Projektdefinition

Services:

Online Shopping Service

Der Onlineshop ist ein Gateway, welcher verschiedene andere kleine Service handelt. Alle Service laufen über diesen Service und werden dann weitergeleitet an das Frontend.

Cart Service

Dieser Service speichert die Produkte, welche man kaufen möchte. Auf Deutsch Warenkorb. Die Produkte werden aufgelistet. Angezeigt werden die Produkte im Warenkorb mit Preis, Anzahl Produkt, Beschreibung und Namen des Produkts.

Catalog Service

Der Katalog ist der Ort, wo man die Übersicht über die Produkte hat, welche es auf diesem Shop gibt. Waren die einem gefallen, können zum Warenkorb Service hinzugefügt werden.

Order Service

Order Service werden die im Warenkorb gespeicherten Produkt beim Anbieter bestellt und dem Kunden ein bestätigunsmail zugesendet, dass das Produkt verpackt wird und versendet wird. Wenn Payment Service eine Zahlung bestätigen konnte, wird der Auftrag ausgeführt (Service wird ausgeführt)

Payment Service

Das ist der Service, welcher gebraucht wird, um die Waren, welche man kaufen möchte, zahlen zu können. Es gibt verschiedene Zahlungsmöglichkeiten. PayPal, Twint, Kreditkarte oder Rechnung. Wenn die Bezahlung funktioniert hat, wird eine Message an den Order Service gesendet, damit der weiss, dass die Bestellung ausgeführt werden kann.

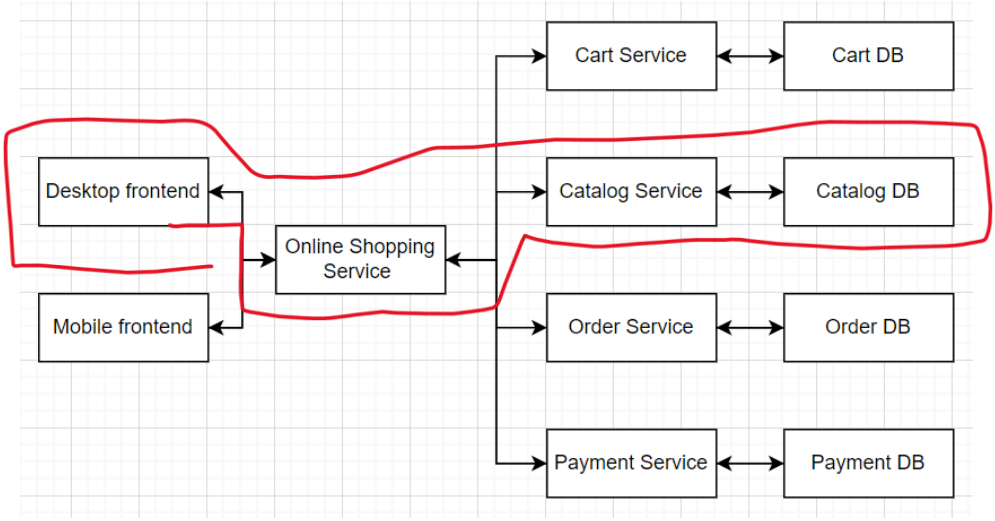
Anwendungsfälle:

- **Benutzer bestellt ein Produkt auf dem Shop und bezahlt mit der Kreditkarte**
    1. Services: Catalog, Cart, Order, Payment.
    2. Produkt im Catalog Service auswählen.
    3. Produkt in den Warenkorb legen (Cart Service, Cart DB).
    4. Bestellung auslösen (Order Service, Order DB).
    5. Zahlung mit Kreditkarte durchführen (Payment Service, Payment DB).
- **Benutzer wählt das letzte Produkt und speichert es im Warenkorb**
    1. Services: Catalog, Cart.
    2. Produktdetails über Catalog Service abrufen (Catalog DB).

3. Produkt in den Warenkorb speichern (Cart Service, Cart DB).

- **Benutzer Liket ein Produkt**
    1. Services: Catalog.
    2. Produkt im Catalog Service anzeigen (Catalog DB).
    3. Like wird verarbeitet (möglicherweise durch den Catalog Service oder separaten Mechanismus).

Architektur Design



Beschreibung: Wir haben unsere Architektur so aufgebaut, dass alles über den Service Online Shopping Service läuft. Die Services darunter sind voneinander getrennt, so dass jeder Service seine spezielle eigene Aufgabe hat und sie sich nicht in die Quere kommen.

