Dokumentation Mikroservices

Abgabe: 22.05.2024

Azhaar Mohamed, Jelena Speer, Thierry Kellenberger

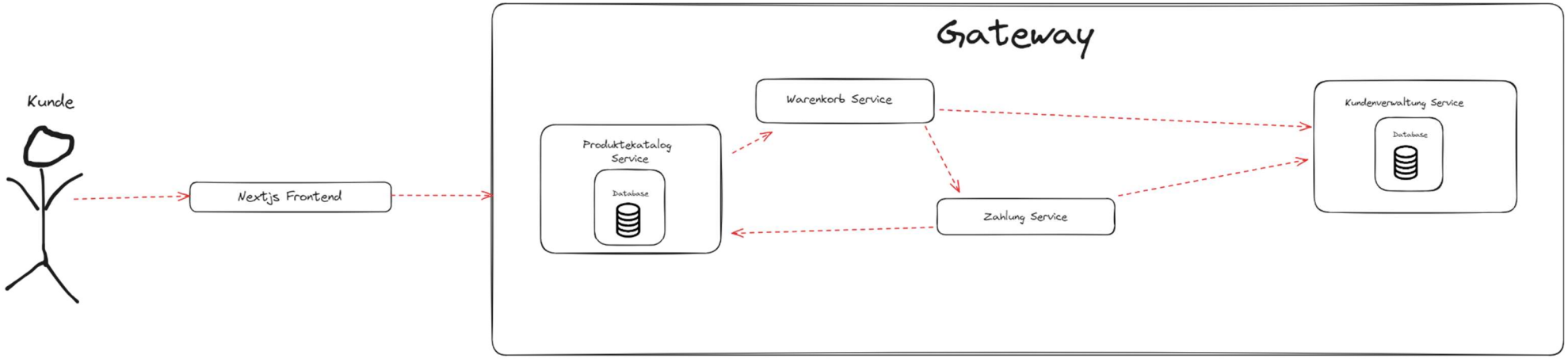
Inhaltsverzeichnis

1 Projektdefinition

In diesem Projekt geht es darum sich Gedanken über Mikroservices zu machen. Für diesen Auftrag haben wir uns einen fiktionalen Onlineshop ausgedacht der Socken verkauft. Dieser verfügt im Backend über mehrere Services. In dieser Dokumentation geht es primär um die Planung des Onlineshops und weniger um die Implementierung. Trotzdem haben wir unser GitHub Repo angehängt, damit daraus mehr Informationen gezogen werden können. In unserem Repo haben wir zu dem einzelnen Service in einem kleineren Umfang implementiert, damit man ein Gefühl dafür bekommt wie es sein soll.

Link zum Repo: https://github.com/Jelenal1/sock_shop_M183_Microservices

2 Architekturskizze



Der Produktekatalog Service hat eine Datenbank, die dafür zuständig ist, den Lagerbestand zu speichern. Die Bestände ändern sich stetig, weshalb hier eine In Memory Datenbank von uns gewählt werden würde.

Der Kundenverwaltung Service hat eine eigne Datenbank die getrennt vom Produktekatalog Service ist. Diese speichert die Kundendaten in einer SQL-Datenbank ab. Sensible Daten werden nach unseren Sicherheitsanforderungen verschlüsselt und gehasht gespeichert.

Wir haben uns für diese Services entschieden, weil das die benötigten Grundbausteine für jeden Online-Shop sind. Jeder Shop braucht die Möglichkeit eine Zahlung durchzuführen, dementsprechend braucht es auch einen Warenkorb. Damit die Kunden auch wissen welche Produkte gerade Verfügbar sind ist ein Produktekatalog auch essenziell und ein Kundeverwaltung Service ermöglicht es unseren Benutzer ihre Daten zu speichern.. Ausserdem ermöglichen uns diese Mikroservices unseren Shop flexibel zu skalieren.

3 Erklärung zur Architektur

3.1.1 Warenkorb Service

Zweck/Aufgabe:

- 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:

- GET /basket alle Produkte zurückgeben
- GET /basket/{id} ein Produkt anhand id zurückgeben
- POST /basket ein Produkt zum Warenkorb hinzufügen
- DELETE /basket Warenkorb leeren
- DELETE /basket/{id} ein bestimmtes Produkt entfernen
- PUT / PATCH /basket/{id} ein Produkt aktualisieren (z.B Anzahl)
- POST zum Hinzufügen von Produkten in den Warenkorb, DELETE zum Entfernen von Produkten, GET zur Anzeige des aktuellen Warenkorbs.

3.1.2 Produktkatalog Service

Zweck/Aufgabe:

- Verwaltung des Lagerbestands, verwaltet Informationen, wie Namen, Preise, Lagerbestände und Stripe Codes.

Technologien:

- Spring Boot für den Service, Spring Data JPA für den Datenbankzugriff, und eine H2 Database

Endpoints:

- /products GET all products
- /products/{id} GET product by id
- /products POST new product
- /products/{id} PUT update a product
- /products/{id} DELETE delete a product by id

3.1.3 Zahlungsservice

Zweck/Aufgabe:

- Abwicklung der Zahlungen über Stripe

Technologien:

- Spring Boot für den Service, Stripe für die effektive Zahlungsabwicklung

Endpoints:

- /checkout POST den aktuellen Stand des Warenkorbs bezahlen.

3.1.4 Kundenverwaltungsservice:

Zweck/Aufgabe:

- Verwaltet Kundeninformationen wie Namen, Adressen und Kontaktinformationen.

Technologien:

- Spring Boot, Spring Data JPA, und eine relationale Datenbank.

