 Template

Januar 2023

**Über arc42**

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Template Version 8.2 DE. (basiert auf AsciiDoc Version), Januar 2023

Created, maintained and © by Dr. Peter Hruschka, Dr. Gernot Starke and contributors. Siehe <https://arc42.org>.

# Einführung und Ziele

## Aufgabenstellung

## Qualitätsziele

**Hohe Performance**  
→ Schnelle und effiziente Datenverarbeitung durch asynchrone Verarbeitung (z. B. mit async/await in Node.js).

**Gute Benutzerfreundlichkeit**  
→ Intuitives, stabiles und verständliches UI-Design, gestützt durch Nutzer-Feedback und automatisierte UI-Tests.

**Zuverlässigkeit im Betrieb**  
→ Ständige Systemverfügbarkeit und Datenintegrität durch robuste Backend-Szenarien.

**Wartbarkeit und Erweiterbarkeit**  
→ Strukturierte Codebasis und Pflegefähigkeit durch modulare Szenarien und nachhaltige Entwicklung.

**Skalierbarkeit**   
→ Das System ist darauf ausgelegt, mit wachsenden Anforderungen und Nutzerzahlen ohne Leistungsverlust mitzuwachsen.

## Stakeholder

| Rolle | Kontakt | Erwartungshaltung |
| --- | --- | --- |
| *<User>* | *<>* | *<* *Intuitive Bedienung der App und stabile Funktionen für Trainingsplanung und -durchführung. Erwartet eine performante App mit nachvollziehbarem UI*.> |
| *<Admin>* | *< >* | *<* *Erwartet zuverlässigen Betrieb, einfache Wartbarkeit und Skalierbarkeit bei steigender Nutzeranzahl. >* |

# Randbedingungen

Damit unsere GymTrackerX App eine große Anzahl an Nutzern erreichen kann, sollte die App auf mehreren Betriebssystemen lauffähig sein. Dabei sind IOS und Android Ausreichend. Des Weiteren sollte sie vom Hardwareaufwand nicht zu anspruchsvoll sein um so Nutzer mit schlechterer Hardware nicht durch lange Wartezeiten oder Programm Abbrüchen zu verlieren. Es sollen die Daten der Nutzer auf einer Datenbank gespeichert werden um so eine geräteübergreifende Nutzung zu ermöglichen. Dabei soll diese über eine REST API angesprochen werden.

