FairShop

Entwickler Dokumentation

2026-01-24

Inhaltsverzeichnis

Chapter 1. Architecture Notebook

1.1. Architekturziele

- Langlebigkeit
- Sicherheit & Schutz
- Minimalismus
- Unabhängigkeit

1.2. Annahmen und Abhängigkeiten

1.2.1. Standort

Die Anwendung muss kompatibel mit den Anforderungen des deutschen Gesetzes sein. Dazu zählt zum Beispiel auch die DSGVO.

1.2.2. Teilzeit

Alle Team-Mitglieder bearbeiten das Projekt in Teilzeit. Es biete sich daher an, wenn möglich, so gut wie möglich auf bereits bekannte Technologien und Konzepte aufzubauen.

1.2.3. Bekannte Frameworks & Technologien

Folgende Frameworks sind dem Team bekannt und finden daher Anwendung bzw. wurden/werden evaluiert:

- Vue.js - UI Web Anwendung
- Nuxt - Full-Stack Vue.js Framework
- FastAPI - API
- Uvicorn - ASGI web server for Python
- Nginx - HTTP web server
- Linux - OS
- Docker - Deployment

1.2.4. VM

Die Resourcen zum Bereitstellen unserer Anwendung beschränken sich auf eine VM unter dem Betriebssystem Ubuntu. Diese ist nur aus dem internen Netzwerk der HTW Dresden oder über eine VPN Verbindung zu erreichen.

1.2.5. API

Da die API zwischen Datenbank und User Interface vermittelt, wird sie als Technologie-Treiber

identifiziert. Datenbank, User Interface und API mussen miteinander kompatibel sein.

1.3. Architektur-relevante Anforderungen

1.3.1. Kompatibilität (sehr hoch)

Kompatibilität ist ein wichtiger Grundsatz, um nachhaltige Anwendungen zu entwickeln und die Langlebigkeit der Architektur zu unterstützen. Dabei sollen diese einen höheren Stellenwerte als Funktionalität haben. Deshalb verzichten wir darauf Web-Client spezifische Optimierungen zu nutzen

1.3.2. Platformunabhängig (sehr hoch)

Endgeräte kommen mit vielen Verschiedenen Betriebssystemen. Die Software sollte so viele Platformen wie möglich unterstützen, also möglichst unabhängig von den Präferenzen des Nutzers funktionieren.

1.3.3. Responsive (hoch)

Displays als User Interface kommen in vielen verschiedenen Varianten. Die Anwendung sollte möglichst alle typischen Standards abdecken und so für einge große Menge an Endgeräten eine angenehme User Experience ermöglichen.

1.3.4. Modularität (mittel)

Um eine bessere Wartbarkeit und Skalierung des komplexen Systems zu ermöglichen und damit die Langlebigkeit der entwickelten Architektur zu unterstützen, wird die Anwendung grob in Frontend (User Interface) und Backend (Funktionalitäten) geteilt. Diese wiederum sind in einzelne Komponenten / Module aufgeteilt die modulare Aufgaben erfüllen.

1.3.5. Offline (mittel)

Die primäre Nutzungsumgebung sind geschlossene Gebäude wie Supermärkte, in denen nicht immer eine stabile Internetverbindung angenommen werden kann. Deshalb muss unsere Software ihre grundsätzlichen Funktionalität auch ohne Internet ausführen können und somit unabhängig von der gegebenen Internetverbindung funktionieren. Oder die zeitweise Trennung vom Internet vor dem Nutzer verstecken.

1.4. Entscheidungen, Nebenbedingungen und Begründungen

1.4.1. Progressive Web Application

Wir haben uns dazu entschieden das Produkt als Progressive Web Application umzusetzen. Dies erfüllt Unabhängigkeit von Betriebssystemen und Hardware. Auch lässt sich damit die Offline Anforderung erfüllen.

1.4.2. User Interface

Das User Interface wird mit Vue.js entwickelt und mit Vite gebaut. Vue.js ist ein modernes Framework für Web Anwendungen (SPA), welches Modulariät durch Single File Components stark begünstigt. Zudem können mit Vite gebaute Anwendungen können von einem Nginx Server gehostet werden, welcher wiederum auf der bereitgestellten VM problemlos läuft.

1.4.3. API

Die Schnittstelle soll dem Architektur prinzip einer REST-API folgen. Hierfür bietet es sich an FastAPI zu nutzen. Dies kann mit Uvicorn zugänglich gemacht werden. Uvicorn läuft auf der bereitgestellten VM. Zudem harmoniert FastAPI sehr gut mit MySQL und SQLAlchemy.

1.4.4. Datenbank

Die Verwaltung der Daten erfolg über eine relationale Datenbank. Auch wenn SQLAlchemy viele Datenbanksysteme unterstütz, ist hier MySQL gewählt worden, da sie sicher auf der bereitgestellten VM läuft.