DB-Nutzung

Warum hat jeder Service seine eigene Datenbank? Ganz einfach: Es macht die Services unabhängig, sie können frei entwickelt, skaliert und optimiert werden. Jeder nutzt die passende Datenbank-Technologie, was Performance steigert und Engpässe vermeidet. Außerdem bleibt alles sicherer, da nur der Service selbst auf seine Daten zugreift. Klar, es gibt etwas mehr Verwaltungsaufwand, aber die Flexibilität und Unabhängigkeit machen das locker wett. Es ist also flexibler und das ist einfach für ein Projekt, wo noch nicht so gross ist. Wenn das Projekt sehr gross wird. Dann kann es sein das es schnell viele DB werden können und das würde Leistung stehlen. Also für so kleine Projekte kann man die Aufteilung gut machen. Die DB für den Service Catalog wurde bereits eingebaut. Der Rest muss noch eingebaut werden

Architektur

Komponenten:

1. Frontend (React):

    - Zeigt Produkte an.

    - Ermöglicht das Hinzufügen von Produkten in den Warenkorb.

    - Kommuniziert über REST APIs mit den Backend-Services.

2. Backend (Spring Boot Microservices):

    - Produktkatalog-Service: Verwaltet Produktinformationen.

    - Warenkorb-Service: Speichert und verwaltet Artikel im Warenkorb.

    - Bestellungs-Service: Handhabt Bestellungen.

    - Zahlungs-Service (Mockup): Simuliert Zahlungsabwicklungen.

3. Datenbanken:

    - Produktkatalog-Service: MySQL

    - Warenkorb-Service: MYSQL

    - Bestellungs - Service: MySQL

    Man könnte auch alles in einer DB machen. Dann kann man einfacher übergreifende Inner joins zb machen und hat alles zusammen und dadurch weniger Schnittstellen zu DBs. Das wäre auch eine Überlegung, welche wir uns gemacht haben. Schlussendlich aber entschieden wir uns für die Trennung der DBs.

4. Docker: Containerisierung von Services und Datenbanken.

Endpunkte der Microservices

Produktkatalog-Service:

- GET /products: Alle Produkte abrufen.
- POST /products: Neues Produkt hinzufügen.

Warenkorb-Service:

- GET /cart: Warenkorbinhalte anzeigen.
- POST /cart: Produkt zum Warenkorb hinzufügen.

Bestellungs-Service:

- POST /orders: Bestellung erstellen.
- GET /orders/{orderId}: Bestelldetails anzeigen.

Zahlungs-Service:

- POST /payments: Zahlung simulieren.

Schritte zur Ausführung

1. Docker:
   - Starten Sie die Anwendung mit docker-compose up --build.
2. Frontend:
   - React läuft auf http://localhost:3000.
3. Backend:
   - Produkt-Service: http://localhost:8081.
   - Warenkorb-Service: http://localhost:8082.

Fehlerhandling

Fehler Möglichkeit	Handling
Service stürzt ab oder hat ein Problem	Message an User senden, dass es Probleme gegeben hat und man es nochmals probieren soll oder später probieren soll

Connection zu der DB funktioniert nicht	Message machen, dass die Produkte oder Warenkorb zurzeit nicht aufgerufen werden können.
Connection zu der DB bricht ab, wärend ein neuer Eintrag gemacht wird oder geupdated wird.	Message machen:» Update zum Eintrag konnte nicht gespeichert werden oder Eintrag konnte nicht gespeichert werden»
Backend stürzt ab.	Kontaktierung von Developer oder Sicherung, damit Projekt neu gestartet wird.

Sicherheitsaspekte

Sensible Sicherheitsaspekte und Schutzmaßnahmen:

1. **Zahlungsdaten**
    - Verschlüsselung, Tokenisierung
2. **Passwörter und Authentifizierung**
    - Hashing, Multi-Faktor-Authentifizierung, starke Passwortrichtlinien
3. **Persönliche Daten**
    - Datenminimierung, Anonymisierung
4. **Gesundheitsdaten**
    - End-to-End-Verschlüsselung, Zugriffskontrollen
5. **Systemzugriffe und Berechtigungen**
    - Rollenbasierte Zugriffskontrolle, Protokollierung, regelmäßige Rechteprüfung
6. **Kommunikationsinhalte**
    - End-to-End-Verschlüsselung, Anti-Phishing-Maßnahmen, digitale Signaturen
7. **Technische Sicherheitsaspekte**
    - Regelmäßige Updates, Penetrationstests

Rest-API

Alle Services laufen über die Rest-APIs. Welch die noch nicht angebunden sind:

- Online Shopping Service
- Catalog Service
- Order Service
- Payment Service

Reflexion

Fabian, ich und anfangs noch Jonathan planten zusammen das Projekt und Fabian und ich kümmerten uns das um die Umsetzung und den Rest des Auftrages. Die Arbeit war anfangs etwas verwirrend. Genauer gesagt, kamen wir nicht draus, was wir machen müssen. Aber mit der Zeit wurde das dann besser. Wir konnten uns für ein Service entscheiden und den Rest der Service planten wir einfach noch. Ich würde mit Fabian Bätscher sicher nochmals ein Projekt machen, aber dann mit genauer Rollen Verteilung. Sonst war das Projekt sehr lehrreich und spannend zu machen.

Github Link: https://github.com/Janbei16/Microservices_Test