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

Einführung und Ziele

Die Rezeptapp beugt Lebensmittelverschwendung vor, indem sie gezielt für übriggebliebene Lebensmittel Rezepte vorschlägt.

Aufgabenstellung

Priorität	Beschreibung
1	Unverbrauchte Lebensmittel sollen durch eine Angabe vom User in der App, passende Rezepte vorgeschlagen bekommen
2	Eine gut gestaltete Architektur soll das Verbessern für zukünftige Anwendungen vereinfachen
3	Eine schnelle Entwicklung soll durch gute Teamabsprachen gewährleistet sein
4	Durch gut gewarteten Quellcode sollen die User eine fehlerfreie Anwendung erfahren

5	Durch moderne Design Entscheidungen, sollen die User eine intuitive verwendung der App wahrnehmen

Qualitätsziele

Qualitätsziele	Szenarios
Intuitive Benutzeroberfläche	Durch ein einheitliches Layout der Seiten und Aktivitätsbuttons die universel verständlich sind, sollen User sich immer zurechtfinden
Wartbarkeit des Quellcodes	Durch ein strukturiertes Projekt können später leichter weitere Funktionen eingebaut werden
Zuverlässige Suchergebnisse	Eingegebene Zutaten in der Suchleiste sollen zuverlässig und passende Rezepte anzeigen

Randbedingungen

technisch	organisatorisch	politisch
Keine kostspieligen Entwicklungsapperaturen vorhanden	• Kein Budget • Begrenzte Stundenzahl pro Woche der Beteiligten • Ausfallbedingte Zeitperioden durch Prüfungsphasen	---

Kontextabgrenzung

Technischer Kontext

Frontend (Next.js) zu MySQL-Datenbank:

- Kanal: Über eine Rust-API
- Übertragungsmedium: Internet

Zutateneingabe:

- Fachliche Eingabe: besitzende Zutat
- Technische Schnittstelle: Datenbankabfrage

Lösungsstrategie

Architekturvision:

Unsere Rezept-App hat das Ziel, Benutzern das einfache Auffinden von Rezepten basierend auf den Zutaten, die sie zu Hause haben, zu ermöglichen. Wir streben eine benutzerfreundliche, performante und skalierbare Lösung an.

Designentscheidungen:

Die Wahl von Next.js für das Frontend ermöglicht uns eine effiziente und dynamische Benutzeroberfläche. Rust wurde für die Entwicklung der REST-API ausgewählt, um von seiner Geschwindigkeit und Systemsprachen-Performance zu profitieren. Die MySQL-Datenbank wurde aufgrund ihrer weit verbreiteten Verwendung und ihrer Fähigkeit, komplexe Abfragen zu bewältigen, gewählt.

Bausteinsicht

Whitebox Gesamtsystem

Begründung

Unsere Architektur basiert auf Prinzipien wie Modularität, um verschiedene Teile der Anwendung getrennt zu halten, Wiederverwendbarkeit, um Codeeffizienz zu fördern, und der klaren Trennung von Frontend und Backend.

1. Frontend (Next.js):

Unser Frontend umfasst eine ansprechende Benutzeroberfläche mit einem Suchfeld, das es Benutzern ermöglicht, Zutaten einzugeben. Die Kommunikation mit der REST-API erfolgt über HTTP-Anfragen. Die Darstellung der Rezeptvorschläge wird durch dynamisches Rendern erreicht.

2. Backend (Rust REST-API):

Die REST-API bietet Endpunkte wie /recipes/get, um die Rezeptsuche zu ermöglichen. Hier erfolgt die Verarbeitung der Suchanfragen und die Interaktion mit der MySQL-Datenbank. Die API ist darauf ausgelegt, Daten effizient und sicher zu verarbeiten und an das Frontend zu liefern.

3. Datenbank (MySQL):

Unsere MySQL-Datenbank enthält relevante Tabellen wie recipes, ingredients und recipes_ingredients. Die Struktur ermöglicht effiziente Abfragen und die Verknüpfung von Rezepten und Zutaten. Die REST-API führt Abfragen durch, um Rezepte basierend auf den eingegebenen Zutaten zu extrahieren.