1.5. Schichten oder Architektur-Framework

Wir oreintieren uns an einer Layered Architecture, um klar und minimalistische Struktur, sowie deren Erweitbarkeit zu gewährleisten. Diese Struktur ermöglicht eine klare Trennung von Verantwortlichkeiten und erleichtert sowohl die Entwicklung als auch die Fehlersuche und Erweiterung.

1.5.1. Frontend - Single Page Application

- UI im Web-Client
- Verwaltet die Bennutzerinteraktion und Darstellung
- nutzt Schnittstelle

1.5.2. Schnittstelle - REST-API

- Implementiert die Geschäftslogik
- Überprüft, verarbeitet und validiert Anfragen aus dem Frontend
- schreibt und holt zum/vom Backend

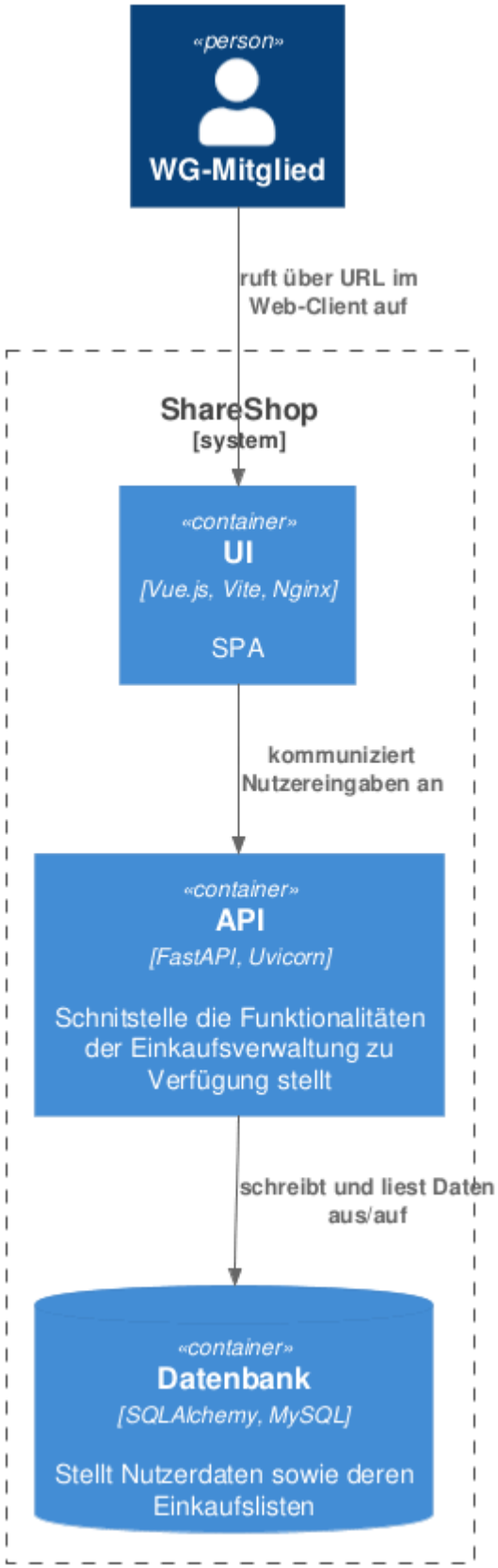
1.5.3. Backend - Datenbankmanagementsystem

- Verwaltet den Zugriff auf die relationale Datenbank
- Datenstruktur wird über ORM definiert

