Nimrag – Software Requirements Specification

Version: 1.0
Datum: 20.10.2025
Autoren: Sebastian, Jannik, Jan, Louis
Status: In Entwicklung

Revision History

Datum	Version	Beschreibung	Autor(en)
17.10.2025	0.9	Erste Dokumentstruktur	Sebastian, Jannik, Jan, Louis
20.10.2025	1.0	Vollständige SRS mit Blogeintrag	Sebastian, Jannik, Jan, Louis

Inhaltsverzeichnis

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms and Abbreviations
    - 1.4 References
    - 1.5 Overview
2. Overall Description
    - 2.1 Product Perspective
    - 2.2 Product Functions
    - 2.3 Technology Stack Decision
    - 2.4 Development Approach
3. Specific Requirements
    - 3.1 Functionality
    - 3.2 Usability
    - 3.3 Reliability
    - 3.4 Performance
    - 3.5 Supportability
    - 3.6 Design Constraints
    - 3.7 Online User Documentation and Help System Requirements
    - 3.8 Purchased Components
    - 3.9 Interfaces
    - 3.10 Licensing Requirements
    - 3.11 Legal, Copyright and Other Notices
    - 3.12 Applicable Standards

4. Supporting Information

---

1. Introduction

1.1 Purpose

Das Ziel des Nimrag-Projekts ist es, einen „schlauen" Spiegel zu entwickeln, der den Alltag erleichtert. Funktionen wie Wetterbericht, Verkehrshinweise, Kalenderintegration und weitere Informationen sollen über eine intuitive, moderne Oberfläche bereitgestellt werden.

Dieses Software Requirements Specification (SRS) Dokument richtet sich an:

- **Entwicklungsteam** (Sebastian, Jannik, Jan, Louis) zur technischen Umsetzung
- **Stakeholder** zur Validierung der Anforderungen
- **Zukünftige Entwickler** für Erweiterungen und Wartung
- **Tester** für die Qualitätssicherung

1.2 Scope

Der Smart Mirror "Nimrag" dient als Smart-Home-Bridge und zentrale Informationsdisplay. Das System kann durch modulare Widgets erweitert werden und bietet eine flexible Architektur für zukünftige Funktionen.

Hauptmerkmale:

- Modulares, erweiterbares Design
- Smart-Home-Integration via MQTT
- Moderne Web-Technologien (Vue 3, TypeScript)
- Raspberry Pi-basierte Hardware
- Sprach- und Gestensteuerung
- Offline-Funktionalität für Grundfunktionen

Projektabgrenzung:

- Fokus auf Software-Entwicklung und -Architektur
- Hardware-Spezifikationen nur soweit für Software relevant
- Eigene Lösung statt MagicMirror² für maximale Flexibilität und Lerneffekt

1.3 Definitions, Acronyms and Abbreviations

Begriff	Definition
API	Application Programming Interface - Programmierschnittstelle
GPIO	General Purpose Input/Output - Programmierbare Ein-/Ausgabe-Pins
HUD	Head-Up Display - Anzeigesystem im Sichtfeld
IoT	Internet of Things - Netzwerk physischer Geräte
MQTT	Message Queuing Telemetry Transport - Messaging-Protokoll für IoT
MVP	Minimum Viable Product - Funktionsfähiger Grundprototyp

Begriff	Definition
PWM	Pulse Width Modulation - Pulsweitenmodulation
REST	Representational State Transfer - Architekturstil für Web-Services
SRS	Software Requirements Specification
UI	User Interface - Benutzeroberfläche
Vue 3	JavaScript-Framework für reaktive Benutzeroberflächen
WebSocket	Protokoll für bidirektionale Kommunikation

1.4 References

Referenz-ID	Titel	Version/Datum	Quelle
[REF-1]	Nimrag Projektbeschreibung	2025.10	GitHub Repository
[REF-2]	Vue 3 Documentation	v3.x	https://vuejs.org/
[REF-3]	FastAPI Documentation	v0.104	https://fastapi.tiangolo.com/
[REF-4]	MediaPipe Documentation	v0.10	https://mediapipe.dev/
[REF-5]	Raspberry Pi OS Documentation	2025.10	https://www.raspberrypi.org/documentation/
[REF-6]	MQTT Protocol Specification	v3.1.1	https://mqtt.org/

1.5 Overview

Dieses Dokument beschreibt die vollständigen Anforderungen für das Nimrag Smart Mirror System. Das Team hat sich bewusst für eine eigene Lösung entschieden, um maximale Flexibilität zu gewährleisten und den Lerneffekt zu maximieren.