# Kontextabgrenzung

## Fachlicher Kontext

**<Diagramm und/oder Tabelle>**

**<optional: Erläuterung der externen fachlichen Schnittstellen>**

## Technischer Kontext

**<Diagramm oder Tabelle>**

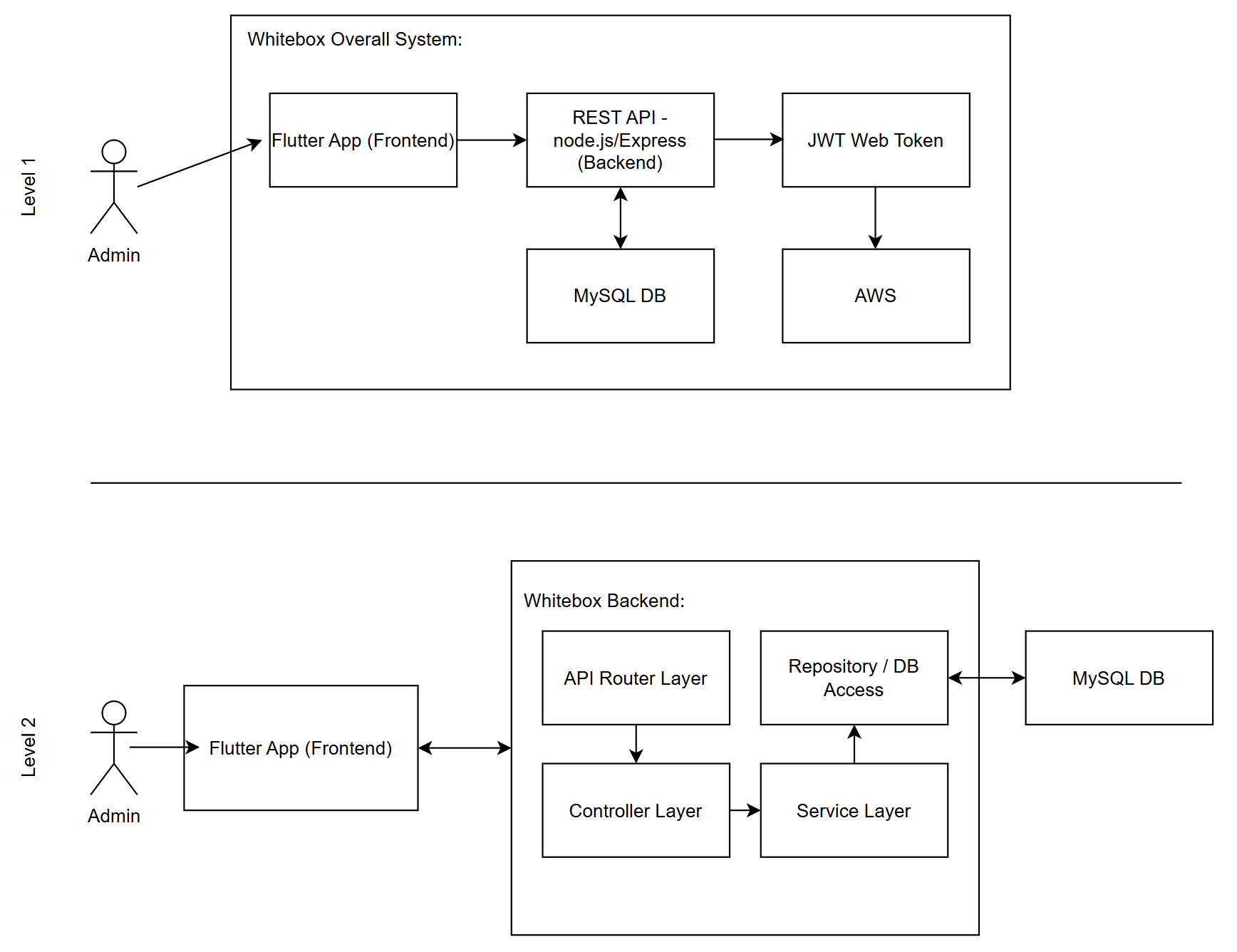
**<optional: Erläuterung der externen technischen Schnittstellen>**

**<Mapping fachliche auf technische Schnittstellen>**

# Lösungsstrategie

# Bausteinsicht

## Whitebox Gesamtsystem



**Begründung**

*Das System besteht aus einer mobilen App zur Erfassung von Workouts, einem REST-basierten Backend zur Datenverarbeitung und -persistierung sowie einer MySQL-Datenbank. Die Kommunikation erfolgt über HTTP/REST.*

**Enthaltene Bausteine**

*Frontend (Flutter App):*

*Mobile App für iOS und Android. Ermöglicht User Login, Workout-Tracking (Übungen, Sets, Reps), Anzeige historischer Daten.*

*Backend (Node.js + Express API):*

*RESTful API, die Anfragen vom Frontend verarbeitet, Authentifizierung durchführt und mit der Datenbank interagiert.*

*Datenbank (MySQL):*

*Speichert Benutzerdaten, Übungen, Workouts, Sets und Wiederholungen.*

*AWS (Cloud-Infrastruktur):*

*Hosting von Backend und Datenbank (z. B. über EC2, RDS, etc.).*

*Authentication (JWT):*

*Authentifizierung & Autorisierung über JSON Web Tokens.*

**Wichtige Schnittstellen**

*Frontend ↔ Backend:*

*REST API (JSON), Auth via JWT im Header*

*Backend ↔ Datenbank:*

*SQL Queries oder ORM (Sequelize, Prisma)*

**Ebene 2:**

**Backend**

*Das Backend ist in ExpressJS geschrieben und stellt eine REST API bereit. Es enthält mehrere Module für Authentifizierung, Workout-Management und User-Verwaltung.*

