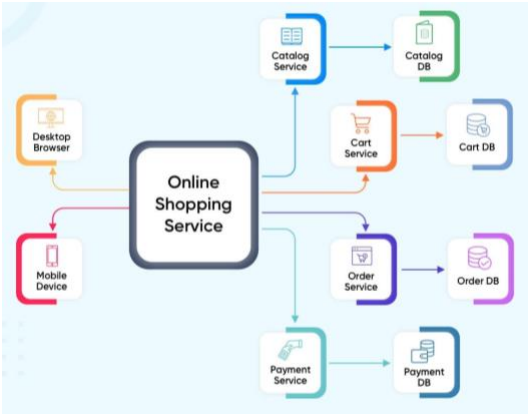
Praktische Umsetzungsarbeit



Beschreibung

Diese praktische Umsetzungsarbeit soll die behandelten Themen vertiefen. Die einzelnen in einer Microservice Architektur typischen Tools und Services sollen dabei zum Einsatz kommen.

Inhaltsverzeichnis

1 Praktische Umsetzungsarbeit

1.1 Einleitung

Die Aufgabe besteht darin, eine einfache Shop-Anwendung mit Microservices umzusetzen.
Dabei wenden Sie das Gelernte an und setzen allenfalls auch neue Technologien und Systemkomponenten ein.

Ziel:

In dieser bewerteten Aufgabe wollen wir die Aspekte an einer kleinen, exemplarischen Business Anwendung betrachten. Sie entwickeln eine einfache Shop-Anwendung und nutzen dabei Microservices-Architekturmuster mit verschiedenen Services und Tools.

Dabei sind einige Betrachtungen direkt umzusetzen und andere nur theoretisch zu untersuchen. Das heisst, die Implementation ist nicht vollständig bzw. fertig zu programmieren. Die Services sind aber so weit implementiert, dass sie die Idee der kompletten Anwendung zeigen. Die wichtigsten Elemente sind implementiert.

Die Anwendung soll folgende Komponenten beinhalten:

- Ein Gateway
- Eureka-Service-Discovery
- Messaging mit Kafka
- Cirquit-Breaker-Pattern (Fehlertoleranzmechanismen)
- Weitere individuelle Services (wie z.B. Produktkatalog, Bestellverwaltung, Bewertungsservice, etc.)

Voraussetzungen:

- Grundkenntnisse in Java und Spring Boot
- Grundverständnis von RESTFul Web Services
- Installierte Entwicklungsumgebung (z.B. IntelliJ IDEA)
- Docker und Docker Compose
- Erfahrung im Umgang mit Microservices Komponenten

Form:

- Partnerarbeit 2-3 Lernende

Planung mit Abgaben:

- Mittwoch, 02.4.2025, Start Projekt
- Mittwoch, 09.4.2025, Projektdefinition, Einrichtung des Repository
- Mittwoch, 16.4.2025, Architekturskizze grafisch, Repo-Push (main), erste Abgabe
- Mittwoch, 07.5.2025, Arbeiten am Projekt
- Mittwoch, 14.5.2025, Repo-Push (main), Reflexion Abgabe
- Mittwoch, 21.5.2025, Repo-Push (main), Update Reflexion, Endabgabe

Bewertung (siehe weiter unten):

- Einsatz von Microservices Komponenten
- Clean Code Regeln
- Architekturskizze
- Dokumentation der Anwendung bzw. der Services

1.2 Hauptaufgabe

1.2.1 Einleitung - Entwicklung von Microservices für die Shop-Anwendung

In diesem Schritt fokussieren Sie sich auf die Entwicklung spezifischer Microservices, welche für die Funktionalität unseres Shops essenziell sind. Jeder Microservice wird eine klar definierte Aufgabe erhalten und über REST APIs mit den anderen Services kommunizieren.

Die Anwendung soll aus den folgenden Elementen bestehen:

- Einfaches Frontend (React/Angular/Nextjs)
- Warenkorb Microservice
- drei bis vier Microservices

→ Die Anwendung soll in sich geschlossen «Sinn» ergeben und muss nicht vollständig ausprogrammiert sein.

Sie wählen zum Frontend und dem Warenkorb weitere Services, die Sie gerne umsetzen wollen und für Ihre Architektur «Sinn» ergeben. Die Anwendung hat exemplarischen Charakter und dient dem Kompetenznachweis.

Beispiele:
Orientieren Sie sich bspw. einen Bestelldurchgang und spielen Sie diesen durch.

1	vom Frontend über/mit Warenkorb und einem Produktekatalog.
2	vom Frontend über/mit Warenkorb, Bestellservice und Lagerverwaltung.

1.2.2 Vorgeschlagene Microservices

Produktkatalog-Service:
Zweck: Verwaltet Informationen zu Produkten, wie z.B. Namen, Beschreibungen, Preise und Lagerbestände.
Technologien: Spring Boot für den Service, Spring Data JPA für den Datenbankzugriff, und eine H2- oder MySQL-Datenbank.
Endpoints: GET für Produktliste, GET für einzelne Produkte, POST für das Hinzufügen neuer Produkte, PUT zum Aktualisieren und DELETE zum Entfernen von Produkten.

Bestellung-Service:
Zweck: Verarbeitet Bestellungen von Kunden, inklusive Bestelldetails und Zahlungsinformationen.
Technologien: Spring Boot, Spring Data JPA, und eine relationale Datenbank.
Endpoints: POST zum Erstellen einer neuen Bestellung, GET zum Abrufen von Bestelldetails, PUT zur Aktualisierung des Bestellstatus.