Kapitel 2 beschreibt die Produktperspektive, Funktionen und die Technologie-Entscheidungen des Teams.

Kapitel 3 definiert detaillierte funktionale und nicht-funktionale Anforderungen.

Kapitel 4 enthält zusätzliche Informationen, Glossar und Anhänge.

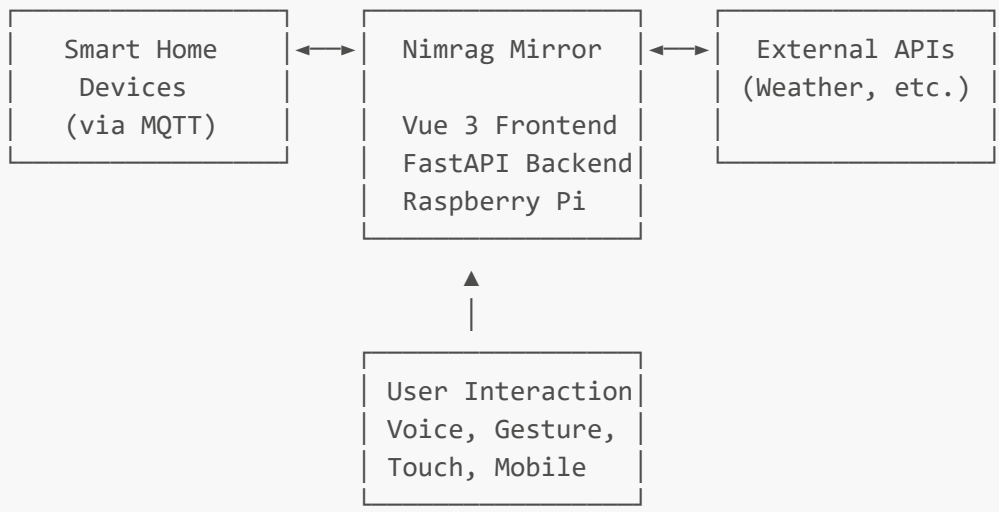
---

2. Overall Description

2.1 Product Perspective

Das Nimrag-System ist ein eigenständiges Smart-Home-Gerät, das als zentrale Informationsdisplay und Steuereinheit fungiert. Es basiert auf einer modernen Web-Architektur und läuft auf Raspberry Pi Hardware.

Systemarchitektur:



2.2 Product Functions

Kernfunktionen (MVP):

- **Zeit und Datum**: Aktuelle Uhrzeit und Datum mit anpassbaren Formaten
- **Wetterinformationen**: Aktuelle Bedingungen und Vorhersage
- **Kalenderintegration**: Anstehende Termine und Ereignisse
- **LED-Steuerung**: Hintergrundbeleuchtung mit RGB-LEDs

Erweiterte Funktionen (geplant):

- **Musikintegration**: Anzeige aktueller Wiedergabe, grundlegende Steuerung
- **Sprachsteuerung**: Vosk ASR für offline Spracherkennung
- **Smart-Home-Integration**: MQTT-basierte Gerätesteuerung und -status
- **Gestensteuerung**: MediaPipe und OpenCV für berührungslose Bedienung
- **Modulares Layout**: Drag-and-Drop Widget-Anordnung
- **Mobile App**: Fernsteuerung über Smartphone

2.3 Technology Stack Decision

Damit alle im Team dieselbe Vorstellung vom Projekt haben, wurde eine umfassende Technologie-Evaluierung durchgeführt. Das Team hat sich bewusst gegen MagicMirror² entschieden, um flexibler zu sein und mehr zu lernen.

