Integration einer Lösung für die individuelle Anpassung von Produkten mit dem ZAKEKE Produktkonfigurator in Salesforce Commerce

I **II** **III** **IV**

Projektarbeit Nr.

vorgelegt am: [redacted]

von: [redacted]

Matrikelnr.: [redacted]

DHGE Campus: [redacted]

Studienbereich: Technik

Studiengang: Praktische Informatik

Kurs: [redacted]

Ausbildungsstätte: dotSource GmbH

Betreuer: [redacted]

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API ..... Programmierschnittstelle
B2B ..... Business-to-Business
CSS ..... Cascading Style Sheets
DHGE ..... Duale Hochschule Gera-Eisenach
DML ..... Data Manipulation Language
GUI ..... Graphical User Interface
HTML ..... Hypertext Markup Language
HTTP ..... Hypertext Transfer Protocol
ID ..... Identifikator
JSON ..... JavaScript Object Notation
LWC ..... Lightning Web Component
MVC ..... Model View Controller
OAuth ..... Open Authorization
REST ..... Representational State Transfer
SKU ..... Stock Keeping Unit
SOQL ..... Salesforce Object Query Language
SQL ..... Structured Query Language
URL ..... Uniform Resource Locator

1 Einleitung

In der modernen Zeit gibt es immer mehr einzigartige und branchenspezifische Arten, Softwarelösungen zu entwickeln. Ein Programmierer hat viele unterschiedliche Möglichkeiten, eine Problemstellung anzugehen und es ist an ihm zu wählen, welche Lösung die Beste für den gegebenen Fall ist. Daraus entstehen immer effizientere und kundenfreundlichere Lösungen für spezielle Anwendungsfälle. Jedoch ist es durch diese große Varianz an Programmiersprachen, Plattformen und Programmierarten nicht immer einfach, zwei bestehende Systeme funktional zusammenzuführen. Deswegen müssen Softwareentwickler heutzutage immer häufiger Schnittstellen zwischen zwei auf verschiedenen Techniken basierende Anwendungen entwickeln, um einen reibungslosen Datenaustausch zwischen ihnen zu ermöglichen.

Diese wissenschaftliche Arbeit soll sich mit der Implementierung einer neuen Funktionalität in ein bestehendes Webshopsystem beschäftigen. Dabei wird ein möglicher Lösungsansatz für die Implementierung als Konzept entstehen und technisch umgesetzt werden. Der Durchführung soll eine ausführliche Analyse unterliegen, aus welcher Schlüsse und Erkenntnisse für weitere Projekte im gegebenen Sachzusammenhang gewonnen werden sollen. Dabei wird unter anderem auf den Mehrwert und die Wiederverwendbarkeit der entstandenen Lösung eingegangen, sodass ein Entwickler sich über die mögliche Implementierung ausführlich informieren und diese mit den gewonnenen Erkenntnissen der Arbeit einfach replizieren kann.

1.1 Beschreibung des IST-Zustand

Ein Kunde der dotSource besitzt einen Webshop, welcher direkt an andere Unternehmen gerichtet ist, um ihnen besondere Konditionen für größere Bestellungen zu ermöglichen. Dies wird als B2B Commerce (Business to Business Commerce) bezeichnet. In diesem Webshop können sich Geschäftskunden Waren aus einem voreingestellten Sortiment in den Warenkorb legen über einen Bestellprozess abwickeln. Dabei geben sie wichtige Kundendaten wie Zustellungsadresse, Zahlungsadresse oder die gewünschte Zahlungsmethode an. Die Produkte sind mit Preisen, Lagerverfügbarkeit und Beschreibung statisch im System hinterlegt und werden von Mitarbeitern der Firma aktualisiert und verwaltet. Um diese Funktionalitäten anzubieten, wird die Salesforce Commerce Cloud eingesetzt. Salesforce hilft Unternehmen mit ihren Softwarelösungen beim Verkauf, beim Marketing, bei der Analyse und bei der Kommunikation mit Kunden. Dazu stellt ihnen Salesforce online eine Plattform zur Verfügung, auf welcher die Unternehmen ihre Daten und deren Beziehungen einfach und automatisiert

verwalten können.[1] Salesforce B2B Commerce bietet viele hauseigene Funktionalitäten für den Onlinehandel mit Unternehmen. Jedoch gibt es keine direkt in Salesforce integrierte Möglichkeit für Kunden, Produkte mit besonderen, individuell gestalteten Texten oder Bildern zu bestellen. Bei solchen Wünschen ist eine direkte Kontaktaufnahme mit einer detaillierten Beschreibung der gewünschten Produktanpassung notwendig.

