|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Ziel der Gruppenarbeit „Trading Platform I“ im Rahmen dieser Vorlesung ist es, eine moderne, wettbewerbsfähige Internet-Handelsplattform zu entwickeln, welche als Software-Architektur auf Microservices setzt. Jeder Microservice soll dabei genau eine einzige Aufgabe erfüllen und über genau definierte Schnittstelen erreichbar sein bzw. mit anderen Microservices über die angebotenen Schnittstellen kommunizieren.**  **Denkbar wären zum Beispiel folgende Microservices:**   * **Shopping-Microservice** * **Bezahl-Microservice** * **Rating-Microservice** * **BauernladenAProduktkatalog-Microservice** * **WeingutBProduktkatalgo-Microservice** * **Storno-Microservice** * **Währungsrechner-Microservice**   **Fokus: Integration von „elektronischen (Geschäfts)-Prozessen“ und Microservices. Nehmen Sie in Ihren Ausführung auch Bezug auf die im Artikel „Microservices a definition of this new architectural term“**  **(**[**http://martinfowler.com/articles/microservices.html**](http://martinfowler.com/articles/microservices.html)**) beschriebenen Konzepte.**  **Teambezeichnung:** | | | | |  |  |  |  | | **Aufgabe 1 (25 Punkte)** |  |  |  | | **a)**  **Analyse: Machen Sie sich mit dem Ausgangs-Source-Code „SolTradingPlatform (ohne Polly)“ vertraut. Publizieren Sie die beiden Services „MeiShop“ und „IEGEasyCreditCardService“ in die Microsoft Azure Cloud und Testen Sie die Funktionalität (0 Punkte) https://github.com/hansgrabner/SolTradingPlatform2023**  **b)**  **Beschreiben Sie zuerst den Ansatz „Domain-Driven Design (DDD) im Zusammenhang mit Microservices. Beschreibung danach die Funktionalitäten / Verantwortlichkeiten der einzelnen Microservices – Stichwort: Business Capabilities**  **Detailbeschreibung der angebotenen Schnittstellen inkl. Datenaustauschformate**  **Detailbeschreibung der Datenhaltung – Stichwort: Decentralized Data Management**  **Umgang mit Ausfallsicherheit –Stichwort: Design for failure / Resilient Software Design**  **(10 Punkte)** | | | | |  | | | | | 1. **Service: MeiShop**   Schnittstelle:   1. **Service: IEGEasyCreditcardService**   Schnittstelle: | | | | | **DDD:**  **Domain-Driven Design (DDD)** ist ein Ansatz zur Softwareentwicklung, der sich auf die Modellierung komplexer Domänen konzentriert und die Softwarearchitektur um diese Domänen herum ausrichtet. Es zielt darauf ab, die Geschäftslogik und die Sprache der Domäne in den Vordergrund zu stellen und eine enge Zusammenarbeit zwischen Entwicklern und Fachexperten zu ermöglichen.  Im Zusammenhang mit Microservices wird DDD oft verwendet, um die einzelnen Microservices um die Kernkonzepte der Domäne herum zu strukturieren. Anstatt alle Funktionen in einem monolithischen System zu bündeln, werden verschiedene Microservices erstellt, die jeweils auf spezifische Geschäftsbereiche oder -fähigkeiten ausgerichtet sind. Dies ermöglicht eine bessere Skalierbarkeit, Wartbarkeit und Flexibilität.  Die Funktionalitäten und Verantwortlichkeiten der einzelnen Microservices werden entsprechend den Geschäftsfähigkeiten (Business Capabilities) der Domäne aufgeteilt. Jeder Microservice konzentriert sich auf eine bestimmte Geschäftsfähigkeit und stellt die erforderlichen Funktionen und Dienste bereit. Hier sind einige Beispiele für Geschäftsfähigkeiten und die zugehörigen Microservices:  **Benutzerverwaltungsservice:** Dieser Microservice ist für die Verwaltung von Benutzerkonten, Authentifizierung und Autorisierung zuständig. Er bietet Funktionen wie Registrierung, Anmeldung, Passwortänderung und Verwaltung von Benutzerprofilen.  **Produktkatalogservice:** Dieser Microservice verwaltet den Produktkatalog, einschließlich der Erstellung, Aktualisierung und Löschung von Produkten. Er stellt Funktionen wie Produktsuche, Abrufen von Produktinformationen und Hinzufügen von Produkten zum Warenkorb bereit.  **Bestellverwaltungsservice:** Dieser Microservice ist für die Verwaltung des Bestellprozesses verantwortlich. Er ermöglicht das Erstellen neuer Bestellungen, das Aktualisieren des Bestellstatus, das Abrufen von Bestelldetails und die Verarbeitung von Zahlungen.  **Lagerverwaltungsservice:** Dieser Microservice verwaltet den Lagerbestand und die Bestandsverfolgung. Er bietet Funktionen wie das Hinzufügen von Produkten zum Lagerbestand, Aktualisierung der Lagermengen und Überprüfung der Produktverfügbarkeit an.  **Versandservice:** Dieser Microservice kümmert sich um die Verwaltung des Versandprozesses. Er umfasst Funktionen wie das Erstellen von Versandetiketten, das Verfolgen von Sendungen und das Verwalten von Rücksendungen.  **Aufgabe 2 (10 Punkte)** | | | | | **2 weitere Microservice Produktkataloge: Erstellen Sie ein Microservice, welches eine Liste von Produkten anbietet. Der Inhalt der Liste soll dabei aus einem „microservice local datastore“ kommen – (Decentralized Data Management). Ersetzen Sie die hard codierten Werte im MeiShop/ProductList-Controller durch den Aufruf des soeben erstellen Services. Ein weiterer Produktkatalog-Service soll Produkte aus einem Text File auf einem FTP-Server auslesen und zur Verfügung stellen. (10 Punkte):** | | | | | **Aufagbe 3 (10 Punkte)** | | | | | **Skalierung, Ausfallssicherheit und Logging (Design for failure) für CreditPaymentService. Detailsbeschreibung: Publizieren Sie das Service „IEGEasyCreditCardService“ mehrfach in die Cloud und rufen Sie die Services im „Round Robin“ Stil auf. Falls es beim Aufruf eines Service zu einem Fehler kommt, soll es eine Retry-Logik geben, außerdem soll der aufgetretene Fehler mit Hilfe eines zentralen Logging-Service protokolliert werden. Nach n erfolglosen Versuchen, soll das nächste Service aufgerufen werden. Recherchieren Sie zusätzlich nach einem geeigneten Framework und Skalierungsmöglichkeiten in der Azure Cloud setzen Sie dieses gegebenenfalls ein(10 Punkte)** | | | | | **Aufgabe 4 (10 Punkte)** | | | | | **(theoretische) Überlegungen zum Einsatz von Asynchronen Kommunikationsstilen in der Handelsplattform (10 Punkte)** [**https://microservices.io/patterns/communication-style/messaging.html**](https://microservices.io/patterns/communication-style/messaging.html) | | | | | **Queues:**   * Registrierung von Clients * Warenkorbeinträge * Abmeldung von Clients * Health Check mit Reply * Payment-Abwickeln   **Topic:**   * Power up – Server * Erzeuger über eine Auftragserstellung informieren * Generelle Info an andere Gewerke   **Entkopplung:** Durch die Verwendung einer Message Queue kann die Kommunikation zwischen den verschiedenen Komponenten des Onlineshops entkoppelt werden. Anstatt dass die Komponenten direkt miteinander interagieren, senden sie Nachrichten an die Queue, und andere Komponenten können diese Nachrichten abrufen und verarbeiten. Dadurch wird die Abhängigkeit zwischen den Komponenten verringert, was die Skalierbarkeit und Wartbarkeit des Systems verbessern kann.  **Skalierbarkeit:** Ein Onlineshop kann starken Lastspitzen ausgesetzt sein, insbesondere während besonderen Aktionen oder zu Stoßzeiten wie dem Black Friday. Durch die Verwendung einer Message Queue können eingehende Anfragen zwischengespeichert und in einem kontrollierten Tempo verarbeitet werden. Dadurch können Ressourcen effizienter genutzt und Engpässe vermieden werden. Bei Bedarf können zusätzliche Verarbeiter hinzugefügt werden, um die Skalierbarkeit weiter zu verbessern.  **Robustheit:** Durch die asynchrone Kommunikation über eine Message Queue kann das System widerstandsfähiger gegen Fehler und Ausfälle sein. Wenn eine Komponente vorübergehend nicht erreichbar ist oder ausfällt, können Nachrichten in der Queue zwischengespeichert werden, bis die Komponente wieder verfügbar ist. Dadurch gehen keine Informationen verloren, und das System kann weiterhin arbeiten, ohne dass der Benutzer dies unmittelbar bemerkt.  **Entlastung der Benutzerschnittstelle:** Durch die Verwendung einer Message Queue können zeitintensive oder ressourcenintensive Aufgaben im Hintergrund ausgeführt werden, ohne die Benutzerschnittstelle zu blockieren. Beispielsweise können Bestellverarbeitungen, Bestandsaktualisierungen oder E-Mail-Benachrichtigungen in der Queue abgelegt und von anderen Komponenten verarbeitet werden. Der Benutzer erhält eine schnelle und reaktionsschnelle Benutzererfahrung, während die langwierigen Aufgaben im Hintergrund erledigt werden.  **Erweiterbarkeit:** Wenn weitere Funktionen oder Integrationen in den Onlineshop hinzugefügt werden sollen, können neue Komponenten entwickelt werden, die Nachrichten in die Queue senden und von anderen Komponenten verarbeitet werden. Dadurch kann das System flexibel erweitert werden, ohne dass bereits bestehende Komponenten stark angepasst werden müssen. | | | | | **Aufgabe 5 (10 Punkte)** | | | | | **Schreiben Sie ein zusätzliches „Paymentservice“. Dieses Payment-Service soll sowohl JSON, XML-Nachrichten als auch Nachrichten im Format CSV verarbeiten und erzeugen können. Orientieren Sie sich an dem Pattern -** [**HTTP Content Negotiation in REST APIs (restfulapi.net)**](https://restfulapi.net/content-negotiation/)  **(10 Punkte)** | | | | |  | | | | |  | | | |   **Aufgabe 6 (10 Punkte)**   |  | | --- | | **(theoretische) Überlegungen zu einem PaymentService-Broker. Dieses Service soll zwischen Shops und Payment-Services „vermitteln“.** [**SOA Patterns | Compound Patterns | Service Broker | Arcitura Patterns**](https://patterns.arcitura.com/soa-patterns/compound_patterns/service_broker)[**ZA\_Deloitte\_Digita\_Canonical\_Schemas.pdf**](https://www2.deloitte.com/content/dam/Deloitte/za/Documents/strategy/ZA_Deloitte_Digita_Canonical_Schemas.pdf) **http://www.enterpriseintegrationpatterns.com/patterns/messaging/CanonicalDataModel.html**    **Recherchieren Sie dazu zusätzliche Patterns** | |  | | **Kanonisches Datenmodell:**  Ein kanonisches Datenmodell (CDM) ist ein Designmuster, das verwendet wird, um eine einheitliche, übergreifende Darstellung von Daten zu erstellen, die von mehreren Systemen oder Anwendungen innerhalb einer Organisation genutzt werden kann.  Angenommen, verschiedene Shops und Payment-Services haben ihre eigene Art, Informationen über eine Transaktion darzustellen. Ein Shop könnte ein Objekt namens "Kauf" mit den Attributen "Käufer", "Produkt" und "Preis" haben. Ein Payment-Service könnte hingegen ein Objekt namens "Zahlung" mit den Attributen "Zahler", "Empfänger" und "Betrag" haben.  In diesem Fall könnte ein kanonisches Datenmodell ein einheitliches "Transaktions"-Objekt definieren, das Attribute wie "Kunde", "Produkt" und "Zahlungsbetrag" hat. Der PaymentService-Broker würde dann dieses einheitliche "Transaktions"-Objekt verwenden, um Informationen zwischen den Shops und den Payment-Services auszutauschen.  **Service Messaging Patterns - Publish-Subscribe**  Das Publish-Subscribe-Muster wie z. B. mit RabbitMQ bietet mehrere Vorteile in unserem PaymentService-Broker. Es entkoppelt die Shops und Payment-Services voneinander, was bedeutet, dass sie unabhängig voneinander skalieren können. Es ermöglicht auch eine asynchrone Kommunikation, da die Publisher nicht auf die Subscriber warten müssen, um ihre Nachrichten zu verarbeiten. Schließlich bietet RabbitMQ auch Funktionen für die Bestätigung und Speicherung von Nachrichten, um sicherzustellen, dass keine Transaktionsnachrichten verloren gehen.  In einem PaymentService-Broker könnten wir dieses Muster verwenden, um Transaktionsnachrichten zwischen Shops und Payment-Services zu vermitteln. Dabei könnte RabbitMQ als zentraler Broker dienen, der die Nachrichten von den Shops (den Publishern) empfängt und an die entsprechenden Payment-Services (die Subscribers) weiterleitet.  Publisher: In unserem Kontext sind die Shops die Publisher. Wenn eine Transaktion in einem Shop durchgeführt wird, sendet der Shop eine Nachricht an die RabbitMQ-Exchange (eine Art Postamt in RabbitMQ), die die Transaktionsdetails enthält.  Exchange: Die Exchange in RabbitMQ nimmt die Nachricht vom Publisher entgegen und leitet sie basierend auf den Routing-Regeln an die entsprechenden Queues weiter. Es gibt verschiedene Arten von Exchanges in RabbitMQ, aber für das Publish-Subscribe-Muster verwenden wir wahrscheinlich eine "Fanout"-Exchange, die Nachrichten an alle gebundenen Queues sendet.  Queues: In RabbitMQ sind Queues temporäre Speicher für Nachrichten, bevor sie an die Subscriber gesendet werden. Jeder Payment-Service hätte seine eigene Queue, um die Nachrichten zu empfangen, die für ihn bestimmt sind.  Subscriber: Die Payment-Services agieren als Subscriber. Sie verbinden sich mit ihren jeweiligen Queues und empfangen die Nachrichten, wenn sie bereit sind, sie zu verarbeiten.  **Compound Pattern -> Enterprise Service Bus**  In Bezug auf einen PaymentService-Broker könnte ein ESB auf verschiedene Arten nützlich sein:  Integration von Services: ESBs bieten einen zentralen Ort, um die Integration und Kommunikation zwischen verschiedenen Services zu erleichtern. Im Kontext des PaymentService-Brokers könnte ein ESB dazu dienen, die Kommunikation zwischen den Shops und den verschiedenen Payment-Services zu erleichtern und zu verwalten.  Transformation und Verarbeitung: ESBs unterstützen die Transformation und Verarbeitung von Nachrichten, um die Datenkonsistenz sicherzustellen. Sie könnten beispielsweise genutzt werden, um das Format der Transaktionsdaten so zu ändern, dass es von allen beteiligten Services gelesen werden kann.  Routen von Nachrichten: ESBs können auch zur intelligenten Nachrichtenweiterleitung genutzt werden, indem sie basierend auf bestimmten Kriterien oder Bedingungen entscheiden, an welchen Service eine bestimmte Nachricht gesendet werden soll.  Fehlerbehandlung und Wiederherstellung: ESBs bieten auch Mechanismen zur Fehlerbehandlung und Wiederherstellung, um sicherzustellen, dass die Nachrichten auch bei Netzwerk- oder Systemausfällen sicher und zuverlässig übertragen werden.  Protokollkonvertierung: Verschiedene Payment-Services könnten verschiedene Kommunikationsprotokolle verwenden. Der ESB kann dabei helfen, die Kommunikation zwischen Protokollen zu ermöglichen, indem er die Protokollkonvertierung durchführt. | |  |   **Aufgabe 7 (10 Punkte)**   |  | | --- | | **Webhook-Subscriber: Überlegen und implementieren Sie ein mögliches Webhook-Szenario (10 Punkte)** | |  | |  | |  |   **Aufgabe 8 (10 Punkte)**   |  | | --- | | **Machen Sie sich mit dem Begriff OData vertraut. Überlegen und implementieren Sie ein mögliches OData (Service & Client)-Szenario (10 Punkte)** | |  |   **Aufgabe 9 (10 Punkte)**   |  | | --- | | **Machen Sie sich mit dem Begriff SAGA-Pattern vertraut. Überlegen und implementieren Sie ein mögliches SAGA-Pattern Szenario(Service & Client)-Szenario (10 Punkte)** | |  | | Das SAGA-Pattern ist ein Designmuster, das in der Architektur von Microservices verwendet wird, um eine koordinierte Abwicklung von Transaktionen über mehrere Dienste hinweg zu ermöglichen. Es bietet eine Möglichkeit, komplexe Transaktionen aufzuteilen und deren Konsistenz in einer verteilten Systemlandschaft sicherzustellen.  