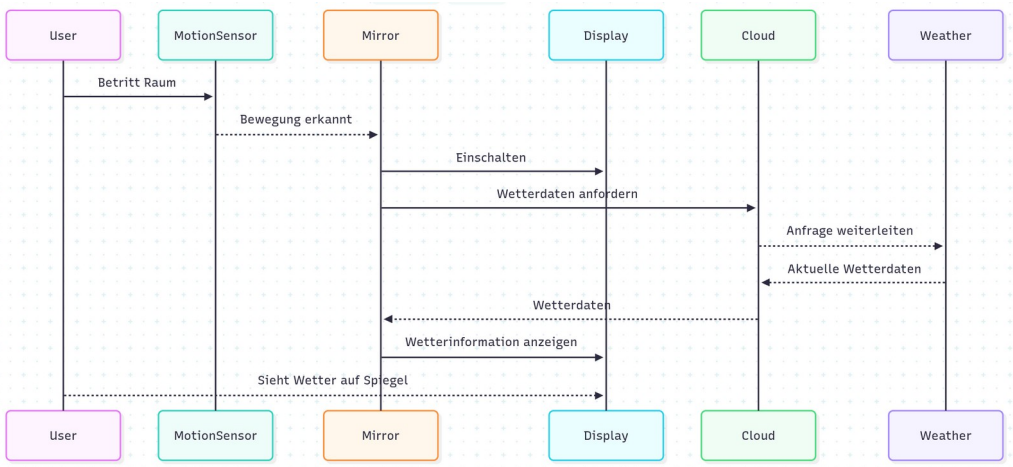
1.5 Overview

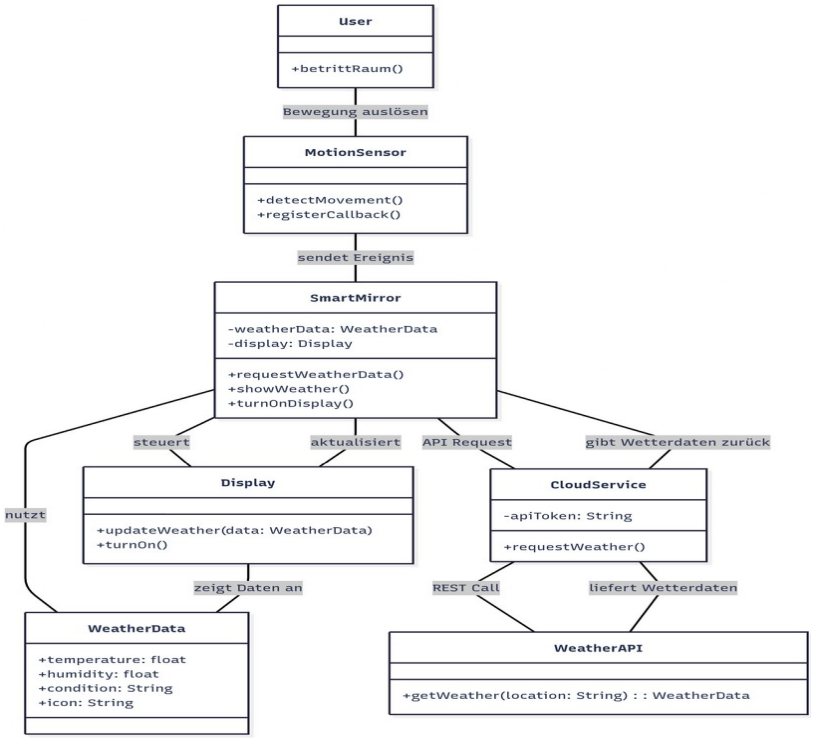
- Introduction: Zweck, Umfang und Kontext dieser Spezifikation
- Flow of Events—Design: Detaillierte Beschreibung der technischen Realisierung
- Derived Requirements: Zusätzliche nicht-funktionale Anforderungen

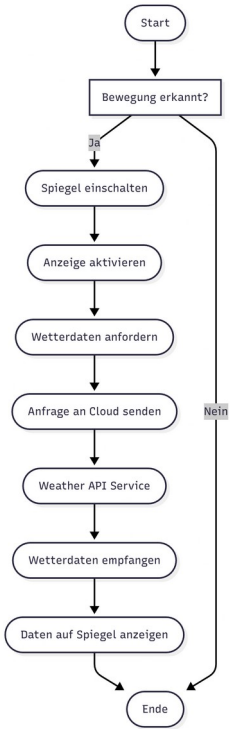
2. Flow of Events—Design

Der Use Case "Wetteranzeige" wird durch die Kollaboration von sechs Hauptkomponenten realisiert:

1. User: Löst durch Raum betreten den Prozess aus
2. MotionSensor (PIR-Sensor): Erkennt Bewegung und sendet Signal
3. SmartMirror (Core System): Zentrale Steuerungskomponente
4. Display: Physisches Anzeigesystem
5. CloudService: Vermittlungsschicht für externe APIs
6. WeatherAPI: Externe Wetterdienstschnittstelle







Detaillierter Design Flow

Phase 1: Triggering (Auslösung)
Der Use Case wird durch Bewegungserkennung initiiert. Der PIR-Sensor (MotionSensor) überwacht kontinuierlich den Raumbereich vor dem Smart Mirror. Bei Bewegungserkennung wird ein Interrupt-Signal an das SmartMirror Core System gesendet.