1.2 Beschreibung des SOLL – Zustand

Diese fehlende Funktionalität soll mithilfe des externen Anbieters ZAKEKE in den Salesforce Shop integriert werden. Die Softwarelösungen von ZAKEKE sind verschiedene Konfiguratoren für die Individualisierung von Waren durch den Kunden selbst vor dem Kauf. Diese haben jedoch keine direkte Kompatibilität mit der Salesforce B2B Commerce Plattform und benötigen eine Schnittstelle, um mit dem Shopsystem kommunizieren und Daten austauschen zu können. Diese Schnittstelle muss direkt in Salesforce programmiert werden, um den Konfigurator später im Bestellprozess verwenden zu können. Am Ende sollen die Kunden bei spezifischen Produkten die Möglichkeit haben, den ZAKEKE Konfigurator für das Produkt zu öffnen und so die Ware nach ihren Vorstellungen individualisieren können. Das ZAKEKE System soll daraufhin druckbare Entwürfe von dem angepassten Produkt erstellen und dem Kunden nach dem Bestellprozess via E-Mail zur Verfügung stellen. Gleichzeitig muss der Konfigurator den Salesforce Bestellprozess darüber informieren, dass ein Produkt individualisiert wurde und somit eventuell preislich angepasst werden muss. Diese Information soll auch am Ende auf der Bestellübersicht einsehbar sein. Für die Nutzung der Schnittstelle ist es notwendig, Zugangsdaten des ZAKEKE Systems in Salesforce zu speichern. Diese müssen von Mitarbeitern aktualisiert werden können. Die technischen Anforderungen spiegeln sich in Tabelle 1 wieder.

[1] [Sal22].

Anforderung	**Beschreibung**
Datenmodellerweiterung der Datenbank	Die Salesforce Standardobjekte sollen um benutzerdefinierte Felder erweitert werden, welche Informationen über die individuellen Konfigurationen speichern.
Authentifizierungslogik	Salesforce soll sich im ZAKEKE System authentifizieren können.
Visuelle Benutzeroberflächen	Der Kunde soll den ZAKEKE Konfigurator mithilfe von grafischen Komponenten im Webshop bedienen können.
Backend Logik	Die Salesforce Organisation soll alle vom Konfigurator empfangenen Daten verarbeiten und in die Standardprozesse integrieren.

Tabelle 1: Technische Anforderungen

2 Konzepterstellung

Bei der Planung des Konzepts für die Integration des ZAKEKE Produktkonfigurators sind drei Eigenschaften vorrangig zu beachten. Zum Einen steht die Wiederverwendbarkeit des Quellcodes als zentrales Ziel, um auch zukünftig die Funktionalitäten in weiteren dotSource Projekten anwenden zu können. Dazu kommt die Unabhängigkeit des Codes als zweite Eigenschaft, welche es ermöglichen soll, die geschaffene Logik getrennt von dem ZAKEKE System auch mit anderen Produktkonfiguratorlösungen zu nutzen. Bei dem Entwurf ist ebenso auf eine gute Wartbarkeit des Quellcodes zu achten, um es zu ermöglichen, auch langfristig noch Änderungen an der Logik durchzuführen. Um sich diesen drei Zielsetzungen zu widmen, ist ein durchdachtes Konzept für die benötigte Softwarearchitektur notwendig.

2.1 Die grundlegende Softwarearchitektur

Unter Softwarearchitektur werden die zu treffenden Entscheidungen verstanden, wenn ein System in verschiedene Softwarekomponenten unterteilt wird, welche miteinander interagieren. Strukturen werden geschaffen, um Logik zu bündeln und unabhängig voneinander warten zu können. Die strategischen Entscheidungen der Softwarearchitektur realisieren langlebige Software, ermöglichen schnelle Anpassungen und steigern die Effizienz bei der Software-Entwicklung.[2]

In dem gegebenen Anwendungsfall ist es wichtig, die Benutzeroberfläche von der dahinter liegenden Logik zu trennen. Beide Komponenten sollten individuell für sich funktionstüchtig und zu testen sein. Beispielsweise soll eine Anwendungsfallsimulation oder ein Unit-Test mit der Benutzeroberfläche auch möglich sein, wenn es die benötigten Daten nicht aus der ZAKEKE Schnittstelle bekommt, sondern diese für den Testfall manuell bereitgestellt werden. Die Komponente erkennt nicht, woher die Daten kommen, aber kann sie unabhängig davon an der richtigen Stelle im System anzeigen und einbetten. Dasselbe soll für die Logik möglich sein. Ziel ist es, eine Softwarekomponente zu schaffen, welche sich aus dem Salesforce Umfeld die richtigen Daten ziehen kann und diese ordnungsgemäß verarbeitet, ohne überhaupt eine grafische Benutzeroberfläche (GUI) zu benötigen. Soll die Komponente dann also getestet werden, reichen Konsoleneingaben vollständig aus.

[2] [INZ15].

2.2 Benötigte Daten für die Kommunikation der Systeme

Um ein Konzept für die Schnittstelle zwischen dem Konfigurationssystem von ZAKEKE und der Salesforce Platform entwerfen zu können, ist zuerst eine Analyse der Datenstrukturen nötig, welche übergeben werden. Hierbei wird in zwei Kategorien unterteilt, einmal die Daten, die dem ZAKEKE System übergeben werden und zum anderen die Daten, die das ZAKEKE System an die Salesforce Organisation zurückgibt. Das Ziel der Schnittstellenlogik ist es, Produktinformationen der ersten Kategorie aus der Salesforce Organisation zu lesen und diese dem Konfigurator bereitzustellen, während sie die Informationen aus der zweiten Kategorie in die Datenbank und die Abläufe des Bestellprozesses einpflegen muss.