**Bausteine:**

*API Router Layer:*

*Verwaltet verschiedene Endpunkte wie /auth, /workouts, /exercises, etc.*

*Controller Layer:*

*Businesslogik für Workouts, Sätze, Wiederholungen, Übungen.*

*Service Layer:*

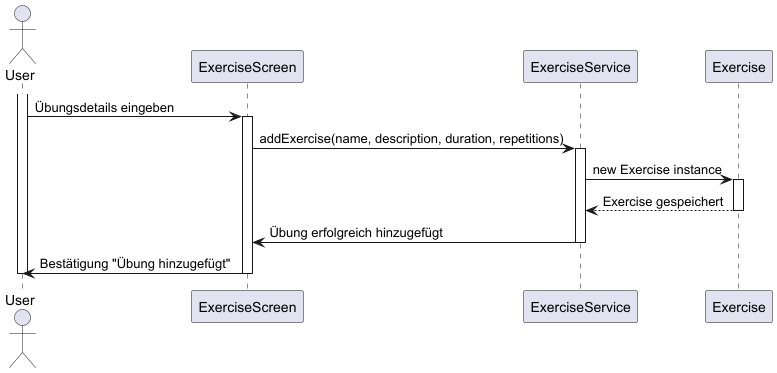
*Verarbeitung und Validierung der Daten, Aufbereitung für die Datenbank.*

*Persistence Layer (Repository):*

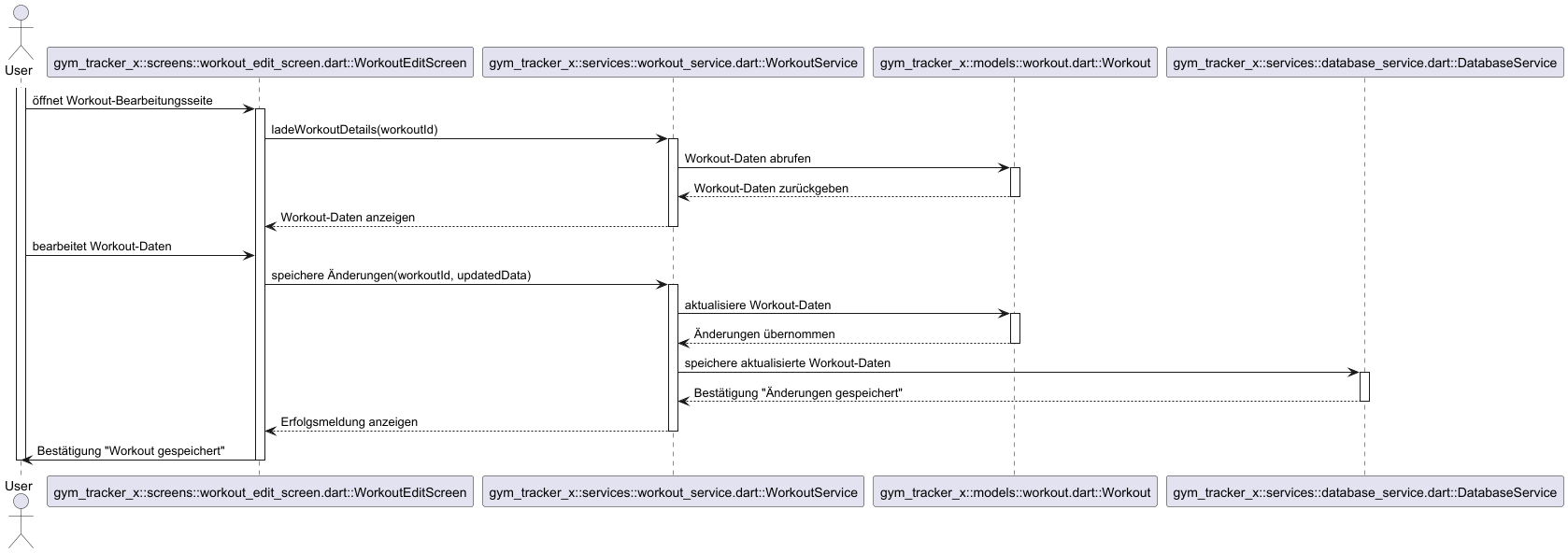
*Kommunikation mit MySQL über einen ORM oder manuelle SQL-Statements.*