Frontend-Technologien:

- **Vue 3**: Modernes, reaktives JavaScript-Framework
- **TypeScript**: Typsicherheit und bessere Entwicklererfahrung
- **Tailwind CSS**: Utility-First CSS-Framework für schnelles, anpassbares Design
- **Vite**: Schneller Build-Tool für moderne Web-Entwicklung

Backend-Technologien:

- **FastAPI (Python)**: Leichtgewichtig, ideal für REST-Schnittstellen
- **WebSocket-Support**: Für Echtzeit-Kommunikation
- **Async/Await**: Für performante I/O-Operationen

Hardware-Integration:

- **MediaPipe + OpenCV**: Experimentelle Gestensteuerung
- **Vosk ASR**: Offline-Spracherkennung
- **GPIO-Control**: Direkte Hardware-Ansteuerung
- **MQTT-Client**: Smart-Home-Kommunikation

Begründung der Technologie-Wahl:

- **Modern & Schnell**: Vue 3 + TypeScript + Tailwind bieten moderne Entwicklung
- **Anpassbar**: Vollständige Kontrolle über alle Aspekte des Systems
- **Lerneffekt**: Tieferes Verständnis durch eigene Implementierung
- **Erweiterbar**: Flexible Architektur für zukünftige Features

2.4 Development Approach

Entwicklungsstrategie:

1. **Setup Phase**: Frontend und Backend Grundgerüst aufsetzen
2. **Emulation**: Raspberry Pi emulieren für hardware-nahes Testen
3. **MVP Development**: Basisfunktionen implementieren (Zeit, Wetter, Kalender, LEDs)
4. **Feature Expansion**: Erweiterte Features wie Musik, Sprache, Smart-Home
5. **Hardware Integration**: Testen mit echter Raspberry Pi Hardware

Projektphasen:

```
Phase 1: Grundgerüst (2-3 Wochen)
├── Vue 3 + TypeScript Frontend Setup
├── FastAPI Backend Setup
├── Raspberry Pi Emulation
└── Basis CI/CD Pipeline

Phase 2: MVP (3-4 Wochen)
├── Zeit/Datum Widget
├── Wetter Widget
├── Kalender Widget
├── LED-Steuerung
└── Basis Testing

Phase 3: Erweiterte Features (4-6 Wochen)
├── Musikintegration
├── Spracherkennung (Vosk)
├── Gestensteuerung (MediaPipe)
├── Smart-Home MQTT
└── Mobile App

Phase 4: Polish & Deployment (2-3 Wochen)
```

```
├── UI/UX Verbesserungen
├── Performance Optimierung
├── Dokumentation
└── Hardware-Integration
```

---

3. Specific Requirements

3.1 Functionality

3.1.1 Kernfunktionen (MVP)

Zeit und Datum Widget

- Aktuelle Uhrzeit in konfigurierbaren Formaten (12h/24h)
- Datum mit Wochentag in deutscher/englischer Sprache
- Automatische Zeitzonenerkennung und Sommerzeit
- Anpassbare Schriftgrößen und -farben

Wetter Widget

- Aktuelle Wetterbedingungen mit Icons
- 5-7 Tage Vorhersage
- Temperatur, Luftfeuchtigkeit, Windgeschwindigkeit
- Wetterwarnung-Integration
- Konfigurierbare Standorte

Kalender Widget

- Google Calendar Integration via API
- Anstehende Termine für die nächsten 7 Tage
- Ganztägige Ereignisse und Geburtstage
- Farbcodierung nach Kalender-Kategorien
- Terminbenachrichtigungen

LED-Steuerung

- RGB-LED-Streifen Ansteuerung via GPIO
- Farbwechsel basierend auf Uhrzeit/Wetter
- Helligkeitssteuerung mit PWM
- Vordefinierte Szenen (Morgen, Abend, Nacht)

3.1.2 Erweiterte Funktionen

Modulares Layout

- Drag-and-Drop Widget-Anordnung
- Responsive Design für verschiedene Bildschirmgrößen
- Widget-spezifische Konfigurationen
- Layout-Profile für verschiedene Benutzer

Multimedia-Integration

- Spotify-Integration für aktuelle Wiedergabe
- Grundlegende Mediensteuerung (Play/Pause/Skip)
- Podcast/Audiobook-Status
- Lautstärkeregelung

Smart-Home-Integration

- MQTT-Broker Kommunikation
- Geräte-Status Anzeige (Lichter, Sensoren, Schalter)
- Grundlegende Gerätesteuerung
- Energieverbrauch-Monitoring

3.2 Usability

3.2.1 Zielgruppe

Primäre Benutzer:

