|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Ziel der Gruppenarbeit „Handelsplattform Erweiterung 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 & Teammitglieder:** | | | | | |  | **Vorname Nachname** | | | **Aufgabengebiet** | | **Elena** | | **Krois** |  | **1 + 6** | | **Ferdinand** | | **Schlögl** |  | **1 + 6** | | **Stefan** | | **Obendrauf** |  | **1 + 2** | | **Maximilian** | | **Brunnsteiner** |  | **1 + 5** | | **Michael** | | **Klug** |  | **1 + 4** | | **Philipp** | | **Rammer** |  | **1 + 3** | | **Alb** | | **Pellumbi** |  | **1 + 3** | | 1. **Analyse: Machen Sie sich mit dem Ausgangs-Source-Code „SolHandelsplattform V.01“ vertraut. Publizieren Sie die beiden Services „BlackFriday“ und „IEGEasyCreditCardService“ in die Microsoft Azure Cloud und Testen Sie die Funktionalität (0 Punkte)** | | | | | | **Both Microservices hosted. Links:**  <http://iwi17-blackfriday.azurewebsites.net/swagger/>  <http://iwi17-easycreditcardservice.azurewebsites.net/swagger/> | | | | | | 1. **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 BlackFriday/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. Der Produktkatalog soll ebenfalls lokal, angereichert um Verkaufsinformationen, im BlackFriday-Service zur Verfügung stehen(10 Punkte):** | | | | | | **URLs:**  <http://iwi17-productservicefile.azurewebsites.net/swagger/>  <http://iwi17-productservicelocal.azurewebsites.net/swagger/>    **File Product Service (FTP):**    **Local Storage Product Service:** | | | | | | 1. **Skalierung, Ausfallssicherheit und Logging (Design for failure) für CreditPaymentService. Detailsbeschreibung: Publizieren Sie das Service „IEGEasyCreditCardService“ mehrfach in die Cloud und rufen Sie die Services aus dem CashDesk-Controller im „Round Robin“ Stil auf. Falls es beim Aufruf eines Services zu einem (Kommunikations)fehler kommt, soll es eine Retry-Logik geben, außerdem soll der aufgetretene Fehler protokolliert werden. Nach n (config-Wert) erfolglosen Versuchen, soll das nächste Service aufgerufen werden. (10 Punkte)** | | | | | | Das mehrfach publizierte Service ist über mehrere URLs erreichbar. Diese werden, gesteuert über eine Variable, nacheinander bei aufeinanderfolgender Benutzung des Service aufgerufen. Abgelegt werden diese der Einfachheit halber in einer Liste von URLs als Strings.  /Users/philipp/Desktop/Bildschirmfoto 2017-12-15 um 12.14.59.png  Zudem wird, falls ein Verbindungsfehler auftritt, eine definierte Anzahl an Verbindungswiederholungen (per Einstellung: \_repeats = 3) für das gleiche Service durchgeführt. Sollten alle diese Versuche scheitern, wird auf das nächste im Round Robin-Verfahren definierte Service weitergeschaltet. Dies passiert im Fehlerfall solange, bis alle Services mit der definierten Anzahl an Wiederholungen probiert wurden. Sollten alle Versuche fehlschlagen, wird ein Fehler zurürckgegeben (5XX internal Server Error), da das Service dann als nicht verfügbar klassifiziert werden muss.  Hierbei müssen verschiedene Fehlerfälle berücksichtigt werden, da z.B. das Service auf Grund einer falschen oder nicht erreichbaren URL (DNS-Error, timeout, ...) oder auf Grund von Fehlern im Service (Rückgabecode) Fehler erzeugen kann.  /Users/philipp/Desktop/Bildschirmfoto 2017-12-15 um 12.21.24.png  Dabei wird detailliert geloggt, welches Service aufgerufen wurde und ob der Verbindungsversuch erfolgreich war bzw. dass ein erneuter Versuch unternommen wurde oder auf ein anderes Service umgeschaltet wurde. | | | | | | 1. **(theoretische) Überlegungen zum Einsatz von Asynchronen Kommunikationsstilen in der Handelsplattform (10 Punkte)** <http://soapatterns.org/design_patterns/reliable_messaging><http://soapatterns.org/design_patterns/event_driven_messaging> **http://soapatterns.org/design\_patterns/asynchronous\_queuing** | | | | | | **Softwareseite**   * Asynchrone Kommunikation kann unter den Microservices Anwendungen eingesetzt werden, um eine strikt nacheinander folgende Abhängigkeit abzuschaffen. Dadurch kann ein Ausfall der Anwendungen nacheinander, ähnlich wie ein Dominoeffekt, verhindert werden; Ein Ausfall, eines in der Reihenfolge am Anfang liegenden Services, würde bei einer asynchronen Kommunikation, nicht alle restlichen Services sofort mitausfallen lassen.   