2.2.1 Vom Konfigurator geforderte Daten

Die wichtigste Information, die beim Start des Konfigurationsprozesses an ZAKEKE übergeben werden muss, ist der Identifikator (ID) des Produktes. Mit diesem kann das System nachvollziehen, welches Produkt aus der Datenbank bearbeitet werden soll. Weiterhin wird die Anzahl übergeben, wie häufig das zu konfigurierende Produkt bestellt werden soll. Um eine Preisanpassung der individualisierten Produkte zu ermöglichen, wird ebenfalls der Preis des unangepassten Produktmodells, sowie der Preis für eine konfigurierte Variante übergeben. Zum Schluss kann optional noch eine Liste mit Attributen übermittelt werden, welche besondere Eigenschaften wie Grundfarbe oder Größe an den ZAKEKE Konfigurator übergibt. Sollte das Produkt bereits im Warenkorb gewesen sein, und der Konfigurator wurde von dort aus gestartet, so ist es notwendig, die Warenkorb-Identifikationsnummer zu übermitteln, damit die Logik nach dem Individualisierungsprozess den alten, unangepassten Eintrag durch die neue konfigurierte Variante ersetzen kann.

2.2.2 Nach dem Konfigurierprozess empfangene Daten

Nach erfolgreicher Individualisierung des Produktes mit dem ZAKEKE Produktkonfigurator werden zwei wichtige Informationen zurück an die Salesforce Platform übermittelt. Zum einen wird eine Design-Identifikationsnummer übergeben, welche es der Schnittstelle ermöglicht, das konfigurierte Design eindeutig zuzuordnen, um später an relevante Informationen der Individualisierung zu kommen. Dazu wird zusätzlich ein “Uniform Resource Locator” (URL), also eine Webadresse übermittelt, der ein Downloadlink für die entstandenen Grafiken ist.

2.3 Die ZAKEKE Logik als Interface

Um die grundlegende Salesforce Logik unabhängig vom Anbieter ZAKEKE und dessen Diensten zu gestalten, muss die direkte Schnittstelle als austauschbarer Baustein im System agieren. Auf diese Weise wird die Wiederverwendbarkeit der Implementierung für andere Projekte der dotSource garantiert, auch wenn das Projekt als Anforderung eine andere Konfiguratorenlösung als das Produkt von ZAKEKE hat. Ein möglicher Lösungsansatz für diese Problemstellung ist das Verwenden von Interfaces.

Ein Interface ähnelt einem Entwurfsplan für Klassen, der dazu dient, Gemeinsamkeiten wie gleiche Funktionalitäten, welche mehreren Klassen zugrunde liegen, in einer seperaten Klasse zu definieren. Es legt fest, welche Methoden und Konstanten alle Klassen besitzen müssen, welche das Interface implementieren. Auf diese Art und Weise wird sichergestellt, dass die Funktionen der Klassen exakt die richtigen und selben Namen haben. So kann eine Logik mittels der festgelegten Namen die Methoden der Klasse aufrufen, ohne genau wissen zu müssen, mit welcher der vielen implementierenden Klassen des Interfaces sie kommuniziert. Dies garantiert, dass die einzelnen Klassen untereinander ausgetauscht werden können, ohne dass die Logik in ihrer Funktionsweise behindert wird oder einen Unterschied merkt. Die Logik weiß nicht, auf welche Art die Daten bereit gestellt werden, aber sie weiß, wie sie universell darauf zugreifen kann, denn sie kennt den einheitlichen Entwurfsplan, welchem die Struktur der Klasse unterliegt.

Im gegebenen Projekt wird ein Interface für einen Konfigurationsservice entworfen. Jeder Konfigurationsservice soll dieselben Funktionalitäten bieten, auch wenn dies auf unterschiedliche Art und Weise geschehen kann. So müssen Daten bereitgestellt und empfangen werden, damit diese in die Salesforce Organisation eingepflegt werden können. Am Ende soll ein solcher Konfigurationsservice implementiert werden, spezifisch zugeschnitten auf die Anforderungen von ZAKEKE. Ist es jedoch jemals nötig, ein anderes System anzuschließen, beispielsweise aufgrund von Konditionsänderungen der Entwickler von ZAKEKE, kann dieser spezifische Service einfach durch einen anderen Service ausgetauscht werden, der unserem Interface unterliegt, und die Funktionalität mit dem Rest der Logik bleibt erhalten.

2.4 Planung der Authentifizierung

Wird in ein bestehendes System eine Schnittstelle an eine Drittanbieterlösung implementiert, ist nahezu immer eine Art der Authentifizierung notwendig, um mithilfe der persönlichen Zugangsdaten die Berechtigung für den Zugriff nachzuweisen. Es gibt viele verschiedene Authentifizierungsarten, die sich in Komplexität und Sicherheit unterscheiden. Das einfachste Beispiel ist eine simple Passwort Authentifizierung, bei welcher sich ein Nutzer via Benutzername und Passwort verifizieren kann. Im Fall dieser ZAKEKE Implementierung nach Salesforce ist eine Authentifizierung seitens Salesforce nötig, um auf die Dienste des ZAKEKE Produktkonfigurators zugreifen zu können. Dazu wird das Protokoll OAuth (Open Authorization) 2.0 verwendet, welches ein häufig verwendeter Industriestandard für die Authentifizierung ist.