- Technik-interessierte Privatpersonen (25-50 Jahre)
- Smart-Home-Enthusiasten
- Familienhaushalte mit technischer Affinität

Barrierefreiheit:

- Sprach- und Gestensteuerung für berührungslose Bedienung
- Hohe Kontrastverhältnisse für Sichtbarkeit durch Spiegel
- Skalierbare Schriftgrößen
- Einfache, intuitive Navigation

3.2.2 Benutzeroberfläche

Design-Prinzipien:

- **Minimalistisch**: Schlichtes, übersichtliches Design
- **Dark Theme**: Optimiert für Zwei-Wege-Spiegel
- **Modulare Widgets**: Flexibel anordbare Informationsblöcke
- **Responsive**: Funktional auf verschiedenen Bildschirmgrößen

Interaktionsmethoden:

- **Touch**: Direkte Berührung für Konfiguration
- **Sprache**: "Spiegel, zeige Wetter" (Vosk ASR)
- **Gesten**: Hand-Tracking mit MediaPipe
- **Mobile App**: Remote Control über Smartphone

3.2.3 Bedienbarkeit

Navigation:

- Konsistente UI-Elemente und Icons
- Kontextuelle Tooltips und Hilfetexte
- Breadcrumb-Navigation bei tieferen Menüs
- Schneller Zugriff auf häufig genutzte Funktionen

Konfiguration:

- Web-basiertes Setup-Interface
- Wizard für Ersteinrichtung
- Live-Vorschau bei Änderungen
- Import/Export von Konfigurationen

3.3 Reliability

3.3.1 Availability

Systemverfügbarkeit:

- 24/7 Betrieb mit automatischem Neustart bei Fehlern
- Graceful Degradation bei API-Ausfällen
- Offline-Modus für Kernfunktionen
- Automatische Updates ohne Downtime

3.3.2 Accuracy

Datengenauigkeit:

- Präzise Zeitanzeige mit NTP-Synchronisation
- Aktuelle Wetterdaten mit max. 10 Minuten Verzögerung
- Kalender-Synchronisation alle 15 Minuten
- Sensor-Daten mit ±2% Genauigkeit

3.3.3 Fehlerkategorien

Kritische Fehler:

- Systemabsturz oder Boot-Failure
- Totaler Datenverlust
- Hardware-Schäden durch Software

Signifikante Fehler:

- Widget-Funktionalität teilweise eingeschränkt
- API-Verbindungsabbrüche
- Performance-Degradation

Kleine Fehler:

- UI-Darstellungsfehler
- Verzögerte Updates
- Kosmetische Probleme

3.4 Performance

3.4.1 Antwortzeiten

System-Performance:

- Boot-Zeit: < 60 Sekunden bis vollständig geladen
- UI-Responsivität: < 200ms für Benutzerinteraktionen
- Widget-Updates: < 5 Sekunden für externe API-Calls
- Sprach-/Gestenerkennung: < 500ms Latenz

3.4.2 Degradationsverhalten

Bei langsamer Internetverbindung:

- Anzeige von "Daten werden geladen" Hinweisen
- Nutzung von gecachten Daten
- Reduzierte Update-Frequenz
- Priorität auf Kernfunktionen

Bei Hardware-Limitierungen:

- Dynamische Qualitätsanpassung für Video-Processing
- Reduzierte Animationen bei niedrigem RAM
- Intelligente Ressourcen-Priorisierung

3.5 Supportability

3.5.1 Wartbarkeit

Logging und Monitoring:

- Strukturierte JSON-Logs mit verschiedenen Log-Levels
- Performance-Metriken für alle Komponenten
- Fehler-Tracking mit Stack-Traces
- Remote-Monitoring via Web-Dashboard

Update-Mechanismus:

- Git-basierte Updates über GitHub
- Rollback-Funktionalität bei fehlgeschlagenen Updates
- Automatische Dependency-Updates
- Konfiguration-Backups vor Updates

3.6 Design Constraints

3.6.1 Hardware-Plattform

Zielplattform & Betriebssystem:

- Raspberry Pi 4 oder 5 mit Raspberry Pi OS (Linux, Debian-Basis)

- System läuft im Kiosk-Modus über Chromium-Browser
- Lokaler Webserver für Frontend-Hosting