4. Such-Algorithmus:

Unser Suchalgorithmus berücksichtigt die Übereinstimmung der eingegebenen Zutaten mit den Rezepten. Zusätzliche Faktoren wie die Beliebtheit der Rezepte können in die Suche einbezogen werden, um den Benutzern relevante und attraktive Vorschläge zu bieten.

5. Benutzerverwaltung:

Die Benutzerverwaltung ermöglicht die Authentifizierung und Autorisierung. Benutzer können Rezepte, die ihnen gefallen, speichern und verwalten, wobei Informationen in der user_recipes Tabelle gespeichert werden.

Laufzeitsicht

Die Laufzeitsicht der Rezept-App bietet einen detaillierten Einblick in die Interaktionen zwischen den verschiedenen Komponenten während der Ausführung.

1. Benutzerinteraktion über das Frontend (Next.js):

Der Benutzer interagiert mit der Anwendung über die Benutzeroberfläche, die von Next.js bereitgestellt wird. Das Suchfeld ermöglicht es Benutzern, Zutaten einzugeben, die sie zu Hause haben. Nach Eingabe startet der Benutzer die Suche, indem er eine Anfrage an die entsprechende API-Endpunkt-URL sendet.

2. Kommunikation mit der REST-API (Rust):

Die Next.js-Anwendung sendet HTTP-Anfragen an die Rust REST-API, die als Backend für die Rezept-App fungiert. Die API verarbeitet die Anfragen und führt die erforderlichen Operationen aus, um Rezeptdaten aus der MySQL-Datenbank abzurufen. Dies beinhaltet die Verarbeitung der Suchanfrage, die Extraktion relevanter Informationen und die Rückgabe von Daten im JSON-Format.

3. Datenbankabfragen (MySQL):

Die REST-API interagiert mit der MySQL-Datenbank, um die Rezeptdaten zu erhalten. Dies beinhaltet Abfragen wie das Suchen von Rezepten basierend auf den eingegebenen Zutaten. Die Datenbank liefert die Ergebnisse an die REST-API zurück, die dann diese Informationen für die weitere Verarbeitung nutzt.

4. Verarbeitung der Suchergebnisse:

Die REST-API verarbeitet die von der Datenbank erhaltenen Suchergebnisse und filtert diese gemäß den Suchkriterien, einschließlich der Verfügbarkeit von Zutaten. Der Algorithmus berücksichtigt möglicherweise auch Beliebtheit oder Bewertungen von Rezepten, um qualitativ hochwertige Vorschläge zu priorisieren.

5. Rückgabe von Daten an das Frontend:

Die REST-API sendet die verarbeiteten Suchergebnisse im JSON-Format an das Next.js-Frontend zurück. Das Frontend rendert dann dynamisch die empfohlenen Rezepte und präsentiert sie dem Benutzer auf der Benutzeroberfläche.

6. Benutzerverwaltung:

Bei Aktionen wie dem merken von Rezepten erfolgt eine Authentifizierung des Benutzers über die REST-API. Die Benutzerverwaltung wird genutzt, um sicherzustellen, dass nur autorisierte Benutzer auf bestimmte Funktionen zugreifen können.

Verteilungssicht

Die Verteilungssicht der Rezept-App beleuchtet die Struktur der verteilten Komponenten und wie sie miteinander kommunizieren.

1. Frontend-Verteilung (Next.js):

Das Frontend der Rezept-App, entwickelt mit Next.js, wird auf den Endgeräten der Benutzer ausgeführt. Dies ermöglicht eine leichtgewichtige und responsive Benutzeroberfläche. Die statischen Ressourcen wie HTML, CSS und JavaScript-Dateien werden von einem Content Delivery Network (CDN) bereitgestellt, um eine schnelle Auslieferung und optimale Performance sicherzustellen.