OAuth dient als offenes Standartprotokoll dafür, dass unbefugte Dritte keine Daten abfangen oder für ihre Zwecke missbrauchen können. Der große Vorteil und Nutzen in der Verwendung von OAuth ist, dass der Nutzer eine Vollmacht für die Verwendung seiner Profilinformationen verteilen kann, ohne der autorisierten Anwendung selbst seinen Benutzernamen und Passwort mitteilen zu müssen. Darüber behält er die volle Kontrolle über seine Daten. Die Vorteile des Verfahrens begründen sich in der hohen Flexibilität und Sicherheit des Protokolls. Es gibt verschiedene Rollen im Genehmigungsprozess, die von unterschiedlichen Parteien übernommen werden. Als „Client“ wird die Anwendung bezeichnet, die sich authentifizieren will, um auf die geschützten Daten zuzugreifen. „Authorization Server“ ist der Server, der bei gültigen Zugangsdaten einen Access-Token (Zugriffstoken für einen von ihm definierten Anwendungsbereich ausstellt. Die geschützten Daten sind auf dem „Resource Server“ sicher gespeichert und erfordern einen Zugriffstoken, damit auf sie zugegriffen werden kann. Der Prozess einer OAuth Authentifizierung ist in mehrere Schritte und Abfragen unterteilt. Zu Beginn gewährt der Besitzer der geschützten Daten einem Client via Benutzername und Passwort Zugriff auf seine Daten. Dieser Client nutzt diese Zugangsdaten, um bei dem Authorization Server einen Access-Token anzufragen. Daraufhin validiert dieser Server die Zugangsdaten und sendet dem Client bei Erfolg einen zeitlich begrenzten Zugriffstoken zurück. Solange dieser noch nicht ausgelaufen ist, kann der Client nun für jede seiner Anfragen an die auf dem Resource Server gespeicherten Daten seinen Access-Token verwenden und sich darüber authentifizieren.[3]

[3] [ION17].

3 Implementierung

3.1 Logik zur Authentifizierung

Unabhängig davon, an welcher Stelle das Salesforce System eine Anfrage an die ZAKEKE Dienste sendet, muss jedes Mal der Access-Token zur Authentifizierung mitgesendet werden. Um also mit der Implementierung zu beginnen, muss zunächst eine Genehmigungslogik stehen. Entsprechend des OAuth Protokolls ist der erste Schritt, eine Anfrage an die ZAKEKE Authentifizierungsserver zu senden und einen Token anzufragen. Die Anfrage wird via REST (Representational State Transfer) gesendet und beinhaltet die Client Id, das Secret, sowie die Salesforce Id des aktuellen Benutzers in Salesforce. Dazu werden zusätzliche Informationen wie der Access-Type (Zugriffstyp) und der Grant-Type (Genehmigungstyp) an den Authentifizierungsserver gesendet. Das Resultat ist eine JSON (JavaScript Object Notation) mit drei relevanten Informationen: „access-token“, welcher dem Zugriffstoken entspricht, „token_type“, die Art des Tokens und „expires_in“, einer Information über das Ablaufdatum des Tokens, ab welchem dieser ungültig wird. Das Ablaufdatum ist in Sekunden ab der Ankunft des Tokens angegeben. Diese von Salesforce bereitgestellten Daten werden verarbeitet und entsprechend in Salesforce gespeichert.

In der ZAKEKE Dokumentation wird statt Benutzername und Passwort von „Client Id“ und „Secret“ gesprochen. Damit die ZAKEKE Zugangsdaten im Salesforce Backend hinterlegt sind und aktualisiert werden können, wird ein NamedCredential (Anmeldeinformationen mit Namen) in Salesforce angelegt, in welches übersichtlich und nutzerfreundlich sowohl die beiden Zugangsdaten Client Id und Secret, als auch die URL des ZAKEKE Authentifizierungsservers gespeichert werden können. Auf diese Weise kann der Kunde seine Daten einfach in einer von Salesforce geschaffenen Benutzeroberfläche ändern, ohne direkte Änderungen am Quellcode der Schnittstelle durchführen zu müssen.

Mithilfe dieser Logik ist es der Salesforce Organisation nun möglich, stetig auf den Access-Token zuzugreifen und bei Bedarf nach Ablauf des vorherigen Tokens einen neuen vom ZAKEKE Server anzufragen. Dies übernimmt die Authentifizierung für kommende REST Anfragen oder andere Schnittstellen des ZAKEKE Systems.