Hardware-Anforderungen:

- Raspberry Pi mit HDMI-Display
- USB-Kamera für Gestenerkennung
- Mikrofon für Spracherkennung
- GPIO-angeschlossene LED-Streifen
- Dauerhafte Netzverbindung (WLAN oder LAN)

3.6.3 Integration-Constraints

Spracherkennung:

- Vosk ASR für lokale, offline-fähige Spracherkennung
- Deutsche und englische Sprachmodelle
- Noise-Cancellation für bessere Erkennung

Gestenerkennung:

- MediaPipe + OpenCV für Echtzeit-Bildverarbeitung
- Hand-Tracking und Gesture-Recognition
- CPU-Optimierung für Raspberry Pi

Smart-Home-Kommunikation:

- MQTT (Mosquitto-Broker) für IoT-Geräte-Integration
- JSON-basierte Message-Formate
- TLS-Verschlüsselung für sichere Kommunikation

3.6.2 Performance-Constraints

Leistungsgrenzen:

- Echtzeit-Verarbeitung (Video + Audio) erfordert CPU-Optimierung
- GPU-Beschleunigung wo verfügbar nutzen
- Memory-Management für 4GB RAM Limit
- Effiziente Ressourcen-Nutzung für 24/7 Betrieb

3.7 Online User Documentation and Help System Requirements

Online-Dokumentation:

- GitHub-Repository mit umfassender Markdown-Dokumentation
- API-Dokumentation automatisch generiert mit FastAPI
- Code-Kommentare und Inline-Dokumentation

Setup-Guide:

- Schritt-für-Schritt Installationsanleitung

- Raspberry Pi OS Setup und Konfiguration
- Backend-Dependencies Installation (Python, FastAPI, Vosk, OpenCV)
- Frontend-Build-Prozess mit Vite
- MQTT-Broker Konfiguration

Benutzerhandbuch:

- Web-basierte Hilfeseite im UI ("?"-Icon)
- Erklärung aller Widgets und Funktionen
- Konfiguration-Tutorials mit Screenshots
- Troubleshooting-Section

Erweiterbarkeit:

- Plugin-Development-Guide
- API-Contracts für neue Widgets
- Code-Style-Guide für Beiträge
- Testing-Guidelines

3.8 Purchased Components

3.8.1 Hardware

Basis-Hardware:

- Fernseher + Wandhalterung (bereits vorhanden)
- Raspberry Pi 4B (4GB RAM) oder Raspberry Pi 5
- Holzrahmen für Fernseher-Verkleidung
- Zwei-Wege-Spiegel (Maße passend zum Fernseher)

Raspberry Pi Zubehör:

- Micro HDMI zu HDMI Kabel (2m)
- USB-C Stromkabel für Pi (5V/3A)
- Schutzhülle/Gehäuse für Pi
- MicroSD-Karte (64GB, Class 10)

Eingabegeräte:

- USB-Kamera für Gestenerkennung
- USB-Mikrofon für Spracherkennung
- Optional: Touch-Overlay für Display

3.8.2 LED-Beleuchtung & Elektronik

LED-System:

- RGB LED-Strip (WS2812B, 5m, schneidbar)
- Smart-Home-kompatible LEDs
- Individuelle Anpassung möglich

Elektronische Steuerung:

- N-Channel MOSFET für LED-Steuerung
- PWM-Kontrolle über Raspberry Pi GPIO
- Sonoff Smart Switch für Haupt-Ein/Ausschaltung
- Smart-Home-Integration und Fernsteuerung

Verkabelung & Prototyping:

- Steckplatine + Jumper-Kabel Set
- Breadboard für Testschaltungen
- Lötausrüstung für finale Verkabelung
- Kabelmanagement und Befestigungsmaterial

3.8.3 Software-Lizenzen

Kostenlose Services:

- GitHub (öffentliches Repository)
- OpenWeatherMap API (Free Tier: 1000 calls/day)
- Google Calendar API (Free Tier)
- Spotify Web API (Free für Metadaten)

Optional kostenpflichtig:

- Premium Weather API für erweiterte Daten
- Cloud-MQTT-Broker Service
- Domain und SSL-Zertifikat für Remote-Access

3.9 Interfaces

3.9.1 User Interfaces

Spiegel-Frontend (Haupt-UI):