Das SAGA-Pattern basiert auf dem Konzept von "Kompensationsaktionen" (Compensation Actions). Anstatt eine große Transaktion in einem einzigen Schritt durchzuführen, wird sie in mehrere Teiltransaktionen oder Schritte unterteilt. Jeder Schritt wird von einem spezifischen Microservice abgewickelt. Wenn ein Schritt erfolgreich abgeschlossen wurde, wird der nächste Schritt gestartet. Wenn ein Fehler auftritt oder eine Konsistenzverletzung festgestellt wird, werden Kompensationsaktionen ausgeführt, um die bereits durchgeführten Änderungen rückgängig zu machen und das System in einen konsistenten Zustand zurückzubringen.  Für den spezifischen Fall eines Shops und eines Zahlungsdienstes könnte die Implementierung des SAGA-Patterns wie folgt aussehen:  Initiierung der Transaktion: Wenn ein Kunde eine Bestellung aufgibt, wird eine Transaktion initialisiert. Der Shop-Service erstellt eine Bestellungsanfrage und ruft den Zahlungsdienst auf, um die Zahlung zu verarbeiten.  Zahlung durchführen: Der Zahlungsdienst validiert die Zahlungsinformationen und führt die Zahlung durch. Wenn die Zahlung erfolgreich ist, wird mit dem nächsten Schritt fortgefahren. Andernfalls wird eine Kompensationsaktion ausgeführt, um die Zahlung rückgängig zu machen oder zu stornieren.  Bestellung abschließen: Der Shop-Service bestätigt die Bestellung, aktualisiert den Bestellstatus und führt weitere erforderliche Aktionen durch. | |  |   **Aufgabe 10 (10 Punkte)**   |  | | --- | | **Machen Sie sich mit dem Begriff „Open Data“ vertraut und beschreiben Sie diesen in einigen wenigen Sätzen. Beschreiben Sie außerdem mögliche Anwendungsfälle im Zusammenhang mit der Handelsplattform** | |  | |  | |  |   Open Data bezeichnet Informationen, die frei verfügbar, zugänglich und weiterverwendbar sind. Diese Daten stammen meist von öffentlichen Einrichtungen, können aber auch von privaten Unternehmen kommen. Das Hauptziel der Open Data ist es, Transparenz zu fördern und die Nutzung von Daten durch Dritte zu erleichtern.  Im Zusammenhang mit einer Handelsplattform könnte Open Data verschiedene Anwendungsfälle haben:  1. Marktanalyse: Open Data kann genutzt werden, um Trends und Muster im Markt zu analysieren. Dies kann Händlern dabei helfen, ihre Strategien zu optimieren und bessere Geschäftsentscheidungen zu treffen.  2. Preisvergleich: Durch die Bereitstellung von Open Data über Produkte und deren Preise könnte eine Handelsplattform es Verbrauchern ermöglichen, Preise zu vergleichen und so die besten Angebote zu finden.  3. Transparenz: Die Bereitstellung von Open Data über die Herkunft von Produkten, ihre Produktionsprozesse und andere relevante Informationen kann zu mehr Transparenz in der Lieferkette führen. Dies kann Verbraucher dazu ermutigen, nachhaltigere oder ethischere Produkte zu wählen.  4. Produktentwicklung: Durch das Verständnis der Kundennachfrage und der Markttrends, die durch Open Data gewonnen wurden, können Unternehmen neue Produkte entwickeln oder bestehende verbessern, um besser auf die Bedürfnisse der Verbraucher einzugehen.  5. Kundenzufriedenheitsanalyse: Durch die Nutzung von Open Data über Kundenbewertungen und -feedback können Unternehmen ihre Produkte und Dienstleistungen verbessern, um die Kundenzufriedenheit zu erhöhen. |