2. REST-API-Server (Rust):

Die REST-API, geschrieben in Rust, bildet das Backend der Anwendung. Dieser Server ist zuständig für die Verarbeitung von Benutzeranfragen, den Zugriff auf die Datenbank und die Bereitstellung von Rezeptinformationen. Der Server ist in einer Containerumgebung gehostet und kann je nach Bedarf horizontal skaliert werden, um eine erhöhte Last zu bewältigen.

3. Datenbank (MySQL):

Die MySQL-Datenbank ist das persistente Datenbanksystem, das die Rezeptdaten speichert. Diese Datenbank wird auf einem dedizierten Server oder einer Cloud-basierten Infrastruktur gehostet. Die Datenbank kann ebenfalls skaliert werden, um mit einer steigenden Menge an Rezeptdaten umzugehen, und ihre Schnittstellen sind nur für den Zugriff von autorisierten Servern geöffnet.

4. Kommunikation zwischen Frontend und Backend:

Die Kommunikation zwischen dem Frontend und der REST-API erfolgt über HTTP-Requests und -Responses. Diese werden durch sichere Protokolle geschützt, um die Integrität und Vertraulichkeit der übertragenen Daten zu gewährleisten. Die REST-API stellt eine einheitliche Schnittstelle bereit, um die Interaktion zwischen Frontend und Backend zu erleichtern.

5. Authentifizierung und Autorisierung:

Der REST-API-Server ist für die Benutzerauthentifizierung und -autorisation verantwortlich. Authentifizierte Benutzer erhalten Zugriff auf zusätzliche Funktionen wie das merken von Rezepte.

Querschnittliche Konzepte

Benutzererfahrung

Responsives Design: Responsives Design ist ein Ansatz, bei dem Webseiten so gestaltet werden, dass sie auf verschiedenen Geräten und Bildschirmgrößen optimal dargestellt werden. Dies wird durch flexible Layouts erreicht, die mithilfe von Grid-Systemen wie CSS Grid oder Flexbox dynamisch an die Bildschirmgröße angepasst werden. Bilder werden skaliert, indem CSS-Techniken wie max-width: 100% verwendet werden, sodass sie sich proportional zum Container anpassen. Der Einsatz von Media Queries ermöglicht spezifische Anpassungen des Designs basierend auf der Bildschirmbreite. Diese Maßnahmen erhöhen die Zugänglichkeit und

Benutzerfreundlichkeit der Anwendung, da Nutzer unabhängig vom verwendeten Gerät eine konsistente und angenehme Erfahrung haben.

CI/CD

Automatisierte Tests: Automatisierte Tests sind essenziell für die Qualitätssicherung in der Softwareentwicklung, da sie Fehler frühzeitig erkennen und die Zuverlässigkeit der Software erhöhen. Unit-Tests überprüfen einzelne Komponenten isoliert, während Integrationstests die Zusammenarbeit verschiedener Module testen. End-to-End-Tests validieren die gesamte Anwendung vom Anfang bis zum Ende. Diese Tests werden im Rahmen von Continuous Integration/Continuous Deployment (CI/CD) mithilfe von GitHub Actions realisiert.

Architekturentscheidungen

RESTful API

Die Rust API wird RESTful gestaltet, um eine einfache Kommunikation zwischen Frontend und Backend zu ermöglichen. Dies fördert auch die Interoperabilität und Skalierbarkeit.

Datenbank-Normalisierung

Die MySQL-Datenbank wird nach dem Prinzip der Datenbanknormalisierung gestaltet, um Datenredundanz zu minimieren und die Integrität der Daten zu gewährleisten.

Continuous Integration/Continuous Deployment (CI/CD)

Automatisierte CI/CD-Pipelines werden implementiert, um eine effiziente Bereitstellung von Änderungen sicherzustellen und die Entwicklungszyklen zu verkürzen.

Qualitätsanforderungen

1. Intuitive Benutzeroberfläche:

- *Beschreibung:* Die Benutzeroberfläche soll durch ein einheitliches Layout und klar verständliche Aktivitätsbuttons eine intuitive Nutzung ermöglichen.
- *Szenario:* Ein Benutzer kann ohne vorherige Anleitung die App bedienen und Rezepte anhand eingegebener Zutaten schnell finden.