Endpoints:

- GET /account Accountdaten abrufen
- GET /account{id} Abrufen von Informationen pro Kunde
- POST /account Hinzufügen eines neuen Kunden
- PUT /account/{id} Aktualisieren von Kundendaten eines Kunden.
- POST /account/register als Neukunde registrieren
- POST /account/login Einloggen

3.2 Sicherheitsaspekte

Für unsere Mikroservices haben wir ein umfassendes Sicherheitskonzept entwickelt. Um maximale Sicherheit zu gewährleisten, übergeben wir viele Sicherheitsaspekte an Drittanbieter, da diese meist sicherere Implementierungen anbieten.

1. Authentifizierung und Autorisierung

- Wir benutzen Dienste wie OAuth 2.0 und OpenID Connect, um sicherzustellen, dass nur autorisierte Benutzer authentifiziert werden. Diese Standards bieten eine sichere und zuverlässige Authentifizierung.

2. Datenverschlüsselung

- Sensible Daten wie Passwörter und Tokens müssen verschlüsselt übertragen werden. Dasselbe gilt für sensible Daten, die in der Datenbank gespeichert werden.

3. API-Endpunkte

- Um Missbrauch zu verhindern, setzen wir Rate Limiting ein. Dies schützt vor Brute-Force- und DDoS-Angriffen.

4. Monitoring

- Mit der Zeit wird unsere Applikation wachsen, was die Übersicht erschwert. Daher benötigen wir Monitoring- und Logging-Systeme, um unerwünschte Aktivitäten und Probleme schnell zu erkennen.

5. Datensicherung und Wiederherstellung

- Unsere Applikation verwendet eine Datenbank, daher sind regelmässige Backups im Falle eines Datenverlusts wichtig. Ein Recovery-Plan stellt sicher, dass wir die Applikation auch bei Ausfällen weiterhin betreiben können.

6. Sicherheit der Anbieter

- Wir vertrauen stark auf die Dienstleistungen von Drittanbietern. Bei der Auswahl achten wir darauf, dass diese keinen negativen Sicherheitsvorfall in der Vergangenheit hatten und ihre Dienstleistungen regelmässig aktualisieren. Sie müssen stets den neuesten Sicherheitsstandards entsprechen und preislich in unser Budget passen.

Grundsätzlich versuchen wir, durch die Zusammenarbeit mit erfahrenen Drittanbietern und die Implementierung bewährter Sicherheitsmassnahmen, ein hohes Sicherheitsniveau für unsere Mikroservices zu gewährleisten.

Für die Zahlungsabwicklung benutzen wir den Anbieter Stripe.

3.3 Error Handling

Jede Applikation benötigt effektive Fehlerbehandlungsmechanismen, insbesondere bei Mikroservices. Es ist wichtig, frühzeitig über Probleme informiert zu werden, um die Ursache schnell zu erkennen und zu beheben. Hier sind einige wichtige Massnahmen:

1. Fehlerbehandlung

- **Fehlerseiten im Frontend:** Unerwartete Fehler im Frontend sollten durch benutzerfreundliche Fehlerseiten abgefangen werden. Diese Seiten sollten dem Benutzer verständliche Informationen anzeigen und gleichzeitig den Fehler protokollieren.
- **Fehlerprotokollierung:** Alle aufgetretenen Fehler sollten automatisch protokolliert werden, um eine spätere Analyse und Behebung zu ermöglichen. Zum Beispiel in einer Log Datei

2. Logging und Monitoring

- **Logging:** Logs sind entscheidend, um Probleme zu diagnostizieren und zu verstehen, welche Fehler in der Applikation auftreten. Logdateien bieten detaillierte Informationen, die bei der Fehlersuche helfen.
- **Monitoring:** Überwachungstools sind notwendig, um den Zustand der verschiedenen Dienste zu überwachen. Sie zeigen an, welche Services laufen und ob es Auffälligkeiten gibt.

3. Automatisierte Tests

- **Unit-Tests:** Da sich Mikroservices ständig weiterentwickeln, sind unabhängige und effektive Unit-Tests wichtig. Diese Tests stellen sicher, dass jeder Service seine Aufgaben korrekt erfüllt.
- **Integrationstests:** Zusätzlich zu Unit-Tests sind Integrationstests wichtig, um sicherzustellen, dass die verschiedenen Mikroservices ohne Probleme zusammenarbeiten.

4. Skalierung und Backups

- **Automatische Skalierung:** Um Überlastungen zu vermeiden, sollte die Applikation automatisch skaliert werden können. Dies stellt sicher, dass die Anwendung auch bei hoher Last performant bleibt und es nicht zu Problemen führt
- **Automatische Backups:** Regelmässige automatische Backups sind essenziell, um im Falle eines Datenverlusts schnell auf die aktuellen Daten zugreifen zu können.

5. Kontinuierliche Verbesserung

- **Regelmässige Updates:** Keine Software ist perfekt. Es ist wichtig, kontinuierlich an der Applikation zu arbeiten und sie zu verbessern. Durch regelmässige Updates stellen wir sicher, dass die Applikation immer auf dem neuesten Stand ist und Sicherheitslücken geschlossen werden.

Durch diese Massnahmen wird die Robustheit und Zuverlässigkeit der Applikation deutlich erhöht, wodurch eine stabile und sichere Nutzung gewährleistet wird.

4 Reflexion

Das Projekt war schnell aufgestellt und die Aufgabenverteilung ging schnell. Als alle wussten, was sie machen sollten, legten sie los und waren auch schnell mit ihrer Aufgabe fertig. Die Dauer des Projektes war sehr kurz, weshalb wir nur wenige Dinge implementieren konnten. Auch trotz der kurzen Dauer gelang es uns, alles umzusetzen, was wir geplant hatten. In einem realen Projekt hätten wir viel mehr Zeit als nur 3 Wochen und könnten alles implementieren.