Phase 2: System Activation (Systemaktivierung)
Das SmartMirror Core System empfängt das Bewegungssignal und führt folgende Aktionen aus:

- Display-Komponente wird eingeschaltet (turnOnDisplay())
- Wetter-Request-Prozess wird initiiert (requestWeatherData())

Phase 3: Data Retrieval (Datenabruf)
Der CloudService fungiert als Abstraktionsschicht zwischen dem SmartMirror und externen APIs:

- SmartMirror ruft CloudService.requestWeather() auf
- CloudService sendet authentifizierten REST-Request an WeatherAPI
- WeatherAPI liefert JSON-strukturierte Wetterdaten zurück
- CloudService transformiert die Daten in interne WeatherData-Objekte

Phase 4: Data Processing and Display (Datenverarbeitung und Anzeige)
Das SmartMirror Core System verarbeitet die empfangenen WeatherData:

- Parsing und Validierung der WeatherData-Attribute (temperature, humidity, condition, icon)
- Aufruf der Display-Methode updateWeather(data: WeatherData)
- Display rendert die Wetterdaten in der Mirror-UI

Diagramm-Zusammenhänge

Use Case Diagramm: Zeigt die Akteure (User, MotionSensor) und die hierarchische Struktur der Use Case Funktionen (Wetter anzeigen → Wetter Daten anzeigen → Wetterdaten abfragen → Cloud Anfrage senden).

Activity Diagramm: Illustriert den sequenziellen Workflow mit Entscheidungslogik. Der Entscheidungsknoten "Bewegung erkannt?" ermöglicht zwei Pfade: Bei "Ja" wird der vollständige Wetteranzeigeprozess durchlaufen, bei "Nein" wird der Prozess beendet.

Sequence Diagramm: Dokumentiert die zeitliche Abfolge der Inter-Objekt-Kommunikation. Synchrone Aufrufe (→) und asynchrone Responses (-->) zwischen allen sechs Komponenten werden chronologisch dargestellt.

Class Diagramm: Definiert die statische Struktur aller beteiligten Klassen, ihre Methoden, Attribute und Beziehungen. Das WeatherData-Objekt als zentrale Datenstruktur verbindet alle Komponenten.

Technische Implementierungsdetails

Hardware-Integration:

- PIR-Sensor an GPIO-Pin des Raspberry Pi angeschlossen
- Display über HDMI-Verbindung angesteuert
- Internetverbindung für CloudService erforderlich

Software-Architektur:

- Vue 3 Frontend für Display-Rendering
- FastAPI Backend als SmartMirror Core System
- Modulare Komponenten-Architektur für Erweiterbarkeit
- Async/Await Pattern für non-blocking API-Calls

Datenfluss:

- Input: Bewegungssignal (boolean)
- Processing: WeatherData (JSON → Object transformation)
- Output: Formatierte Wetteranzeige auf Mirror-UI

3. Derived Requirements

Performance Requirements

PER-01: Das System muss innerhalb von 2 Sekunden nach Bewegungserkennung die Wetteranzeige aktivieren.

PER-02: Der Wetterdatenabruf darf nicht länger als 5 Sekunden dauern bei stabiler Internetverbindung.

PER-03: Das Display muss mit mindestens 30 FPS aktualisiert werden, um eine flüssige Darstellung zu gewährleisten.

Reliability Requirements

REL-01: Bei Ausfall der Internetverbindung muss das System zuletzt abgerufene Wetterdaten für mindestens 30 Minuten zwischenspeichern.

REL-02: Bei API-Timeouts (>10 Sekunden) muss eine Fallback-Anzeige ("Wetterdaten nicht verfügbar") erscheinen.

REL-03: Der MotionSensor muss eine Verfügbarkeit von 99,5% gewährleisten.

Security Requirements

SEC-01: API-Keys für den WeatherService müssen verschlüsselt in Konfigurationsdateien gespeichert werden.

SEC-02: Alle HTTP-Requests an externe APIs müssen über HTTPS (TLS 1.3) erfolgen.

SEC-03: Keine personenbezogenen Daten dürfen an externe WeatherAPIs übermittelt werden.

Usability Requirements

USA-01: Die Wetterdaten müssen in gut lesbarer Schriftgröße (min. 24px) dargestellt werden.

USA-02: Wettersymbole müssen auch bei schlechten Lichtverhältnissen erkennbar sein (Kontrast ≥ 4.5:1).

USA-03: Die Wetteranzeige muss nach 60 Sekunden Inaktivität automatisch in den Standby-Modus wechseln.

Maintainability Requirements

MAI-01: Die WeatherAPI-Integration muss modular implementiert werden, um Anbieterwechsel zu ermöglichen.

MAI-02: Alle Komponenten müssen Unit-Tests mit mindestens 80% Code Coverage aufweisen.

MAI-03: Konfigurationsparameter (Update-Intervalle, API-Endpoints) müssen über externe Konfigurationsdateien anpassbar sein.

Platform Requirements

PLA-01: Das System muss auf Raspberry Pi 4 (4GB RAM) lauffähig sein.

PLA-02: Unterstützung für Full HD Displays (1920x1080) ist erforderlich.

PLA-03: Das System muss unter Raspberry Pi OS (Debian-basiert) betriebsbereit sein.