**Schichtenarchitektur**

**Kontext und Problemstellung**  
Die Fitness-App soll modular, wartbar und skalierbar entwickelt werden. Eine klare Trennung der Verantwortlichkeiten ist entscheidend, um unterschiedliche Anforderungen wie UI-Änderungen, Geschäftslogik-Erweiterungen oder Datenbankmigrationen unabhängig voneinander umsetzen zu können.  
**Frage:**  
Wie kann die Architektur der Anwendung so gestaltet werden, dass sie sowohl wartbar als auch erweiterbar bleibt und gleichzeitig eine klare Trennung der Verantwortlichkeiten gewährleistet?

**Betrachtete Varianten**

* **Variante 1:** Schichtenarchitektur mit Frontend, Backend und Datenbank
* **Variante 2:** Monolithische Architektur mit eng gekoppelter UI, Logik und Datenzugriff
* **Variante 3:** Microservice-Architektur mit getrennten Services für Benutzer, Übungen, Trainingspläne etc.

**Entscheidung**  
**Gewählte Variante:** *Schichtenarchitektur mit Frontend, Backend und Datenbank*  
Diese Architektur sorgt für eine klare Trennung der Zuständigkeiten, ist bewährt, gut dokumentiert und erlaubt eine saubere Weiterentwicklung der Anwendung.

**Begründung:**

* **Variante 1 (Schichtenarchitektur):**  
  **Vorteile:**
  + Gute Trennung von UI, Geschäftslogik und Datenzugriff
  + Ermöglicht parallele Entwicklung von Frontend und Backend
  + Bessere Testbarkeit der einzelnen Schichten
  + Klare Verantwortlichkeiten und einfache Wartbarkeit  
    **Nachteile:**
  + Initial höherer Aufwand durch Schnittstellendefinition zwischen Frontend und Backend
  + Komplexität bei der Koordination zwischen Schichten
* **Variante 2 (Monolithisch):**  
  **Vorteile:**
  + Schnellere initiale Entwicklung
  + Geringe Komplexität beim Start  
    **Nachteile:**
  + Schwer wartbar und schlecht skalierbar bei wachsendem Funktionsumfang
  + Änderungen in einem Bereich können ungewollt andere Bereiche beeinflussen
* **Variante 3 (Microservices):**  
  **Vorteile:**
  + Hohe Skalierbarkeit und Unabhängigkeit der einzelnen Module
  + Ermöglicht unabhängige Releases und Technologiestacks  
    **Nachteile:**
  + Deutlich höherer Entwicklungs- und Infrastrukturaufwand
  + Erfordert erfahrenes Team und umfassendes Monitoring

**Status**  
**Angenommen**

**Konsequenzen**

* **Gut, weil:**
  + Die Struktur ist übersichtlich, wartbar und leicht verständlich für Entwickler
  + Änderungen im Frontend oder Backend können unabhängig voneinander erfolgen
  + Die Lösung ist zukunftssicher und gut dokumentierbar
  + Technologische Weiterentwicklung in einzelnen Schichten möglich (z. B. Wechsel der Datenbank oder Migration von Frameworks im Frontend)
* **Schlecht, weil:**
  + Kommunikation zwischen Frontend und Backend muss sauber definiert und gewartet werden (z. B. API-Definitionen)
  + Höherer initialer Aufwand für Setup und Abstimmung
  + Entwickler müssen sich auf die Trennung und die Interoperabilität konzentrieren (z. B. CORS, Authentifizierung, etc.)