Hochschule Kaiserslautern
University of Applied Sciences

Andreas Eitel
University of Applied Sciences
Kaiserslautern
andreas.eitel@hs-kl.de

SOFTWARE-ARCHITEKTUR:

AUFGABENPAKET 4

27. Februar 2022

Vorbemerkung

Das Hauptthema dieses Arbeitspakets ist „Verteilte Systeme“. Diese Übung geht etwas tiefer auf das Thema Microservices ein. Nachdem in der letzten Übung die theoretischen Grundlagen erarbeitet wurden, folgen in dieser Übung weitere theoretische und praktische Übungen mit einem Buchlager Backend auf Basis von Spring-Boot.

Theoretischer Teil

1. **Visualisieren Sie eine Microservice-Architektur**
   Im praktischen Teil dieser Übung werden Sie unser monolithisches Buchlager-Beispiel so bearbeiten, dass verschiedene Microservices entstehen. In dieser Aufgabe sollen Sie zur Übersicht zuerst einen Architekturplan erstellen, wie unser vereinfachtes Microservice-System aussehen wird.

   Gehen Sie davon aus, dass folgende Services vorhanden sind:

   - Der **UI-Service** liefert die Oberfläche der Anwendung aus.
   - Der **SearchService** ist optimiert für die Durchführung von Suchen. Es ist eine Suche nach Autorenname, Buchtitel und ISBN möglich. Als Ergebnis kann er auch nur diese Daten zurück liefern.
   - Der **BookService** kennt alle Details zu einem Buch und bietet entsprechende Verwaltungsfunktionen (Hinzufügen, Bearbeiten, Löschen) an.
   - Im **MasterDataService** werden sämtliche Kundendaten und notwendige Managementfunktionen vorgehalten.
   - Der **CartService** stellt den virtuellen Warenkorb dar. In ihm wird eine der KundenID zugeordnete Session verwaltet, die die ausgewählten Artikel aufbewahrt und zum Schluss den Bezahlvorgang durchführt.

   Darüber hinaus, werden die folgenden Daten vom System verarbeitet:

- Die **Buchdaten** umfassen Titel, Verlagsname, Zusammenfassung, Autorennamen, ISBN (ID), Genre und die Anzahl der verfügbaren Exemplare.
- Die **Stammdaten** umfassen KundenID, Anrede, Vorname, Name, Straße, Postleitzahl, Ort, E-Mailadresse und letzte Bestellungen.
- Die **Warenkorbdaten** umfassen eine SessionID, eine KundenID, die ISBN und die dazugehörige Buchanzahl.

Weitere Anforderungen: Die Kommunikation erfolgt über REST-Schnittstellen und zur Erhöhung der Ausfallsicherheit sollen von jedem Service zwei Instanzen existieren, d.h. sobald ein Service ausfällt, soll die andere Instanz übernehmen.

Zur Visualisierung können Sie die Software draw.io (online erreichbar unter https://draw.io) nutzen. Diese steht unter https://www.diagrams.net auch zum Download bereit.

In den Unterlagen finden Sie außerdem ein kleines draw.io Template mit dem Sie starten können.

(a) (5 Punkte) Vervollständigen Sie das Template: Zeichen Sie mindestens 3 Komponenten ein, die zum Management und Nutzung der Microservices benötigt werden. Zeichnen Sie außerdem die fehlende Datenhaltung (Datenbanken) ein. Zeichnen Sie zuletzt die notwendigen Kommunikationsverbindungen zwischen den Komponenten ein.

(b) (1½ Punkte) Erklären Sie schriftlich die Funktionsweise der 3 Managementkomponenten (fassen Sie sich kurz).

(c) (1 Punkt) Beschreiben Sie kurz den Ablauf und die einzelnen Stationen einer Anfrage: Ein Benutzer befindet sich auf der Homepage und tippt einen Suchbegriff ein, der automatisch eine Suche nach einem Buch auslöst.

(d) (3 Punkte) In den Unterlagen finden Sie im draw.io Template außerdem eine Seite mit einem Datenmodell zu den genannten Services.

- Überlegen Sie, welche Daten der SearchService kennen muss und vervollständigen Sie das Datenmodell.
- Erklären Sie schriftlich (kurz), welcher der obigen Services, welche Daten aus dem Datenmodell kennt.
- Erklären Sie schriftlich am Beispiel des SearchService, wie bzw. mit welcher Strategie er seine Daten beziehen und „aktuell“ halten kann. Er soll gleichzeitig schnell antworten können (Caching sollte möglich sein), als auch Änderungen schnell reflektieren (falls sich beispielsweise der Buchtitel ändert). Denken Sie dabei an eine Technik, die Sie in Übung 2 genutzt haben.

Praktischer Teil

Nutzen Sie eclipse oder IntelliJ zum Bearbeiten der nachfolgenden Aufgaben. Laden Sie Ihre Lösung als ZIP-Datei hoch. Build-Artefakte sollten nicht hochgeladen werden.

2. **Refactoring des Buchlager-Backend**
Im Paket dieser Übung finden Sie ein Buchlager-Backend auf Basis von Spring-Boot

mit einer sehr einfachen UI (dient nur der Visualisierung – erreichbar unter http://localhost:8888/). Die REST-API erreichen Sie mit der URL http://localhost:8888/swagger-ui.html. Dieses Backend wurde als monolithische Anwendung gebaut und soll nun nach und nach zu mehreren Microservices umgebaut werden. Machen Sie sich mit dem Aufbau vertraut. Beachten Sie dabei speziell die REST-Schnittstellen. Abbildung 1 zeigt den grundlegenden Aufbau des Backends grafisch.