# Laufzeitsicht

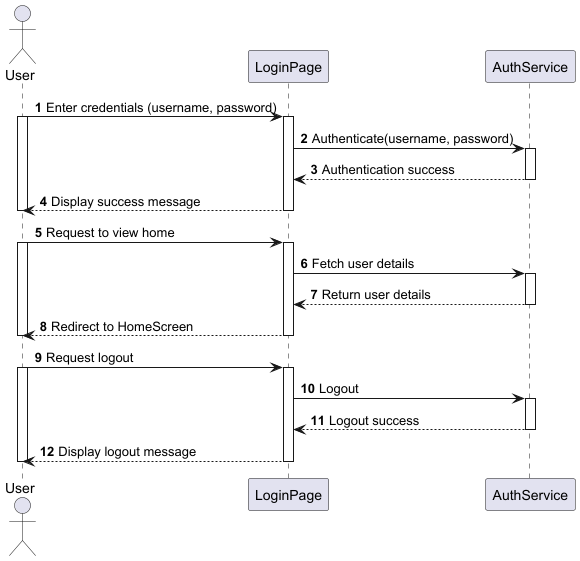
## *Adding Exercise*



## *Start Workout*



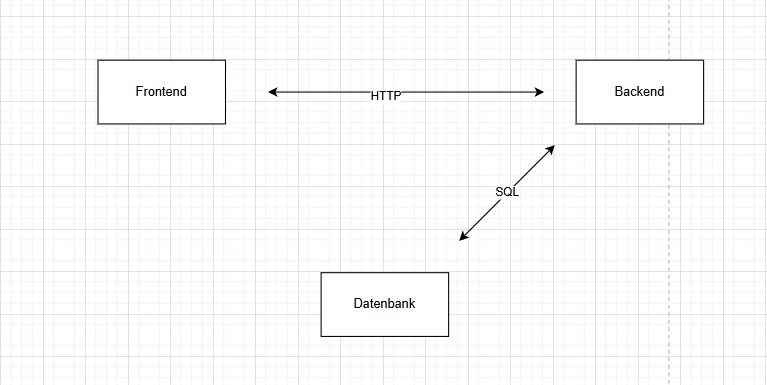
## Login Sequence



# Verteilungssicht

## Infrastruktur Ebene 1

***<Übersichtsdiagramm>***



Begründung

*Die gewählte Schichtenarchitektur trennt Präsentation (Frontend), Geschäftslogik (Backend) und Datenhaltung (Datenbank) klar voneinander.  
Diese Struktur unterstützt Wartbarkeit, Skalierbarkeit und parallele Entwicklung.*

***Vorteile:***

* *Klare Verantwortlichkeiten*
* *Gute Testbarkeit je Schicht*
* *Einfache Wartung und Erweiterung*

***Nachteile:***

* *Höherer initialer Aufwand durch Schnittstellen*
* *Höhere Komplexität bei der Synchronisierung der Schichten*

Qualitäts- und/oder Leistungsmerkmale

** ***Wartbarkeit:*** *Klare Trennung reduziert Nebeneffekte bei Änderungen*

** ***Skalierbarkeit:*** *Backend kann bei Last separat skaliert werden*

** ***Sicherheit:*** *Nur Backend kommuniziert direkt mit Datenbank*

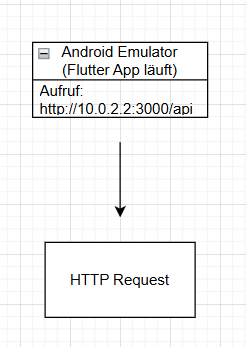
** ***Performance:*** *Caching im Backend möglich*

