005-Webshop

Fullstack Aufgabe
Anforderungsniveau: Schwer

1 Ziel der Aufgabe

Ziel dieser Aufgabe ist es, Ihre Fähigkeiten in moderner Fullstack-Entwicklung unter Beweis zu stellen. Sie erstellen eine kleine Webshop-Anwendung mit Front- und Backend, setzen eine relationale Datenbank ein und orchestrieren beides in Containern.

2 Aufgabenbeschreibung

Implementieren Sie eine Fullstack-Web-Anwendung mit folgenden Komponenten:

1. **Datenbank**
   - Verwenden Sie MySQL oder PostgreSQL als relationale Datenbank.
   - Definieren Sie ein einfaches Schema für Produkte, Benutzer und Bestellungen.
2. **Backend-API**
   - Bevorzugt in Rust (z. B. mit Warp oder Actix-web); alternativ in Node.js/Express.
   - REST- oder GraphQL-Endpunkte für:
     - Benutzer-Registrierung, Login (JWT-basiert)
     - Produktverwaltung (CRUD)
     - Warenkorb und Bestellverarbeitung
   - Serverseitige Validierung und angemessene HTTP-Statuscodes.
3. **Frontend**
   - Mit Next.js (React) als Single-Page-App.
   - Seiten/Komponenten für:
     - Nutzer-Authentifizierung (Registrierung/Login)

    - Produktübersicht, Detailseite
    - Warenkorb und Bestellübersicht
  - State-Management über React Context oder eine Bibliothek Ihrer Wahl.

4. **Containerisierung**
   - Docker-Container für Backend und Frontend jeweils separat.
   - Nutzung von `docker-compose.yml` zur Orchestrierung und zum Networking.
   - Beide Container müssen über das interne Docker-Netzwerk kommunizieren.

3 Anforderungen

- **Code-Organisation**
  - Klare Modulstruktur: `frontend/`, `backend/`, `db/`.
  - Trennung von Präsentation, Logik und Datenzugriff.
- **Sicherheit**
  - Sichere Passwort-Hashes (z. B. bcrypt oder argon2).
  - Schutz vor XSS, CSRF und SQL-Injection.
- **Dokumentation**
  - README mit Setup-, Build- und Run-Anweisungen.
  - API-Referenz (Swagger/OpenAPI oder Markdown im Repo).
- **Tests**
  - Unit-Tests für Backend-Logik und Frontend-Komponenten.
  - Integrationstests für Authentifizierung und Endpunkte (optional).

4 Optionale Erweiterungen

1. **Payment Gateway**
   - Integration eines Mock-/Sandbox-Zahlungsanbieters (z. B. Stripe).

2. **Suchfunktion**
    - Volltextsuche über Produkte (z. B. mit einer SQL- oder NoSQL-Extension).
3. **CI/CD**
    - GitHub Actions oder GitLab-CI Pipeline für Build, Test und Deployment.

5 Expertenanforderungen

Für besonders tiefgehende Kenntnisse können Sie folgende Kann-Kriterien umsetzen:

- **Caching & Performance**
    - Einsatz von Redis oder In-Memory-Cache für häufige Abfragen.
    - Load-Balancing und horizontale Skalierungsvorbereitung.
- **Echtzeit-Updates**
    - WebSocket- oder Server-Sent-Events für Lagerbestands- und Bestellstatus-Updates.
- **Rollen- und Berechtigungsmodell**
    - Admin-Panel mit spezifischen Admin-Rollen.
    - Feingranulare Zugriffskontrolle (RBAC) auf Endpunkte.
- **Monitoring & Logging**
    - Zentralisiertes Logging (z. B. ELK-Stack) und Health-Checks.
    - Performance-Metriken mit Prometheus/Grafana.
- **Container Security**
    - Multi-Stage Docker-Builds, Image-Scanning auf Schwachstellen.
    - Geheimnis-Management (z. B. Docker Secrets oder Vault).

6 Abgabe und Bewertung

- **Lieferumfang**
  - Öffentlicher Git-Repository-Link mit allen Quellcodes und Docker-Konfigurationen.
  - README mit klaren Anweisungen für `docker-compose up`.
  - Optional: Live-Demo-URL.
- **Bewertungskriterien**
  - Funktionalität: Vollständiges User- und Produkt-Management.
  - Architektur: Saubere Trennung und Container-Orchestrierung.
  - Sicherheit & Performance: Absicherung, Caching, Skalierbarkeit.
  - UX/UI: Intuitive Bedienung, responsives Design.
  - Optionales & Experten: Umsetzung der erweiterten Kriterien.

Viel Erfolg bei der Umsetzung!
Stand: 31. Mai 2025