Architecture Decision Records (ADR)

Projekt: FeWo-Verwaltungssystem

Systemarchitektur Dokumentation

1. Februar 2026

Einführung

Dieses Dokument beschreibt die fundamentalen Architekturentscheidungen für das Kundenverwaltungsprogramm. Es dokumentiert den gewählten Technologie-Stack (Hardware, Netzwerk, Backend, Frontend) und begründet die Auswahl der Komponenten im Hinblick auf Performance, Wartbarkeit und Sicherheit.

---

1 ADR-001: Hardware-Infrastruktur

Status: Akzeptiert

Kontext

Das System benötigt eine Hosting-Umgebung, die 24/7 verfügbar ist, geringe Betriebskosten verursacht und die volle Kontrolle über die Daten gewährleistet. Cloud-VPS-Lösungen verursachen laufende Kosten, während lokale Server oft energieintensiv sind.

Entscheidung

Einsatz eines **Raspberry Pi 5** als On-Premise-Server.

Begründung

Der Raspberry Pi 5 bietet im Vergleich zu Vorgängermodellen eine signifikante Leistungssteigerung (PCIe, CPU-Takt), die für Webanwendungen mit geringer bis mittlerer Last mehr als ausreichend ist ("Overkill"). Dies garantiert schnelle Antwortzeiten bei minimalem Stromverbrauch (ca. 3-5 Watt im Idle). Die Daten bleiben physisch im Besitz des Betreibers.

---

2 ADR-002: Ingress & Netzwerksicherheit

Status: Akzeptiert

Kontext

Der Server befindet sich hinter einer privaten Firewall (Heimnetzwerk) mit dynamischer IP-Adresse. Das Öffnen von Ports am Router (Port Forwarding) stellt ein Sicherheitsrisiko dar. Zudem ist die Verwaltung von SSL-Zertifikaten und dynamischem DNS oft fehleranfällig.

Entscheidung

Nutzung von **Cloudflare Tunnel (cloudflared)** als einziger Einstiegspunkt.

Begründung

Der Cloudflare Tunnel baut eine persistente, ausgehende Verbindung zur Cloudflare-Edge auf. Dadurch sind keine eingehenden Ports am Router notwendig. Cloudflare übernimmt die Terminierung von HTTPS (SSL-Offloading), den DDoS-Schutz und das Routing. Dies entkoppelt die lokale Netzwerkkonfiguration von der öffentlichen Erreichbarkeit.

---

3 ADR-003: Reverse Proxy & Statische Assets

Status: Akzeptiert

Kontext

Python-Anwendungsserver sind nicht für das Ausliefern statischer Dateien (CSS, JavaScript, Bilder) optimiert und können bei langsamen Client-Verbindungen blockieren.

Entscheidung

Einsatz von **Nginx** als Webserver und Reverse Proxy.

Begründung

Nginx fungiert als effizienter Puffer zwischen dem Cloudflare Tunnel und der Applikation.

Asset Management: Nginx liefert statische Dateien direkt vom Dateisystem aus, was die Last auf der CPU minimiert.

Proxy-Funktion: Anfragen an die Applikationslogik werden intern an Port 3000 weitergeleitet. Nginx hält Verbindungen zu langsamen Clients offen, sodass der Applikationsserver (Gunicorn) sofort wieder frei für neue Anfragen ist.

---

4 ADR-004: Applikationsserver (WSGI)

Status: Akzeptiert

Kontext

Das gewählte Backend-Framework (Django) benötigt eine Schnittstelle, um mit dem Webserver zu kommunizieren (WSGI-Standard). Der integrierte Entwicklungsserver von Django ist nicht sicher für den Produktionsbetrieb.

Entscheidung

Einsatz von **Gunicorn** (Green Unicorn).

Begründung

Gunicorn ist der Industriestandard für Python-Webanwendungen im Produktionsumfeld. Es ist ressourcenschonend und stabil. Die Konfiguration mit mehreren Workern ermöglicht die parallele Verarbeitung von Anfragen, selbst auf der begrenzten Hardware des Raspberry Pi. Gunicorn wird als Systemd-Dienst verwaltet, um automatische Neustarts zu gewährleisten.

---

5 ADR-005: Backend-Framework

Status: Akzeptiert

Kontext

Es wird ein robustes Framework benötigt, das relationale Datenstrukturen (Kunden, Objekte, Buchungen) effizient abbilden kann und Sicherheitsfeatures (Authentifizierung, CSRF-Schutz) von Haus aus mitbringt.

Entscheidung

Verwendung von **Django** (Python).

Begründung

Django folgt dem "Batteries-Included"-Ansatz. Das integrierte ORM (Object-Relational Mapping) abstrahiert die Datenbankzugriffe. Das automatische Admin-Interface reduziert den Entwicklungsaufwand für die Stammdatenverwaltung erheblich. Python als Sprache ermöglicht einfache Wartbarkeit und Zugang zu zahlreichen Bibliotheken.

---

6 ADR-006: Datenhaltung

Status: Akzeptiert

Kontext

Die Anwendung muss strukturierte Daten speichern. Die erwartete Schreiblast ist gering, die Komplexität der Infrastruktur soll minimiert werden. Ein vollwertiger Datenbankserver (PostgreSQL/MySQL) verbraucht zusätzlichen Arbeitsspeicher.

Entscheidung

Einsatz von **SQLite**.

Begründung

SQLite ist eine serverlose, dateibasierte Datenbank. Sie benötigt keinen separaten Prozess und keinen Konfigurationsaufwand. Für eine Single-Server-Instanz mit moderater Last ist die Performance von SQLite dank SSD/NVMe-Speicher oder schnellen SD-Karten absolut ausreichend. Backups können durch einfaches Kopieren des Dateisystems durchgeführt werden.

---

7 ADR-007: Frontend-Technologie

Status: Akzeptiert

Kontext

Die Benutzeroberfläche muss responsiv sein, soll aber keine unnötige Komplexität durch eine Trennung in Backend-API und Frontend-SPA (Single Page Application) einführen.

Entscheidung

Verwendung der **Django Template Engine** mit **Vanilla JavaScript**.

Begründung

Server-Side Rendering (SSR) via Django Templates ermöglicht eine schnelle Entwicklung und direkte Integration mit dem Backend-Kontext. Für interaktive Elemente wird einfaches JavaScript genutzt, ohne den Overhead großer Frameworks wie React oder Vue. Dies hält den Build-Prozess simpel (kein separater Node.js Build-Step notwendig) und reduziert die Ladezeiten auf dem Client.