Abbildung 1: Benutzeroberfläche für die ZAKEKE Zugangsdaten

```
@AuraEnabled(cacheable=false)
public static String getNewToken(Boolean includeCustomerCode){

    HttpRequest req = new HttpRequest();


    req.setEndpoint('callout:ZakekeToken/token');
    req.setMethod('POST');
    req.setHeader('Content-Type', 'application/x-www-form-urlencoded');

    String body;
    if (includeCustomerCode == true) {
        body = 'grant_type=client_credentials&access_type=S2S&visitorcode=' + UserInfo.getUserId() + '&customercode=' + UserInfo.getUserId();
    } else {
        body = 'grant_type=client_credentials&access_type=S2S';
    }
    req.setBody(body);

    Http http = new Http();
    HTTPResponse res = http.send(req);
    String responseBody = res.getBody();
    Map<String,Object> values = (Map<String,Object>) JSON.deserializeUntyped(responseBody);
    return (String) values.get('access_token');
}
```

Abbildung 2: Quellcode zur Anfrage eines neuen Tokens

3.2 Anpassung des Salesforce Datenmodells und des Checkouts

Sobald ein Kunde ein Produkt mithilfe der ZAKEKE Anwendung individualisiert, bekommt Salesforce eine DesignID zugesand. Diese ist der Schlüssel, der nötig ist, um auf sämtliche Informationen zum Design zuzugreifen. Um diesen Wert für Administratoren des Webshops im Salesforce Backend zu speichern, ist eine Anpassung am Datenmodell notwendig. An den Salesforce Objekten CartItem und OrderItem ist ein neues Feld namens DesignID__c anzulegen. Dabei deutet das __c im Namen darauf hin, dass es sich um ein Benutzerdefiniertes Feld und kein Standardfeld handelt. Zusätzlich wird im OrderItem Objekt noch ein weiteres neues Feld angelegt: DesignFileURL__c. Dieses dient dazu, den später von ZAKEKE empfangenen Link zum erstellten Bild des individualisierten Produktes zu erhalten.

Mit einer neuen Checkbox im Product2 Objekt, welches Produkte in Salesforce darstellt, wird sichergestellt, dass der Knopf zum Konfigurieren auch nur bei individualisierbaren Produkten angezeigt wird. Mit diesen Änderungen können nun alle relevanten Informationen in Salesforce hinterlegt und von den Administratoren des Webshops eingesehen werden.

Im Bestellabwicklungsprozess (Checkout) werde mithilfe eines Salesforce Flows alle sich im Warenkorb befindenden CartItems in OrderItems umgewandelt. Bei diesem Prozess ist eine zusätzliche Logik in den Flow zu integrieren, um später auf die ZAKEKE Informationen zugreifen zu können. Es wird mithilfe einer Apex Action dafür gesorgt, dass das Feld DesignID__c korrekt übernommen und in das richtige OrderItem geschrieben wird. In einer Schleife wird jedes bestehende CartItem solange mit OrderItems verglichen, bis das passende OrderItem gefunden wurde, welches genau den Artikel darstellt. Die Prüfbedingung für die Einheit testet auf übereinstimmende Product2Id, den Einheitspreis und die gewählte Anzahl des Artikels. Dazu gibt es eine zusätzliche Prüfung, ob das OrderItem bereits als passender Eintrag für ein anderes CartItem gefunden wurde, denn auch dann wird es übersprungen. Ist eine Übereinstimmung dieser Daten gefunden, wird die DesignID aus dem CartItem in das OrderItem kopiert. Dieses wird dann zu der Liste mit den gefundenen Übereinstimmungen hinzugefügt und am Ende der Apex Action in der Datenbank aktualisiert. Diese Apex Action wird im Checkout Flow der Salesforce Organisation eingefügt, nachdem die Bestellung (Order) bereits angelegt und aktiviert wurde. An dieser Stelle wird später auch die erstellte Logik zum Platzieren der Bestellung im ZAKEKE System aufgerufen. Geworfene Fehler sind an den Subflow zur Varwaltung von Fehlern zu übermitteln.

3.3 Logik der REST Schnittstellen

Um zwischen den beiden Systemen hin und her kommunizieren zu können, wird die ZAKEKE API (Programmierschnittstelle) verwendet. Diese dient dazu, Informationen zwischen den Plattformen auszutauschen und relevante Daten zu übermitteln. Damit beide Webanwendungen Daten miteinander austauschen können, sind als Schnittstellen sogenannte Webservices nötig, welche an verschiedene Endpunkte Anfragen senden und verarbeiten können. Diese nutzen das Protokoll REST, um einheitlich miteinander zu kommunizieren. „Representational State Transfer“, kurz REST ist international weit verbreitet und zum Industriestandard für die Kopplung verschiedener Systeme geworden. Es wird immer mit dem HTTP (HyperText Transfer Protocol) Protokoll realisiert. Grundsätzlich unterliegt jede Kommunikation via REST folgendem Ablauf: eine HTTP Anfrage (HTTP Request) wird an einen Webservice geschickt.

Dieser verarbeitet sie und sendet eine Antwort (http Response) zurück, welche nun ebenfalls verarbeitet werden kann und nützliche Informationen wie den Status der Anfrage oder möglicherweise aufgetretene Fehler beim Verarbeiten anzeigt.

Jede an das ZAKEKE System gesendete Anfrage muss mithilfe eines gültigen Access-Tokens genehmigt werden. Dafür wird ein HTTP-Header für die Eigenschaft „Authorization“ angelegt, in welchem der generierte Token übergeben wird.