- Web-basierte Benutzeroberfläche im Vollbild-Modus
- Entwickelt mit Vue 3, TypeScript und Tailwind CSS
- Dark Theme optimiert für Zwei-Wege-Spiegel
- Modulare Widget-Anordnung
- Touch-optimierte Bedienelemente

Widget-Kategorien:

- Zeit/Datum mit anpassbaren Formaten
- Wetter mit Icons und Vorhersage
- Kalender mit Terminen und Erinnerungen
- Smart-Home-Status und -Steuerung
- Nachrichten und RSS-Feeds
- Musik-Player-Informationen
- Persönliche Notizen und To-Do-Listen

Interaktionsmethoden:

- **Sprachbefehle**: "Spiegel, zeige Wetter" (Vosk ASR)
- **Handgesten**: MediaPipe-basierte Erkennung (Swipe, Point, etc.)
- **Touch-Eingabe**: Direkte Berührung für Konfiguration
- **Mobile Remote**: Smartphone-App für Fernsteuerung

Konfiguration-Interface:

- Web-basierte Konfigurationsseite
- Live-Vorschau von Layout-Änderungen
- Widget-spezifische Einstellungen
- Benutzer-Profile und Szenen

3.9.2 Hardware Interfaces

GPIO-Schnittstellen:

- PWM-Pins für LED-Steuerung (Pin 18, 19)
- Digital I/O für Statusanzeigen und Sensoren
- I2C für Temperatur- und Feuchtigkeitssensoren
- SPI für erweiterte Sensormodule

Video/Audio-Interfaces:

- USB-Kamera (USB 2.0/3.0) für Gestenerkennung
- USB-Mikrofon für Spracherkennung
- HDMI-Ausgang für Display (1920x1080@60Hz)
- 3.5mm Audio-Ausgang für optionales Audio-Feedback

Netzwerk-Interfaces:

- WiFi 802.11ac für Internet-Konnektivität
- Ethernet als Backup-Verbindung
- Bluetooth für Smartphone-Kopplung

Stromversorgung:

- USB-C Stromeingang (5V/3A)
- GPIO-basierte Spannungsüberwachung
- Soft-Shutdown über Hardware-Button

3.9.3 Software Interfaces

REST API (FastAPI):

- RESTful Endpunkte für alle Widget-Daten
- OpenAPI/Swagger Dokumentation
- Async/Await für performante I/O
- JWT-basierte Authentifizierung

API-Endpunkte:

```
GET /api/v1/weather        - Wetterdaten
GET /api/v1/calendar       - Kalenderereignisse
GET /api/v1/smart-home     - Smart-Home-Status
POST /api/v1/led/control   - LED-Steuerung
GET /api/v1/system/status  - System-Informationen
```

WebSocket-Schnittstellen:

- Echtzeit-Updates für Widgets
- Sprach-/Gesten-Erkennungs-Events
- Smart-Home-Zustandsänderungen
- System-Notifications

MQTT-Integration:

- Mosquitto-Broker Kommunikation
- Topic-basierte Nachrichten-Routing
- QoS-Levels für verschiedene Nachrichtentypen
- JSON-Payload für strukturierte Daten

MQTT-Topics:

```
nimrag/status/online       - System-Status
nimrag/led/brightness      - LED-Helligkeits-Control
nimrag/voice/command       - Sprachbefehle
nimrag/gesture/detected    - Erkannte Gesten
smart-home/+/status        - Smart-Home-Device-Status
```

3.9.4 Communications Interfaces

Netzwerkprotokolle:

- **HTTP/HTTPS**: Frontend ↔ Backend Kommunikation
- **WebSocket**: Echtzeit-Datenübertragung
- **MQTT over TCP/TLS**: Smart-Home-Integration
- **mDNS**: Service-Discovery im lokalen Netzwerk

API-Kommunikation:

- RESTful JSON APIs für externe Services
- OAuth 2.0 für Google Calendar Integration
- API-Keys für Weather Services
- Rate-Limiting und Retry-Mechanismen

Authentifizierung & Sicherheit:

- JWT-basierte Session-Tokens
- API-Key-Management für externe Services
- TLS-Verschlüsselung für alle externen Verbindungen
- Local-only Access für Security-kritische Funktionen

