Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Vorlesung Objektorientierte
Komponenten-Architekturen
(OOKA)
SS 2022

Prof. Dr. Sascha Alda

16. Mai 2022

Übungen zur Vorlesung
Objektorientierte Komponenten-Architekturen

SS 2022

Übungsblatt Nr. 5
(Abgabe bis: Montag, den 30. Mai 2022, **12:00 Uhr**. Auch eine spätere Präsentation einer erweiterten oder reiferen Version der Lösung wäre möglich)

Aufgabe 1 (Entwicklung einer Choreographie von Microservices)

Die Geschäftsleitung der Firma WirSchiff*en*Das GmbH ist wenig begeistert von der starren und wenig einsichtigen Haltung der Abteilung Technik zum Thema Microservices und verlangt daher, ohne Abstimmung mit dem Abteilungsleiter, einen ersten Proof-Of-Concept von *ihnen*! Man ist dabei interessiert, die kritische Analyse-Komponente innerhalb der Komponente „Manufacturing Products“ mittels Microservices zu optimieren. Man ist dabei angetan von der Idee des Ingenieurs, der im Rahmen einer ersten Anforderungsanalyse interviewt wurde (vgl. Unterlage Case Study, Folie 9):

„*Die Analyse-Komponente für Komponenten ist deswegen so komplex, weil für jedes optionale Equipment ein eigenständiger und abgeschlossener Algorithmus zur Validierung und Simulation der Konfiguration durchgeführt werden muss. Aktuell sind diese Algorithmen in einer Klasse integriert. Ob man das nicht mal auftrennen könnte, indem man die Algorithmen in einer Choreographie nebenläufig abarbeitet? Die übrigen Komponenten sollte also responsive bleiben! Möchte auch gerne die aktuellen Status der einzelnen Algorithmen sehen können, das ist in der aktuellen Implementierung nicht möglich. Auch ein Retry eines einzelnen Algorithmus sollte möglich sein. (...)*“

Natürlich soll die Analyse-Komponente nicht originalgetreu nach-implementiert werden, aber die wichtigsten Abläufe sollen simuliert werden. Gehen sie wie folgt vor:

- Simulieren sie die Eingabe einer Konfiguration des Optional Equipments für eine Diesel Engine. Dies kann eine einfache Eingabe-Maske sein, die über einen separaten Microservice bereitgestellt werden *kann*. Als Eingabe-Werte können sie die möglichen Werte aus der Tabelle auf Folie 9 verwenden. Können sie keine alternativen Eingabewerte identifizieren, so können sie Dummy-Werte annehmen.

- Eine Konfiguration soll über einen Microservice persistent verwaltet werden können.

- Die eigentliche Analyse soll über einen entsprechenden Button oder einem sonstigen Interaktionselement ausgeführt werden können. Durch diese Interaktion wird ein bestimmter „Anker“-Algorithmus ausgeführt.

- Wie von dem Ingenieur vorgeschlagen, *sollten* die Algorithmen auf *choreographierte* Microservices aufgeteilt werden, die sich nacheinander in einer Sequenz aufrufen können. Auch eine *parallele* Ausführung wäre denkbar, die dann durch einen Microservice wieder synchronisiert werden muss. Die Algorithmen können über entsprechende REST-Endpoints bei den Microservices aufgerufen werden. Sie können die Durchläufe der Algorithmen simulieren, indem sie z.B. den Thread der jeweiligen Implementierung für eine Zeit (z.B. 20 sec) anhalten (suspendieren). Nach Beendigung eines Algorithmus ruft der jeweilige Microservices den aus seiner Sicht *nächsten* Microservice (oder: die nächste*n* Microservices) auf. Das finale Ergebnis der Choreographie soll dann in der Benutzeroberfläche des Microservice dargestellt werden.

- Sie können die Algorithmen nach einer eigenen Systematik clustern (Beispiel: ein Algorithmus zur Validierung der Flüssigkeits-Konfiguration, also der Optional Equipments „Oil System“ und „Fuel System“). Somit sollten sie *mindestens* 3-4 Algorithmen bzw. Microservices bereitstellen. Sie können davon ausgehen, dass die Algorithmen unabhängig voneinander agieren können, sie sind also nicht auf das Vorergebnis eines Algorithmus angewiesen. Jeder Algorithmus nimmt sich aus dem Input, die gegebene Konfiguration des Optional Equipments, seinen Anteil, den er berechnen muss.

- Wie in der Anforderungsbeschreibung des Ingenieurs beschrieben, soll der Bearbeitungs-Status der Algorithmen in der Benutzeroberfläche der Anwendung stets zu entnehmen werden. Als Status können sie *pro* Algorithmus *zumindest* folgende Werte annehmen: „`running`“, „`failed`“, „`ready`“, „`not started`“.

- Beachten sie zudem den Fall, dass Microservices nicht erreichbar sind. Verwenden sie das Pattern Circuit Breaker (Kapitel 4), um die Widerstandsfähigkeit der gesamten Anwendung zu garantieren. Der Ausfall eines Microservices und die damit verbundenen Auswirkungen sollten sie natürlich entsprechend simulieren.

- Machen sie sich Gedanken über die Gestaltung der Schnittstellen der einzelnen Microservices; diese sollten als REST-Endpoints entwickelt werden.

Ich empfehle die Entwicklung der Microservice-basierten Architektur bzw. der einzelnen Microservices mit der Plattform „Spring Boot“ (vgl. Kapitel 2, Folie 77ff.). Für die Implementierung des Circuit Breaker empfehle ich das Framework Resilience4j:

https://github.com/resilience4j/resilience4j

Falls ihnen die Entwicklung eines Circuit Breaker mit einem Framework zu komplex erscheint, so lassen sie diese Anforderung zunächst aus oder implementieren sie den Ansatz eines Circuit Breaker selber.

Optional können sie überlegen, die Microservices mit einer *alternativen Technologie* wie z.B. Node.js oder dem Django REST Framework zu implementieren. Auch die Verwendung der Plattform Docker für eine Containerisierung der Microservices wäre denkbar.

Modellieren sie die Baustein-Sicht ihrer resultierenden Software-Architektur entsprechend. Auch einen ersten Prototyp sollen sie entwickeln und vorführen können.

Für die Entwicklung der Benutzeroberfläche können sie ein beliebiges Framework verwenden, so z.B. Angular, JavaFX oder Vaadin. Auch ein Command Line Interface (CLI) wäre denkbar für einen *ersten* Prototyp.

ToDo in der Übungsstunde: Entwerfen sie eine erste High-Level-Architektur sowie einen ersten „Papier-Prototyp“ der Benutzeroberfläche.

Ausblick für ein Semesterprojekt (nicht zwingend relevant für die Übung!!):

TA1: Integrieren sie ein Monitoring-Framework für die Überwachung ihrer Microservices. Hier empfehle ich die Verwendung des Frameworks Prometheus:
https://prometheus.io/

TA2: Integrieren sie ein beliebiges UI-Framework (kein CLI!).

Quellen für die Entwicklung für eine *eigenständige* Einarbeitung:

REST-basierter (Micro-)Service mit SpringBoot:
https://spring.io/guides/gs/rest-service/

REST-basierter (Micros-)Service mit Node.js und Express:
https://stackabuse.com/building-a-rest-api-with-node-and-express/

REST-basierter (Micro-)Service mit Django:
https://www.django-rest-framework.org/tutorial/quickstart/