Die asynchrone Kommunikation der Services bewirkt zudem eine Entkopplung über die Zeit, da die Services nicht wie bei einer synchronen Kommunikation auf eine Antwort der vorherigen Anwendung warten.    *Referenz:*  *https://docs.microsoft.com/de-de/dotnet/standard/microservices-architecture/architect-microservice-container-applications/communication-in-microservice-architecture*  **Kundeseite**   * Einsatz einer Subscribe- bzw. Share-Option mit deren Hilfe der Plattformbetreiber laufende Informationen zu Aktionen, Rabatten und anderen Erneuerungen innerhalb der Handelsplattform, an alle abonnierten User senden kann. Diese Funktion wird in den meisten Fällen mittels der SSE-Anwendung eingesetzt und an alle Abonnenten bzw. Newsgroup wird ein entsprechender Newsletter ausgesendet. * Ebenso kann als asynchrones Kommunikationstool eine Fragefunktion mittels WS-Anwendung eingeführt werden. Die Kunden/User stellen dabei ihre aufgetretenen Fragen/Anliegen mittels Anfrageformular an die Handelsplattformbetreiber. Eine Rückmeldung oder Antwort erfolgt nach Bearbeitung und zeitlich versetzt. * Asynchrone Punkt-zu-Punkt-Verbindungen (Messaging über Queues) bei dem der Kunde ein Angebot berechnen lassen möchte wäre eine weitere Anwendung. Ein Partnerservice wird über eine asynchrone Schnittstelle aufgerufen während das Angebotsservice auf eine Reply-Queue wartet. Wenn der Partnerservice die entsprechenden Daten liefert, wird eine Berechnung gestartet und das Ergebnis anschließend zum Client zurückgesendet. Auch hier erfolgt das Ergebnis nicht unmittelbar, sondern zeitlich versetzt * Mittels Datenduplikation können bestehende Daten/Angebote von Kunden dupliziert werden. Dabei werden die Daten der Kunden in einem eigenen Datenarchiv eines Services gespeichert. Das hat zur Folge, dass ein schnellerer Abruf und somit Bearbeitung möglich ist. Die Informationen werden von einem Service geliefert und dem Angebotsservice bereitgestellt um diese effektiv nutzen zu können. | | | | | |  | | | | | | 1. **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://soapatterns.org/design\_patterns/content\_negotiation  (10 Punkte)** | | | | | |  | | | | | |  | | | | |   **URL:**  <https://iwi17-paymentservice.azurewebsites.net/swagger/>  **Startup.cs:**  public void ConfigureServices(IServiceCollection services)  {  services.AddMvc(options =>  {  options.RespectBrowserAcceptHeader = true;  options.InputFormatters.Insert(0, new CsvInputFormatter());  options.OutputFormatters.Insert(0, new CsvOutputFormatter());  options.FormatterMappings.SetMediaTypeMappingForFormat("csv", MediaTypeHeaderValue.Parse("text/csv"));  }).AddXmlSerializerFormatters();  services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", new Info { Title = "ProductServiceFile API", Version = "v1" });  });  }  **CSVOutputFormatter:**  public class CsvOutputFormatter : TextOutputFormatter  {  public string ContentType { get; private set; }  public CsvOutputFormatter()  {  ContentType = "text/csv";  SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/csv"));  SupportedEncodings.Add(Encoding.GetEncoding("utf-8"));  }  protected override bool CanWriteType(Type type)  {  if (typeof(Payment).IsAssignableFrom(type)  || typeof(IEnumerable<Payment>).IsAssignableFrom(type))  {  return base.CanWriteType(type);  }  return false;  }  public async override Task WriteResponseBodyAsync(OutputFormatterWriteContext context, Encoding selectedEncoding)  {  var response = context.HttpContext.Response;  Type type = context.Object.GetType();  Type itemType;  if (type.GetGenericArguments().Length > 0)  {  itemType = type.GetGenericArguments()[0];  }  else  {  itemType = type.GetElementType();  }  StringWriter \_stringWriter = new StringWriter();  if (true)  {  \_stringWriter.WriteLine(  string.Join<string>(  ";", itemType.GetProperties().Select(x => x.Name)  )  );  }  foreach (var obj in (IEnumerable<object>)context.Object)  {  var vals = obj.GetType().GetProperties().Select(  pi => new {  Value = pi.GetValue(obj, null)  }  );  string \_valueLine = string.Empty;  foreach (var val in vals)  {  if (val.Value != null)  {  var \_val = val.Value.ToString();    if (\_val.Contains(","))  \_val = string.Concat("\"", \_val, "\"");  if (\_val.Contains("\r"))  \_val = \_val.Replace("\r", " ");  if (\_val.Contains("\n"))  \_val = \_val.Replace("\n", " ");  \_valueLine = string.Concat(\_valueLine, \_val, ";");  }  else  {  \_valueLine = string.Concat(\_valueLine, string.Empty, ";");  }  }  \_stringWriter.WriteLine(\_valueLine.TrimEnd(";".ToCharArray()));  }  var streamWriter = new StreamWriter(response.Body);  await streamWriter.WriteAsync(\_stringWriter.ToString());  await streamWriter.FlushAsync();  }  }  **CSVInputFormatter:**  public CsvInputFormatter()  {  SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/csv"));  }  public override Task<InputFormatterResult> ReadRequestBodyAsync(InputFormatterContext context, Encoding encoding)  {  var type = context.ModelType;  var request = context.HttpContext.Request;  MediaTypeHeaderValue requestContentType = null;  MediaTypeHeaderValue.TryParse(request.ContentType, out requestContentType);  var result = ReadStream(type, request.Body);  return InputFormatterResult.SuccessAsync(result);  }  public override bool CanRead(InputFormatterContext context)  {  var type = context.ModelType;  if (typeof(Payment).IsAssignableFrom(type)  || typeof(IEnumerable<Payment>).IsAssignableFrom(type))  {  return true;  }  return false; ;  }  private object ReadStream(Type type, Stream stream)  {  Type itemType;  var typeIsArray = false;  IList list;  if (type.GetGenericArguments().Length > 0)  {  itemType = type.GetGenericArguments()[0];  list = (IList)Activator.CreateInstance(type);  }  else  {  typeIsArray = true;  itemType = type.GetElementType();  var listType = typeof(List<>);  var constructedListType = listType.MakeGenericType(itemType);  list = (IList)Activator.CreateInstance(constructedListType);  }  var reader = new StreamReader(stream);  bool skipFirstLine = true;  while (!reader.EndOfStream)  {  var line = reader.ReadLine();  var values = line.Split(";".ToCharArray());  if (skipFirstLine)  {  skipFirstLine = false;  }  else  {  var itemTypeInGeneric = list.GetType().GetTypeInfo().GenericTypeArguments[0];  var item = Activator.CreateInstance(itemTypeInGeneric);  var properties = item.GetType().GetProperties();  for (int i = 0; i < values.Length; i++)  {  properties[i].SetValue(item, Convert.ChangeType(values[i], properties[i].PropertyType), null);  }  list.Add(item);  }  }  if (typeIsArray)  {  Array array = Array.CreateInstance(itemType, list.Count);  for (int t = 0; t < list.Count; t++)  {  array.SetValue(list[t], t);  }  return array;  }  return list;  }  }   |  | | --- | | 1. **(theoretische) Überlegungen zu einem PaymentService-Broker. Dieses Service soll zwischen Shops und Payment-Services „vermitteln“.** <http://serviceorientation.com/serviceorientation/service_loose_coupling><http://serviceorientation.com/serviceorientation/service_discoverability> **http://soapatterns.org/design\_patterns/data\_format\_transformation**   **http://www.enterpriseintegrationpatterns.com/patterns/messaging/ CanonicalDataModel.html**  **http://soapatterns.org/design\_patterns/canonical\_schema**  **http://soapatterns.org/design\_patterns/canonical\_protocol  (10 Punkte)** | |  | |  |   **Einleitung**:  Ein Service Broker ist ein Bindeglied zwischen Service-Anbieter (Service Provider) und den Service-Nutzer (Service Consumer). Gründe für die Notwendigkeit eines Service Brokers können sein:   * Services verwendenden verschiedene Datenformate (z.B. verschiedene Dateiformate: XML, TXT, CSV…) * Services verwenden unterschiedliche Datenmodelle (z.B. Objektklassen: „Auto“ vs. „Kfz“…) * Services benutzen verschiedene Kommunikationsprotokolle (z.B. SOAP over http v1.1 zu SOAP v1.2 oder JSON-RPC)   Service Broker  Abbildung 1: http://soapatterns.org/compound\_patterns/service\_broker  Der Scope (welches Service über den Broker abgehandelt wird) wird vom Entwickler/team selbst definiert, denn ein Service Broker benötigt Rechenkapazität und erhöht somit auch die Latenz der Kommunikation. Die Aufgaben eines Service Broker sind das Sicherstellen einer asynchronen Kommunikation (Messaging) zwischen Services und Vereinheitlichung der Kommunikation.  