Zuordnung von Bausteinen zu Infrastruktur

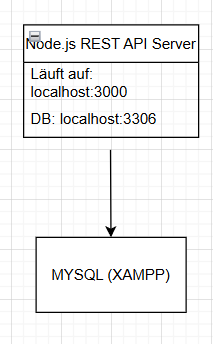
| ***Baustein*** | ***Infrastrukturkomponente*** |
| --- | --- |
| *Flutter-App* | *Bsp. Android-Emulator oder Mobilgerät* |
| *API-Server* | *Lokaler Server auf Port 3000* |
| *Datenbank* | *MySQL* |

## Infrastruktur Ebene 2

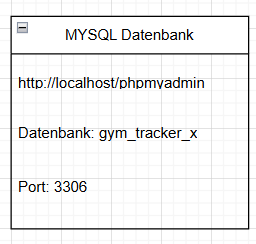
### *<Infrastrukturelement 1>*



### *<Infrastrukturelement 2>*



### *<Infrastrukturelement 3>*



# Querschnittliche Konzepte

## *Plattformübergreifende Entwicklung*

* *Die Entscheidung für Flutter ermöglicht die Nutzung einer gemeinsamen Codebasis für iOS und Android, wodurch Entwicklungsaufwand und Wartung reduziert werden. Flutter bietet zudem native Performance und eine moderne UI-Komponentenbibliothek, die ein konsistentes Nutzererlebnis auf beiden Plattformen gewährleistet.*

## *Synchronisation und Datenhaltung*

* *GymTrackerX speichert Benutzerdaten in einer MySQL-Datenbank, die über eine REST-API angesprochen wird. Die Synchronisierung erfolgt in Echtzeit, um geräteübergreifenden Zugriff auf Trainingsdaten zu ermöglichen. Dies erlaubt den Nutzern eine konsistente Nutzung der App auf mehreren Endgeräten.*

## *Modularität und Wartbarkeit*

* *Durch eine klare Schichtenarchitektur (Frontend – Backend – Datenbank) wird die Trennung von Verantwortlichkeiten gewährleistet. Dies ermöglicht parallele Entwicklungsteams, gute Testbarkeit und eine nachhaltige Erweiterbarkeit.*

## Qualitätssicherung und CI/CD

* *Ein durchgängiger CI/CD-Prozess mit automatisierten Tests (Unit, Integration, UI) sorgt für stabile Release.*

## Benutzerfreundlichkeit

* *Das intuitive UI-Design wird durch Nutzerfeedback und automatisierte UI-Tests kontinuierlich verbessert. Zusätzlich steigert ein Punktesystem, basierend auf dem Schwierigkeitsgrad der Übungen, die Motivation der Nutzer, regelmäßig zu trainieren.*

# Architekturentscheidungen

**Einsatz einer Schichtenarchitektur (Frontend / Backend / Datenbank)**

**Kontext:**  
Für die Fitness-App ist eine klare Trennung der Verantwortlichkeiten zwischen Benutzeroberfläche, Geschäftslogik und Datenhaltung notwendig. Dies soll Wartbarkeit, Skalierbarkeit und eine saubere Weiterentwicklung der Anwendung ermöglichen.

**Entscheidung:**  
Einsatz einer klassischen Schichtenarchitektur mit drei Ebenen:

* **Frontend** für Benutzerinteraktion
* **Backend** für Geschäftslogik und Schnittstellen
* **Datenbank** für dauerhafte Speicherung

**Begründung:**

* Klare Verantwortlichkeiten zwischen Schichten
* Parallele Entwicklung von Frontend und Backend möglich
* Gute Testbarkeit und Erweiterbarkeit
* Bewährtes Architekturmodell mit breiter Unterstützung

**Alternativen:**

* Monolithische Architektur (UI, Logik und Datenzugriff eng gekoppelt)
* Microservice-Architektur mit mehreren unabhängigen Services