3.3.1 Webservice für Produktinformationen

Dieser Webservice besitzt die einfache Aufgabe, unter gegebenen Produktinformationen einen finalen Preis für den Kauf eines Designs an das ZAKEKE System zu übermitteln. Er bekommt von ZAKEKE eine REST Request mit verschiedenen Informationen wie der Produkt ID, der Menge, der gewählten Variation des Produkte oder dem zu zahlenden Aufpreis für die Individualisierung des Artikels, welche als Variablen für die folgende Berechnung dienen. Die Informationen sind im JSON Format gespeichert und müssen erst deserialisiert werden, um sie verwenden zu können. APEX, die Programmiersprache, welche Salesforce Entwickler nutzen, besitzt bereits in der Standardbibliothek Methoden, die diesen Schritt übernehmen können und aus der JSON eine Map gestalten, welche dann den einfachen Zugriff auf die Informationen ermöglicht.

Hier wird der Salesforce Plattform nochmal individuell die Möglichkeit gegeben, eigene Anpassungen an der Preislogik durchzuführen, bevor der finale Wert dann als HTTP Response zurückgesendet werden soll. Dieser wird nach der Berechnung ebenfalls im JSON Format in den Body der Response angehängt.

```
{
    "productid": "productid",
    "quantity": 1,
    "selectedattributes":
       [{"attrKey1":"attrValue1"},...,
       {"attrKeyn":"attrValuen"}],
    "zakekeprice": 10.5,
    "zakekepercentageprice": 10,
    "additionaldata": {}
}
```

Abbildung 3: Von ZAKEKE gesendete JSON zur Abfrage der Produktinformationen[4]

[4] [Zak20].

3.3.2 Webservice um Produkte zum Warenkorb hinzuzufügen

Bei dieser Schnittstelle sendet ZAKEKE wieder eine HTTP Request an einen selbstgeschriebenen Endpunkt in der Salesforce Organisation. Im Body der Request befindet sich eine JSON mit den neuen Produktinformationen nach dem Konfigurationsprozess. Diese wird wieder deserialisiert und zu einer Map gewandelt, um auf die Informationen zuzugreifen. Um die Daten besser verwalten zu können, werden die Informationen aus der Map in eine Hilfsklasse namens CustomProduct gespeichert, welche Variablen für alle von ZAKEKE übergebenen Angaben besitzt. So müssen Methoden nicht mit unnötig vielen Parametern befüllt werden, sondern verwenden universell eine Datenstruktur, die alle Informationen zu der Individualisierung des Artikels beinhaltet.

src > Zakeke > classes > CustomProduct.cls

```
1   global with sharing class CustomProduct {
2
3       //Provided by internal system
4       global String productID;
5       global Decimal quantity;
6       global Decimal modelUnitPrice;
7       global Decimal designUnitPrice;
8       global Map<String, Object> attributes;
9       global String cartItemId;
10
11      global Map<String, Object> additionalData;
12      /* Content: accountId, webstoreId */
13
14      //Provided by external system
15      global String designID;
16      global String downloadLink;
17
18      global CustomProduct(String newProductId) {
19          this.productID = newProductId;
20          this.attributes = new Map<String, Object>();
21      }
```

Abbildung 4: Quellcode der Hilfsklasse

```
@HttpPost
global static void getRequest(){
    if(RestContext.request != null){
        String requestBody = RestContext.request.requestBody.toString();
        if(String.isNotBlank(requestBody)){
            try {
                Map<String, Object> content = (Map<String, Object>) JSON.deserializeUntyped(requestBody);

                CustomProduct product = new CustomProduct((String) content.get('productid'));
                product.quantity = (Double) content.get('quantity');
                product.designUnitPrice = (Decimal) content.get('zakekeprice');
                product.designID = (String) content.get('designid');
                product.additionalData = (Map<String,Object>) content.get('additionaldata');

                String cartUrl = addToCart(product);
                String responseJSON;
                JSONGenerator gen = JSON.createGenerator(true);
                gen.writeStartObject();
                gen.writeStringField('returnurl', cartUrl);
                gen.writeEndObject();
                responseJSON = gen.getAsString();

                RestContext.response.statusCode = 200;
                RestContext.response.responseBody = Blob.valueOf(responseJSON);
            } catch (Exception e) {
                // 500 Internal Server Error
                RestContext.response.statusCode = 500;
                RestContext.response.responseBody = Blob.valueOf(
                    JSON.serialize(e.getMessage())
                );
            }
```

Abbildung 5: Quellcode zum Empfangen und Deserialisieren der Request

Nachdem eine Instanz der CustomObjekt Klasse angelegt und befüllt ist, wird mit ihr als Parameter eine Methode aufgerufen, deren Aufgabe es ist, das gewünschte Produkt mit allen gegebenen Informationen in den Warenkorb zu legen und somit funktional in die Salesforce B2B Logik zu integrieren. Um dies in Salesforce zu realisieren, wird ein von Salesforce angebotenes Standardobjekt namens „CartItem“ erstellt und in die Datenbank eingefügt. Das Objekt CartItem besitzt Informationen wie die SKU (Stock Keeping Unit) oder die gewünschte Menge des Artikels und wird in den Warenkorb (Cart) des eingeloggten Benutzers eingefügt, in dem ihm bei der Erstellung die ID des Warenkorbs mitgegeben wird, welcher die selbe Account ID besitzt wie der eingeloggte Benutzer. Um an den Warenkorb und Produktdaten zu gelangen, nutzen Salesforce Entwickler SOQL, die Salesforce Object Query Language. Sie ähnelt der SQL (Structured Query Language) Syntax und ermöglicht es mit Filtern nach Einträgen in der Datenbank zu suchen und diese in APEX auszugeben. Die Entscheidung fällt hier bewusst gegen die ConnectAPI, den eigentlichen Standardweg, wenn man versucht Artikel in den Salesforce B2B Warenkorb zu legen, da für die Verwendung der Connect API ein eingeloggter Benutzer notwendig ist. Da die Methode jedoch über eine REST Request ausgelöst wird, welche vom externen ZAKEKE System kommt, greift der Gastbenutzer des Webshops.