(a) (4 Punkte) Wir können in dieser Übung kein vollständiges Refactoring durchführen. Deshalb soll z. B. nur eine Datenbank für alle Dienste benutzt werden. Es soll außerdem auf Synchronisationsmechanismen und Aufteilung des Datenmodells verzichtet werden. Es werden für diese Übung nur 3 Services benötigt: SearchService, BookService, AddressAuthorCartPublisherService. Die notwendigen APIs erkennen Sie über den Namen. Tipp: Erzeugen Sie mehrere kleine Microservices aus dem monolithischen Backend, indem Sie das Backend kopieren und die REST-Schnittstellen entsprechend bearbeiten/entfernen. Die einzelnen Microservices sollen nach ihrer Aufteilung auch funktionieren.

(b) (2 Punkte) Sorgen Sie dafür, dass alle Microservices den oben genannten eindeutigen Namen (`spring.application.name`) haben (z. B. Such-Service: SearchService) und mit einem zufällig gewählten Port (`server.port`) laufen. Der Port soll von Spring-Boot automatisch beim Start gewählt werden.

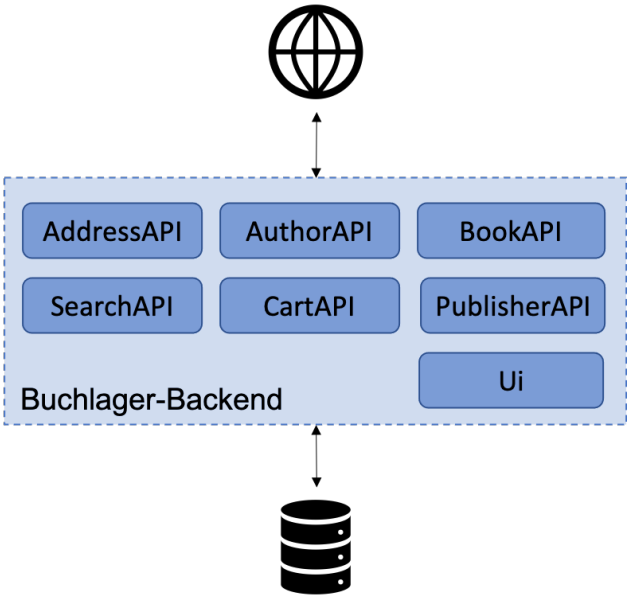

Abbildung 1: Grundsätzlicher Aufbau und Schnittstellen des monolithischen Buchlager-Backends (Web-Version).

3. **Umgang mit GIT üben**
Das Thema Continuous Integration (CI) spielt bei Microservices eine große Rolle. Die Grundlage dafür ist die Verwaltung des Quellcodes mit einem Versionskontrollsystem. In dieser Übung soll daher der Umgang mit GIT an einem einfachen Beispiel geübt werden. Zur Übung verwenden wir die GitLab-Instanz des Landes Rheinland-Pfalz (https://gitlab.rlp.net/).

Geben Sie mir Zugriff auf Ihr Repository, damit ich die Übung kontrollieren kann. Zudem erzeugen Sie als Dokumentation für die Einsendung der Übungsunterlagen einen Screenshot der Historie und der (ausgeführten) Build-Pipeline.

(a) (1 Punkt) Überführen Sie das Spring-Boot UI-Projekt (im Anhang) in ein GIT Repository.

(b) (1 Punkt) Sorgen Sie anschließend dafür, dass der Build-Order (target) und typische Java Build-Dateien ignoriert werden. Erstellen und committen Sie die entsprechende Datei mit den Einstellungen.

(c) (3 Punkte) Als letzten Schritt realisieren Sie eine erste Stufe von Continuous Integration. Wir nutzen dazu die integrierte Build-Umgebung in GitLab (einen GitLab-Runner). Erzeugen Sie dazu eine passende CI-Datei (`.gitlab-ci.yml`), die 2 Stages hat (die nacheinander ausgeführt werden):

- build (hier soll `mvn build` ausgeführt werden)
- package (hier soll `mvn package` ausgeführt werden)

Das in `mvn package` erzeugte Artefakt soll für die Dauer von 2 Wochen zum Download bereitstehen und danach wieder gelöscht werden. Nutzen Sie als Basis-Image `maven:3-amazoncorretto-11`.

Der Aufbau der `.gitlab-ci.yml`-Datei war nicht Bestandteil der Vorlesung. Als Ausgangspunkt für Ihre Recherchen können Sie die Befehlsreferenzseite von Gitlab benutzen: https://docs.gitlab.com/ee/ci/yaml/index.html. Diese Seite führt Sie zu einem Quick Start Guide, Beispielen und der Beschreibung der verfügbaren Optionen. Ein konkretes CI-Beispiel für Maven finden Sie hier: https://gitlab.com/gitlab-examples/maven/simple-maven-example.

Tipps zur Implementierung

Bevor Sie das Buchlager-Backend starten können, müssen Sie sicherstellen, dass alle Abhängigkeiten vorhanden sind. Dies betrifft in diesem Fall die Bibliothek mit dem Domänenmodell und dem Repository-Layer. Diese Abhängigkeit können Sie mit `mvn install` bauen und dadurch in Ihrem Maven-Cache ablegen und in anderen Projekten nutzen.