**Konsequenzen:**

* **Positiv:** Verbesserte Wartbarkeit, klare Schnittstellen, einfache Erweiterbarkeit
* **Negativ:** Höherer initialer Aufwand, Pflege der Schnittstellen notwendig

**Entscheidung: Flutter als Frontend-Technologie**

**Kontext:  
Die App soll auf Android- und iOS-Geräten lauffähig sein, ohne für jede Plattform separate native Anwendungen zu entwickeln. Es wird eine moderne, reaktive Benutzeroberfläche angestrebt, die schnell entwickelt und gut wartbar ist.**

**Entscheidung:  
Verwendung von Flutter als plattformübergreifendes UI-Framework für das Frontend.**

**Begründung:**

* **Eine gemeinsame Codebasis für Android und iOS spart Entwicklungszeit**
* **Flutter bietet gute Performance durch native Kompilierung**
* **Umfangreiche und flexible UI-Komponenten ermöglichen eine konsistente Gestaltung**
* **Hot-Reload beschleunigt die Entwicklung**

**Alternativen:**

* **Native Entwicklung mit Kotlin (Android) und Swift (iOS)**
* **React Native als alternatives Cross-Plattform-Framework**

**Konsequenzen:**

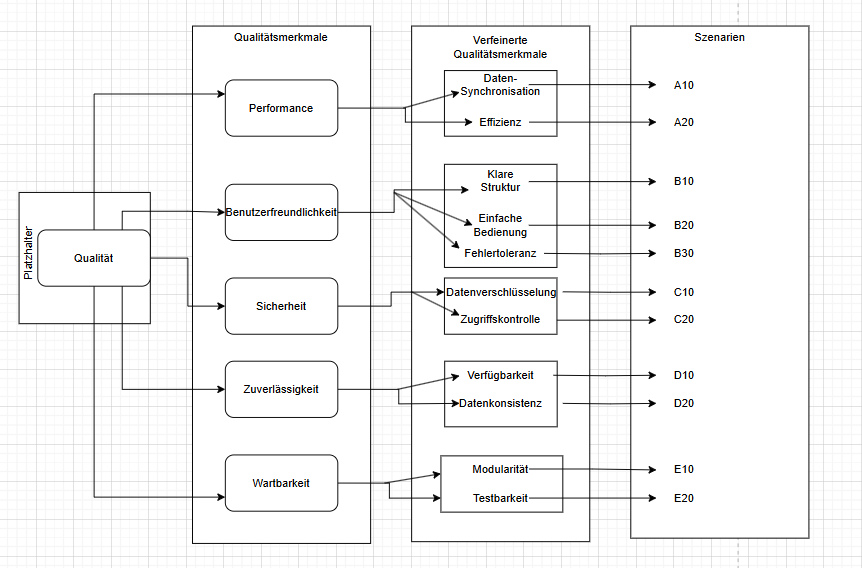
* **Positiv: Schnelle plattformübergreifende Entwicklung, konsistentes UI, geringerer Wartungsaufwand**
* **Negativ: Größere App-Größe, komplexere Anbindung an plattformspezifische Funktionen**

# Qualitätsanforderungen

**Weiterführende Informationen**

Siehe [Qualitätsanforderungen](https://docs.arc42.org/section-10/) in der online-Dokumentation (auf Englisch!).

## Qualitätsbaum



## 

## Qualitätsszenarien

**1. Performance:**

* Die Szenarien A10 und A20 sind für die effiziente und schnelle Datenverarbeitung verantwortlich

**2. Benutzerfreundlichkeit:**

* wird durch ein intuitives, stabiles und verständliches Design aufgelistet (B10, B20 und B30)

**3. Sicherheit:**

* Durch C10 und C20 wird der Schutz von Daten und Zugriffsregulierung sichergestellt