Kundenverwaltungs-Service:
Zweck: Verwaltet Kundeninformationen wie Namen, Adressen und Kontaktinformationen.
Technologien: Spring Boot, Spring Data JPA, und eine relationale Datenbank.
Endpoints: GET zum Abrufen von Kundeninformationen, POST zum Hinzufügen neuer Kunden, PUT zum Aktualisieren von Kundendaten, Verwaltung von Benutzerkonten.
Evtl. Registrierung, Anmeldung, Profilverwaltung und Sicherheitsfunktionen wie Passwortzurücksetzung.

Warenkorb-Service:
Zweck: Ermöglicht Kunden das Hinzufügen und Entfernen von Produkten in einem virtuellen Warenkorb.
Technologien: Spring Boot und eine NoSQL-Datenbank wie Redis für schnelle Datenzugriffe.
Endpoints: POST zum Hinzufügen von Produkten in den Warenkorb, DELETE zum Entfernen von Produkten, GET zur Anzeige des aktuellen Warenkorbs.

Zahlungs-Service (Mockup):
Zweck: Verarbeitet Zahlungsvorgänge, einschließlich Kreditkarten- und PayPal-Transaktionen.
Technologien: Spring Boot und Integration von Zahlungsdienstleistern über deren APIs.
Endpoints: POST zum Durchführen einer Zahlung, GET zum Abrufen des Zahlungsstatus.

Bewertungs- und Rezensions-Service:
Zweck: Ermöglicht Kunden das Hinterlassen von Bewertungen und Rezensionen für Produkte.
Technologien: Spring Boot, Spring Data JPA, und eine relationale Datenbank.
Endpoints: POST zum Hinzufügen einer Bewertung, GET zum Abrufen von Bewertungen.
Implementierungshinweise:

Versand- und Lieferdienst:
Zweck: Dieser Service könnte sich um die Logistik der Bestellung kümmern, wie z.B. Versandoptionen, Tracking-Informationen und Lieferzeitberechnung.

1.3 Aufgabe 1 – Projektdefinition

Wählen Sie eine Architektur mit den ausgewählten Microservices.

→ Erstellen Sie eine grafische Architekturskizze. Informieren Sie sich, wie eine solche Architekturskizze aufgebaut werden kann.

→ Schreiben Sie eine kurze Beschreiben Ihrer Idee und den angedachten Workflow.

1.4 Aufgabe 2 – Reflexion und Git-Repository

Einblick in die laufenden Arbeiten

Erstellen Sie für das Projekt ein Git-Repository mit allen Services und laden Sie die Lehrperson zu Ihrem Repository ein (BBWRL).

Dokumentation

Die Dokumentation der Arbeit erfolgt in einer Markdown-Datei (.md). Die Datei ist im Git-Repository zu speichern. Die Umsetzungsarbeiten am Projekt, sowie die Dokumentation sind wöchentlich kontinuierlich zu dokumentieren. Dieser formale Parameter wird ebenfalls bewertet.

1.5 Aufgabe 3 – Abgabe

Die Abgabe des Projektes erfolgt regelmässig im Git-Repository/Repositories sowie mit der entsprechenden Dokumentationsdatei(en).

2 Bewertung der Arbeit

2.1 Abgaben

Folgende Artefakte gehören zur Abgabe:

- Eine PDF-Datei mit der Projektdefinition, Architekturskizze, Überlegungen zur Architektur, Microservices und ihre Funktionen, sowie eine Reflexion und den Link zum Git-Repo.
  Namen der Autoren der Gruppe.
- Ein Git-Repository mit allen Microservices (inkl. Frontend)

2.2 Grobe Einschätzung

Hat das Projekt und die Anwendung sowie die Umsetzung und Betrachtungspunkte eine hohe Professionalität und Seriosität?
Ist das Projekt und die Planung zukunftsgerichtet?
Stehen der erbrachte Aufwand und der daraus gewonnene Ertrag im Einklang?

→ im Bereich minimal: Note 3.5 – 4.5
→ im Bereich gut bis sehr gut: Note 4.5 – 5.5
→ im Bereich top: Note 5.0 - 6

2.3 Anforderung an die Lösung

Der Fokus liegt auf:

- Der Architekturskizze und den dahinter stehenden Überlegungen (Argumentation).
- Ein Ausschnitt wird ausgewählt und programmiert sowie erläutert.

Nr.	Anforderung
1	Jeder Service soll bei Bedarf eigene Datenbank besitzen, um die Dienstunabhängigkeit zu gewährleisten. ~~Eines der Microservices muss auf eine Datenbank zugreifen, die anderen können mit Mockups arbeiten~~. Argumentation, welcher Service eine eigene Datenbank hat, welche Datenbank gemeinsam genutzt wird.
~~2~~	~~Nutzen Sie Docker, um ihre Datenbank(en) zu starten.~~
3	Stellen Sie sicher, dass die Services über REST-APIs kommunizieren können. ~~Nutzen Sie Spring REST Controllers für die Implementierung~~.
4	Berücksichtigen Sie Sicherheitsaspekte, insbesondere bei der Übertragung von sensiblen Kundendaten und Zahlungsinformationen.
5	~~Implementieren~~ Machen Sie sich Gedanken über geeignete Fehlerbehandlungsmechanismen und Tools, um die Robustheit der Anwendung zu gewährleisten.
6	Jeder Service hat eine genau definierte Aufgabe und diese wurde dokumentiert. Die Kombination der Services macht für die gewählte App Sinn.
7	Grafische Architekturskizze mit Überlegungen.
9	~~Services nach Clean-Code Regeln implementiert.~~
9	~~API-Dokumentation und -Design z.B. mit Swagger.~~
10	~~Formale Ausführung der Arbeiten (Gruppe)~~