Offline-Betrieb:

- Lokaler Cache für Wetter- und Kalender-Daten
- SQLite-Datenbank für persistente Speicherung
- Graceful Degradation bei Netzwerkausfällen
- Automatische Wiederverbindung und Sync

Fehlerbehandlung:

- Exponential Backoff für API-Retry-Logic
- Circuit Breaker Pattern für externe Services
- Comprehensive Logging für Debugging
- User-friendly Error Messages im UI

3.10 Licensing Requirements

Open Source Komponenten:

- Vue 3: MIT License
- FastAPI: MIT License
- TypeScript: Apache License 2.0
- Tailwind CSS: MIT License
- MediaPipe: Apache License 2.0
- OpenCV: Apache License 2.0
- Vosk: Apache License 2.0

Kommerzielle API-Services:

- OpenWeatherMap: Free Tier für Entwicklung
- Google Calendar API: Free für persönliche Nutzung
- Spotify Web API: Free für Metadaten-Zugriff

Projektlizenz:

- MIT License für den Nimrag-Code
- Offene Entwicklung auf GitHub
- Community-Beiträge willkommen

3.11 Legal, Copyright and Other Notices

Datenschutz (DSGVO-Konform):

- Lokale Datenverarbeitung wo immer möglich
- Transparente Informationen über externe API-Nutzung
- Opt-in für alle Cloud-Services
- Benutzer-Kontrolle über Datensammlung

Sicherheitshinweise:

- Regular Security Updates für alle Dependencies
- Sichere Speicherung von API-Credentials
- Network Security Best Practices
- Hardware-spezifische Sicherheitsmaßnahmen

3.12 Applicable Standards

Web-Standards:

- HTML5, CSS3, ES2020+ JavaScript
- W3C Accessibility Guidelines (WCAG 2.1)
- Progressive Web App (PWA) Standards
- RESTful API Design Principles

Hardware-Standards:

- Raspberry Pi GPIO Interface Standards
- USB HID Standards für Input-Devices
- HDMI Display Standards
- Bluetooth Low Energy (BLE) für mobile Verbindungen

Kommunikations-Standards:

- MQTT v3.1.1/v5.0 für IoT-Kommunikation
- WebSocket (RFC 6455) für Realtime-Kommunikation
- OAuth 2.0 für sichere API-Authentifizierung
- TLS 1.2+ für verschlüsselte Verbindungen

---

4. Supporting Information

4.1 Projekthistorie

Das Nimrag-Projekt entstand aus dem Bedürfnis nach einem flexiblen, modernen Smart Mirror System. Das Entwicklungsteam (Sebastian, Jannik, Jan, Louis) entschied sich bewusst gegen bestehende Lösungen wie MagicMirror^2, um maximale Kontrolle über die Technologie und den Lerneffekt zu haben.

4.2 Entwicklungs-Roadmap

Phase 1 (Aktuell): Grundgerüst

- Frontend/Backend Setup
- Raspberry Pi Emulation
- Basis-CI/CD

Phase 2: MVP Development

- Kern-Widgets implementieren
- Hardware-Integration testen

- Basic Testing Suite

Phase 3: Advanced Features

- Sprach- und Gestensteuerung
- Smart-Home MQTT Integration
- Mobile App Development

Phase 4: Polish & Deployment

- Performance Optimierung
- Dokumentation vervollständigen
- Community Release

4.3 Glossar

Begriff	Definition	Kontext
Vosk ASR	Open-Source Automatic Speech Recognition Library	Offline Spracherkennung
MediaPipe	Google's Framework für Multimedia-Processing	Gestenerkennung
FastAPI	Modernes Python Web-Framework für APIs	Backend-Entwicklung
Vite	Next-generation Frontend-Build-Tool	Entwicklungs-Toolchain
PWA	Progressive Web Application	Web-App mit App-ähnlicher Funktionalität
GPIO	General Purpose Input/Output	Raspberry Pi Hardware-Schnittstelle
MQTT	Message Queuing Telemetry Transport	IoT-Kommunikationsprotokoll
JWT	JSON Web Token	Sichere Token-basierte Authentifizierung

Ende des Dokuments

Dieses Dokument umfasst die vollständigen Anforderungen für das Nimrag Smart Mirror System v1.0