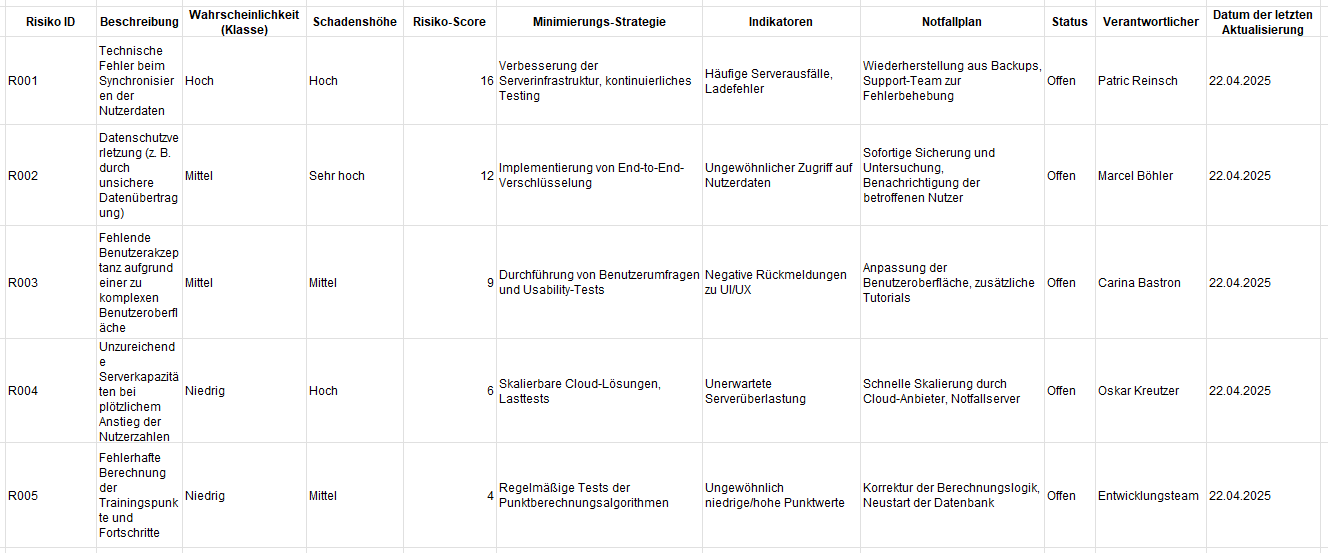
**4. Zuverlässigkeit:**

* Die Szenarien D10 und D20 sorgen für eine ständige Verfügbarkeit und Datenintegrität

**5. Wartbarkeit:**

* ermöglicht eine nachhaltige Pflege und Erweiterung des Systems durch die Szenarien E10 und E20

# Risiken und technische Schulden



# Glossar

| **Begriff** | **Definition** |
| --- | --- |
| **Flutter** | Open-Source UI-Toolkit von Google zur Entwicklung plattformübergreifender mobiler Anwendungen. |
| **CI/CD** | Continuous Integration / Continuous Deployment – Automatisierte Prozesse für das Testen und Ausliefern von Software. |
| **UML** | Unified Modeling Language – Standardisierte Sprache zur Modellierung von Softwarearchitekturen. |
| **Bausteinsicht** | Sichtweise in der Softwarearchitektur, die die statische Struktur des Systems in Module und deren Beziehungen beschreibt. |
| **Laufzeitsicht** | Sichtweise, die das dynamische Verhalten des Systems während der Ausführung darstellt, z. B. durch Sequenzdiagramme. |
| **RMMM-Liste** | Risk Mitigation, Monitoring, and Management – Liste zur Identifikation und Behandlung von Projektrisiken. |
| **Debug-Build** | Softwareversion, die für Testzwecke mit zusätzlichen Debugging-Informationen erstellt wird. |
| **Artefakt** | Ergebnis eines Build-Prozesses, z. B. eine APK-Datei, die für Tests oder Releases verwendet wird. |
| **Projektvision** | Langfristiges Ziel und Zweck des Projekts, z. B. die Entwicklung einer benutzerfreundlichen Fitness-App. |