Aus Datenschutzgründen wäre es die falsche Wahl, dem ZAKEKE System beim Initialisieren der Konfiguratorenkomponente jedes Mal die Nutzerdaten das aktuell angemeldeten Kunden zu übermitteln und eine komplexe Logik zu entwerfen, die beim Eingang der REST Request diese Daten verarbeitet und einen Login-Versuch startet.

Dennoch ist die ID des Endnutzers nötig, um den Prozess funktional umzusetzen, da der richtige Warenkorb gefunden werden muss, welcher dem Kunden gehört, der das Design erstellt hat. Um das trotz der HTTP Request ermöglichen zu können, kommt ein nützliches Feld in der JSON zum tragen: „additionaldata“. Dies ist ein Feld, welches die Salesforce Plattform bei der Initialisierung des Konfigurators nutzen kann, um Informationen an ZAKEKE zu übermitteln. Später werden wird bei jedem von dem ZAKEKE System angesteuerten Endpunkt eine identische Liste übertragen, welche dann von Salesforce verarbeitet werden kann. Um in Salesforce alle erforderlichen Informationen für die Erstellung eines CartItems bereitzustellen, übermitteln wir in diesem „additionaldata“ Feld die Account ID des eingeloggten Benutzers und die Community ID des eingeloggten Benutzers, welche beschreibt, in welchem Webshop er sich befindet.

```
public static String addToCart(CustomProduct product){
    String accountId = (String) product.additionalData.get('UserAccountId');
    String communityId = (String) product.additionalData.get('CommunityId');
    String userId = (String) product.additionalData.get('UserId');
    WebCart cart = [SELECT Id, AccountId FROM WebCart WHERE Status='Active' AND AccountId=:accountId ORDER BY CreatedDate DESC LIMIT 1];
    CartDeliveryGroup cdp = [SELECT Id FROM CartDeliveryGroup WHERE CartId = :cart.Id LIMIT 1];
    Product2 prod2 = [SELECT Id, Name, StockKeepingUnit FROM Product2 WHERE Id=:product.productId];
    Decimal tempprice = product.quantity * Decimal.valueOf((String) product.additionalData.get('originalUnitPrice'));

    CartItem cartItem = new CartItem(
        Sku=prod2.StockKeepingUnit,
        CartId=cart.Id,
        Product2Id=prod2.Id,
        Quantity=product.quantity,
        Type='Product',
        Name=prod2.Name + ' - designID: ' + product.designID,
        DesignId__c=product.designID,
        CartDeliveryGroupId=cdp.Id,

        ListPrice = tempprice,
        TotalPrice = tempprice,
        UnitAdjustmentAmount = tempprice,
        UnitAdjustedPrice = tempprice
    );

    try {
        insert(cartItem);
    } catch (Exception e) {
        System.debug('RestResource : addToCart : ' + e.getMessage());
        throw new CustomException(e.getMessage());
    }

    String communityURL = URL.getSalesforceBaseUrl().toExternalForm();
    String targetURL = communityURL + '/s' + '/cart/' + cart.Id;
    return targetURL;
}
```

Abbildung 6: Quellcode zur Erstellung eines CartItems

Ist das CartItem einmal erstellt, ist der letzte notwendige Schritt, es in die Datenbank einzupflegen. Um Datenbankeinträge anzulegen oder zu manipulieren, verwendet Salesforce DML (Data Manipulation Language), die Datenmanipulationssprache. Über einen insert – Befehl werden neue Einträge in die Datenbank eingefügt, vorausgesetzt der Benutzer besitzt die Rechte, solche Einträge zu erstellen. Ist dies getan, wird die Individualisierung nun im Warenkorb des Nutzers angelegt und kann fortan im Bestellabwicklungsprozess wie jedes andere Produkt im Warenkorb als Salesforce Standardobjekt verarbeitet werden.