2. Wartbarkeit des Quellcodes:

- *Beschreibung:* Der Quellcode soll so strukturiert sein, dass neue Funktionen leicht hinzugefügt und bestehende Funktionen problemlos gewartet werden können.
- *Szenario:* Ein Entwickler kann innerhalb kurzer Zeit neue Rezepte hinzufügen oder bestehende Rezeptvorschläge anpassen, ohne umfangreiche Änderungen an anderen Teilen des Codes vornehmen zu müssen.

3. Zuverlässige Suchergebnisse:

- *Beschreibung:* Die Suchfunktion soll zuverlässig genaue und passende Rezepte basierend auf den eingegebenen Zutaten liefern.
- *Szenario:* Bei der Eingabe von Zutaten in die Suchleiste werden dem Benutzer stets relevante Rezepte angezeigt, die die eingegebenen Zutaten enthalten.

4. Sicherheit:

- *Beschreibung:* Die Anwendung soll sicher vor unbefugtem Zugriff und Datenverlust geschützt sein.
- *Szenario:* Benutzer können sicher sein, dass ihre persönlichen Daten geschützt sind und nur autorisierte Benutzer Zugriff auf sensible Funktionen haben.

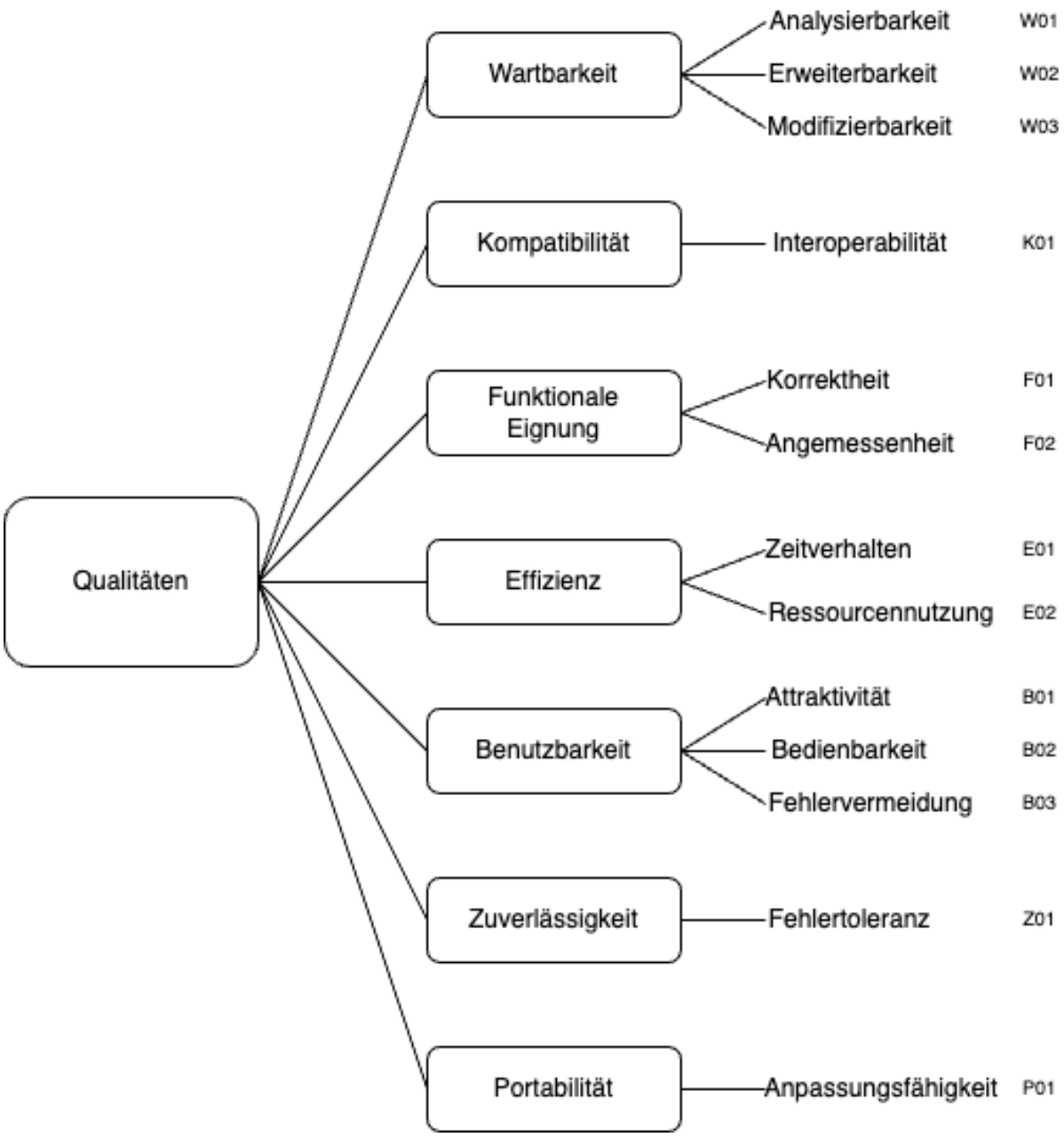
5. Performance:

- *Beschreibung:* Die Anwendung soll schnell und reaktionsfähig sein, unabhängig von der Anzahl der Benutzer und der Menge der verarbeiteten Daten.
- *Szenario:* Benutzer erleben kurze Ladezeiten, sowohl bei der Eingabe von Suchanfragen als auch bei der Anzeige von Rezeptvorschlägen.

6. Skalierbarkeit:

- *Beschreibung:* Die Anwendung soll skalierbar sein, um mit einer steigenden Anzahl von Benutzern und Daten umgehen zu können.
- *Szenario:* Die Architektur unterstützt horizontale Skalierung, um bei Bedarf zusätzliche Serverressourcen hinzuzufügen.

Qualitätsbaum



Qualitätsszenarien

Identifier	Beschreibung
W01	Strukturierter und gut dokumentierter Quellcode, der leicht verständlich ist.
W02	Modulbasierte Architektur, die zukünftige Erweiterungen ermöglicht.
W03	Einfaches Anpassen und Verbessern bestehender Funktionen.

K01	Integration mit bestehenden Frontends und Datenquellen, wie z.B. MySQL und externe APIs.
F01	Genauigkeit der Rezeptvorschläge basierend auf den eingegebenen Zutaten.
F02	Vorschläge relevanter und beliebter Rezepte.
E01	Schnelle Ladezeiten und Reaktionszeiten bei der Suche nach Rezepten.
E02	Optimale Nutzung der Serverressourcen für eine performante Anwendung.
U01	Ansprechendes und intuitives Design der Benutzeroberfläche.
U02	Einfache Navigation und Bedienung der App.
Z01	Eingabefehler werden durch klare Anweisungen und Eingabekontrollen minimiert.
Z02	Die App bleibt auch bei Fehlern funktionsfähig, z.B. durch Fallback-Mechanismen.
P01	Unterstützung verschiedener Geräte und Plattformen durch responsives Design.

Risiken und technische Schulden

Risiken

Leistungsriskien

Skalierungsprobleme: Bei einer wachsenden Benutzerbasis könnten Leistungsprobleme auftreten, wenn die Architektur nicht ausreichend skalierbar ist.

Entwicklungsrisiken

Abhängigkeiten von Schlüsselpersonen: Wenn kritisches Wissen bei wenigen Entwicklern konzentriert ist, besteht das Risiko von Wissensverlust bei deren Ausfall.

Technische Schulden

Codequalität

Veraltete oder ineffiziente Algorithmen: Teile des Codes könnten ineffiziente Algorithmen enthalten, die optimiert werden müssen.

Hardcodierte Werte: Nutzung von Hardcodierungen anstelle von konfigurierbaren Parametern, was Flexibilität und Wartbarkeit reduziert.

Technologische Abhängigkeiten

Veraltete Technologien: Abhängigkeit von veralteten Frameworks oder Bibliotheken kann die Wartung und Weiterentwicklung erschweren.