Ein theoretischer PaymentService Broker muss zwischen Shops und Payment Services vermitteln. Ein Beispiel dafür wäre die Eingabe der Kredikarteninformationen beim Bezahlvorgang. Da es aber mehrere Bezahlservices gibt, die jeweils die Käuferinformation in unterschiedlichen Formaten fordern, müssen in jedem Shop Service sämtliche Übersetzungsschnittstellen für die alle Bezahldienste erstellt werden. Der Service Broker hingegen transformiert die Daten in ein einheitliches Format und leitet sie an das spezifische Bezahlservice weiter. Jedes Zielservice kann nun die einheitlichen Nachrichten wieder in das gewünschte Datenformat zurückübersetzen.  Wichtig dabei ist das Prinzip der „losen Koppelung“. Die Shop-Services und Payment-Services müssen vollständig von dem jeweils andern Datenformat bzw. Datenbankmodellen entkoppelt sein. Z.B. eine Änderung im Datenmodell eines Shop-Services dürfen sich nicht unmittelbar auf den Payment-Service auswirken. Im oben genannten Beispiel dürfen Änderungen des Datenmodells der Käuferinformationen nicht direkt die Arbeitsweise eines Zahlungsabwicklungsservice beeinflussen. Dafür wird eine Art systemübergreifende Daten-Virtualisierung notwendig, das „kanonische Datenmodell“.  http://www.enterpriseintegrationpatterns.com/img/CanonicalDataModel.gif  Abbildung 2: <http://www.enterpriseintegrationpatterns.com/patterns/messaging/CanonicalDataModel.html>  Ein kanonisches Datenmodell ist objektorientiert und soll die Kommunikation mit dem Service Broker vereinheitlichen. Es reduziert die Aufwände beim „Übersetzen“ der Daten für die einzelnen Zielservices. Es wird nur noch eine zentrale Übersetzung ins kanonische Datenmodell benötigt. Die Services müssen sozusagen die Sprachen der anderen Services nicht mehr kennen. Ein weiterer Vorteil ist die Abgrenzung der einzelnen Verantwortungsbereiche. Jedes Entwicklerteam ist selbst dafür zuständig die Datenschnittstellen so zu implementieren, dass die Dienste mit dem zentralen Service Broker kommunizieren können. Das bedeutet auch, dass zukünftige Services verhältnismäßig problemlos an vorhandene Komponenten angebunden werden können, im Sinne der Kommunikationsschnittstellen.    Abbildung 3: Point-To-Point Kommunikation zwischen Services  Diese Abbildung veranschaulicht eine Point-to-Point Kommunikation zwischen Diensten. Die farbliche Unterscheidung der Kommunikationswege stellt die verschiedenen Protokolle und Datenformate dar. In jedem Service müssen die nötigen Schnittstellen für die Kommunikation mit implementiert werden.  Folgende Abbildung zeigt einen dazwischengeschalteten Payment-Service Broker.    Abbildung 4: Shops mit Payment Service Broker  Das Datenformat für die Kommunikation ist vereinheitlicht und verringert somit die Abhängigkeiten zwischen den Services. Bezahlinformation werden vom Shop Service in das kanonische Datemodell transformiert und den Payment Service Broker geschickt. Dieser leitet die Daten (ebenfalls im kanonischen Datenmodell) weiter an das Ziel-Bezahlservice. Am Endpunkt werden die kanonischen Daten in das vom Bezahlservice verwendete Format umgewandelt. Das bedeutet, dass die einzelnen Services intern nicht das kanonische Datenformat verwenden müssen. Ob eine Anbindung an einen Service Bus mit kanonischen Datenmodell sinnvoll ist, muss je nach Service individuell entschieden werden:  Wann und wo sollte ein kanonisches Datenmodell eingeführt werden? Diese Frage führt im Projekt zu vielen Diskussionen. Verfechter des kanonischen Datenmodells können leicht übers Ziel hinausschießen, wenn sie ein kanonisches Datenmodell über sämtliche Datenrepräsentationen in allen Applikationen stülpen wollen. Dies könnte schnell das Ziel der losen Kopplung torpedieren, da aus allen Datenbanken und Applikationen Abhängigkeiten ins kanonische Datenmodell entstehen können, die das Gesamtsystem in einen unwartbaren kanonischen Moloch verwandeln.  (<https://www.doag.org/formes/pubfiles/1387278/120-2009-K-DEV_SOA-Maier-SOA_aus_dem_wahren_Leben_Kanonisches_Datenmodell.pdf>, Berthold Maier, Oracle Deutschland GmbH, S.1 )   |  | | --- | | 1. **Webhook-Subscriber mit Worfklow-Tracking - (Dezember)** | |  | |  | |