1.6. Architektursichten

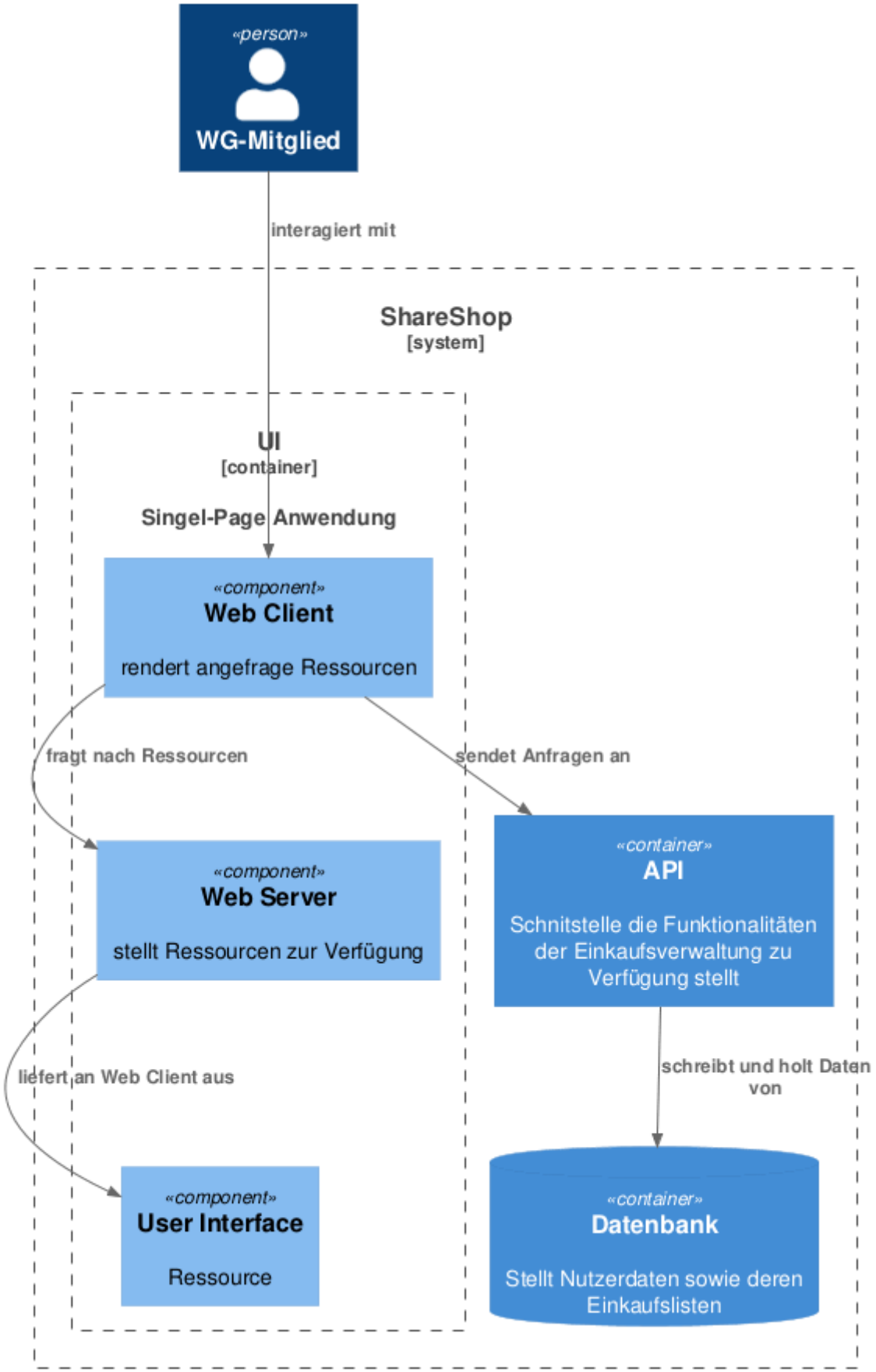
1.6.1. Kontextabgrenzung

1.6.2. Container

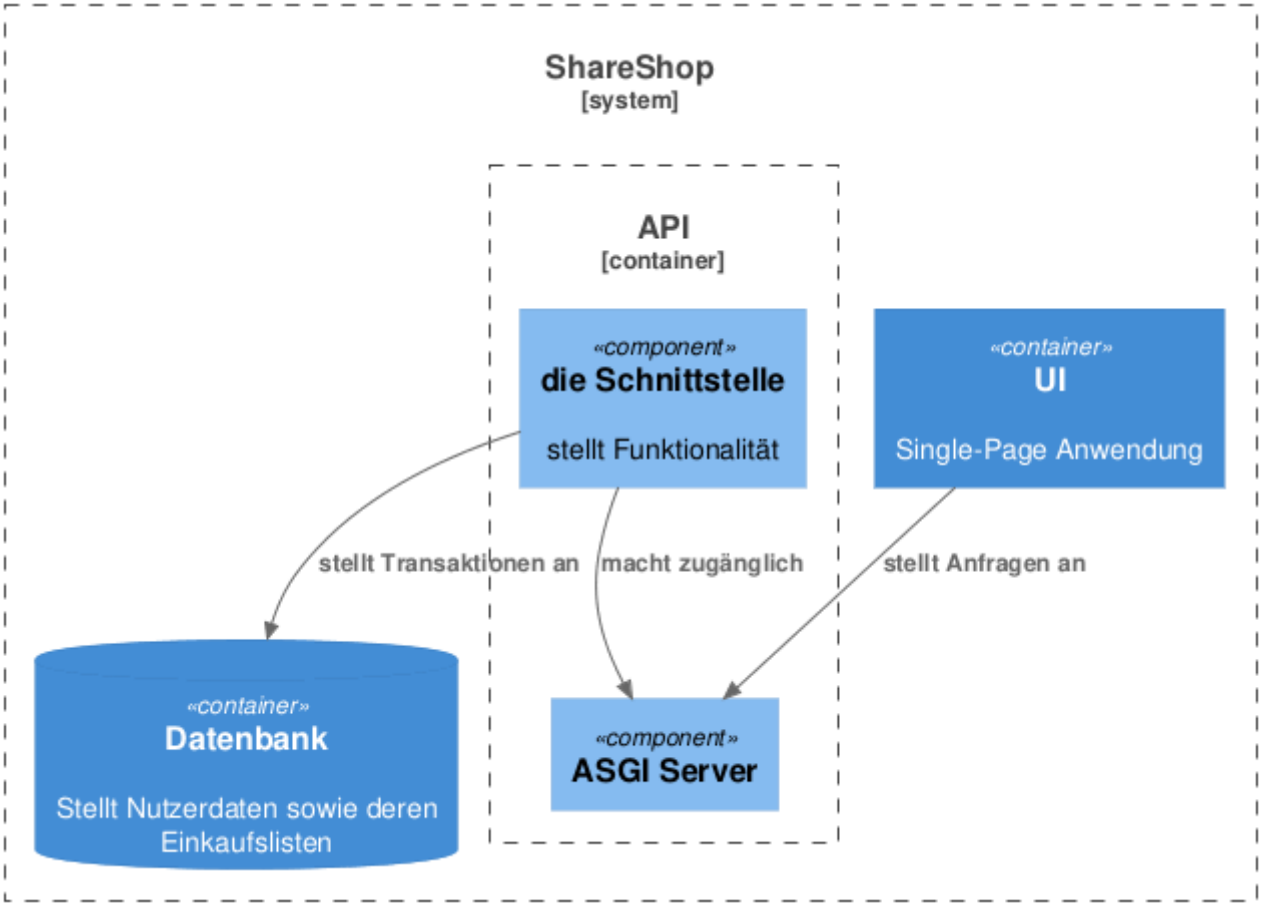


1.6.3. Components

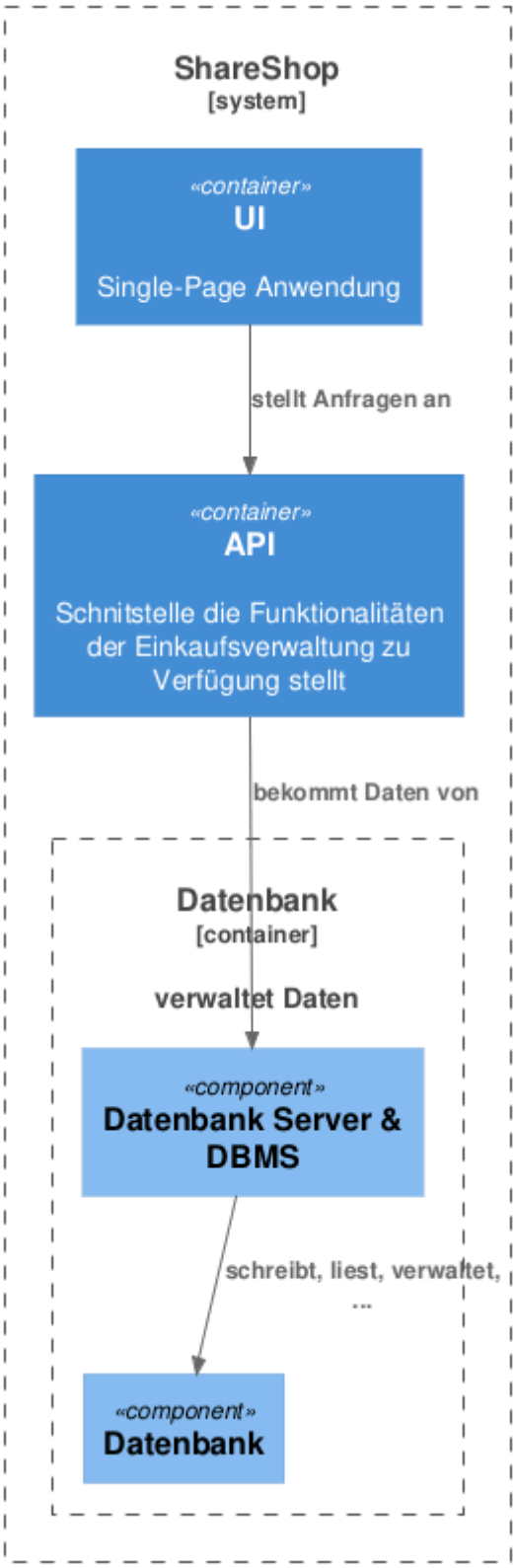
Component: User Interface



Component: API



Component: Datenbank



1.6.4. Verteilungsdiagramm

Alle 3 Components (User Interface, API & Datenbank) können auf dedizierten Knoten gehostet werden. Es wird aber auch ein Docker-Compose-File zur Verfügung gestellt, der alle Components auf einem Knoten installiert.