

Frontend-Aufgabe: Progressive Product Catalog (ohne Backend)

Aufgabenstellung

Stand: 26. Oktober 2025

1 Ziel der Aufgabe

Ziel dieser Aufgabe ist es, Ihre Kenntnisse in moderner, robuster Frontend-Entwicklung mit React und TypeScript zu prüfen. Die Applikation darf keine serverseitigen Abhängigkeiten haben — alle Daten werden lokal gehalten (z. B. mittels `localStorage` oder `IndexedDB`). Es sollen Architektur, Typisierung, Zugänglichkeit, Performance und Tests demonstriert werden.

2 Kurzbeschreibung

Erstellen Sie eine Single-Page-Applikation „Progressive Product Catalog“, in der Benutzer Produkte betrachten, filtern, sortieren und in einem lokalen Warenkorb verwalten können. Persistenz erfolgt rein clientseitig. Die App soll offline-freundlich sein (Progressive Web App Ansätze) und eine gute Developer-Experience besitzen (Scripts, README, saubere Struktur).

3 Anforderungen

1. Framework und Tooling

- React mit TypeScript (z. B. Next.js oder Create React App mit TypeScript).
- Paketmanager: npm oder yarn.
- Build-Skripte: `dev`, `build`, `start`, `test`, ggf. `lint`.

2. Daten und Persistenz

- Produkte kommen aus einer lokalen JSON-Datei (z. B. `src/data/products.json`).
- Änderungen des Warenkorbs und Benutzerpräferenzen werden in `localStorage` oder `IndexedDB` persistiert.
- Keine Remote-API-Aufrufe.

3. Kern-Funktionalitäten

- Produktliste mit Bild, Name, Preis, Kurzbeschreibung, Tags/Kategorie.
- Produkt-Detailseite/Modal mit erweiterten Informationen.
- Filter (Kategorie, Preisbereich, Tags) und Sortierung (Preis, Name, Beliebtheit).
- Suchfunktion mit Debounce (z. B. 300ms).
- Lokaler Warenkorb: Hinzufügen/Entfernen von Artikeln, Mengen, Zwischensumme, persistenter Checkout-Simulator (ohne Zahlungsintegration).
- Zustandshandling: React Context oder State-Management-Library (z. B. Zustand/Redux) — strikt typisiert.

- Lade- und Fehlerzustände, auch wenn Daten lokal sind (z. B. simulierte Verzögerung).
4. UI / Styling
- Responsives Design (Desktop, Tablet, Mobile).
 - Minimalistisches, konsistentes Styling (CSS Modules, Tailwind oder Styled Components).
 - UI-Elemente: Header mit Navigation, Suchfeld, Filter-Panel, Produkt-Grid, Footer.
5. Accessibility (A11y)
- Semantische HTML-Elemente, Tastaturnavigation, ARIA-Attribute wo nötig.
 - Farbkontrast prüfen (WCAG AA).
6. Typisierung
- Alle Komponenten und Hooks mit TypeScript typisiert.
 - Keine Verwendung von `any`.
7. Tests
- Unit-Tests für Kern-Logik (z. B. Filter-/Sortierfunktionen).
 - Komponententests (React Testing Library) für wichtige UI-Flows (Hinzufügen in Warenkorb, Filter anwenden, Suche).
8. Dokumentation
- README mit Installations- und Startanleitung sowie kurzer Erklärung der Architektur.
 - Beschreibe, wie Persistenz funktioniert und wie Tests ausgeführt werden.

4 Vorschläge für optionale Erweiterungen

- Offline-Unterstützung: Service Worker (Workbox) + Offline-Fallback-Seiten.
- Internationalisierung (i18n) mit mindestens zwei Sprachen.
- Drag-and-Drop zur Sortierung einer Wunschliste (HTML5 Drag API oder react-beautiful-dnd).
- Persistenz via IndexedDB (z. B. mit idb) statt localStorage.
- Dark Mode / Theming mit CSS Custom Properties.
- Storybook für visuelle Komponenten-Dokumentation.
- Performance-Optimierungen: Code-Splitting, lazy-loading von Bildern, Bild-Optimierung (srcset).

Viel Erfolg bei der Umsetzung!