3.3.3 Webservice um Bestellung im ZAKEKE System zu hinterlegen

Bei diesem Webservice liegt der Endpunkt seitens ZAKEKE. Von der Salesforce Plattform muss eine HTTP Request ausgehen, welche die Bestellung im ZAKEKE System platziert. Diese Logik agiert, nachdem der Salesforceinterne Bestellprozess (Checkout) verarbeitet wurde und aus dem Warenkorb (Cart) eine Bestellung (Order) wurde. Der standartmäßig implementierte Salesforce B2B Checkout wandelt während dieses Prozesses alle im Warenkorb befindlichen CartItems (Warenkorbgegenstände) in OrderItems (Bestellungsgegenstände). Hier ist bereits eine wichtige Änderung vorzunehmen: Mithilfe einer Apex Methode wird dafür gesorgt, dass die in den CartItems gespeicherte DesignID ebenfalls in das gleichnamig angelegte Feld DesignID in die OrderItems übernommen wird. Erreicht der Checkout dann sein Ende, wird kurz bevor der Nutzer auf seine Bestellübersicht verwiesen wird, noch die HTTP Request an das ZAKEKE System gesendet. Diese erfordert wichtige Informationen, die wieder im JSON Format gesendet werden müssen. Darunter sind Angaben wie die Bestellnummer, das Datum, der Gesamtpreis und ausführliche Produktinformationen zu jedem individualisierten Gegenstand in der Bestellung, welcher mithilfe des ZAKEKE Konfigurators erstellt wurde.

Die HTTP – Response, die das ZAKEKE System zurücksendet, enthält keinen Body, in welchem Informationen über die Bestellung übermittelt werden, jedoch wird ein HTTP Statuscode mitgesandt, welcher im Salesforce System verarbeitet werden muss. Mit dem Code 200 weiß die Plattform, dass die Bestellung erfolgreich platziert. Bei einem Fehlercode, der in der Spanne von 400 bis 403 liegt, gab es ein Problem bei der Authentifizierung. Das bedeutet, dass der Access-Token ungültig war oder die Kundennummer nicht ordentlich in den Token eingepflegt wurde. In diesem Fall soll das System einen neuen Token generieren und die Anfrage erneut senden. Sollte die Response den Code 500 liefern, so war es dem ZAKEKE System nicht möglich, die Bestellung zu platzieren. Dies lässt entweder auf ungültige IDs schließen oder einer aktuell fehlenden Verfügbarkeit der Systeme von ZAKEKE. Dies kann beispielsweise während Wartungsarbeiten auftreten. In dem Falle wird der Salesforce Bestellprozess abgebrochen.

3.4 Logik für die Konfiguratorenseite

Während die Webservices rein in Apex Code realisiert werden konnten, ist beim Setup der Konfiguratorenseite vorrangig die Programmiersprache JavaScript notwendig. Grundlegend ist es erforderlich, einen HTML (Hypertext Markup Language) Container anzulegen und dann ein auf den ZAKEKE Servern hinterlegtes Script mit einer auf das Produkt angepassten Konfigurationsdatei (Config) anzulegen. Dieses Script erstellt dann ein iFrame, welche auf der Website genau in den Container eingefügt wird und alle Funktionalitäten des ZAKEKE Konfigurators beinhaltet, sodass der Kunde des Webshops in diesem sein Wunschprodukt individualisieren kann.

Diese Anforderungen auf der Salesforce Plattform zu integrieren, ist jedoch komplizierter als im Standardfall, da die Salesforce Cloud viele individuelle Regeln bezüglich Datenschutz und Sicherheit hat. Die grundlegende Architektur im Salesforce Frontend, das Prinzip einer Lightning Web Komponente (Lightning Web Component, LWC) ist aus Sicherheitsgründen darauf ausgelegt, nicht zu wissen, was um sie herum passiert. Eine LWC ist wie ein Baustein für eine Website, der nur seine eigenen Daten kennt und von außen oder von anderen Komponenten nicht verändert werden darf. Jede Lightning Web Komponente besitzt entsprechend der Model-View-Controller (MVC) Architektur sowohl ihr eigenes Frontend Design, als auch ihr eigenes Backend zur Bereitstellung der notwendigen Daten. Die Benutzeroberfläche der Komponente, die den Controller beinhaltet, wird via HTML und CSS (Cascading Style Sheets) realisiert, während der Controller (Steuereinheit) in JavaScript geschrieben wird, welches jedoch ebenfalls auf Apex Code zurückgreifen kann. Hier steht nun jedoch das Problem, dass die Skripte auf dem ZAKEKE System die HTML Struktur manipulieren müssen, um das iFrame des Konfigurators funktional einzufügen. Die Lösung ist umständlich, aber nicht unmöglich. Die URLs zu der ZAKEKE API und zum ZAKEKE Portal werden als vertrauliche Seiten im Salesforce Backend hinterlegt und ihnen werden die Eigenschaften „connect-src“ und „frame-src“ gewährt, damit das Sicherheitssystem von Salesforce die Initialisierung des iFrames nicht verhindert. Nun wird eine HTML – Komponente angelegt, welche im Salesforcestandard ein Feld für HTML Code ist, welcher auf der Seitenstruktur angezeigt wird und nicht wie eine individualisierte LWC abgekapselt ist. Sie bekommt ein später benötigtes Tag mit der Klasse „zakeke-container“. Zum Schluss werden in dem Quelltext der Seite des Salesforce Erfahrungsgenerator (Experience Builder) die beiden Skripte via <script> Tags eingefügt.

Nun wird in den Einstellungen des Erfahrungsgenerators “Allow framing by any page” aktiviert und die Sichercheitsstufe (Security Level) auf “Relaxed CSP: Permit Access to Inline Scripts and Allowed Host” gesetzt. So werden die Skripte der externen Seite trotz der strikten Sicherheitsanforderungen von Salesforce geladen.

Um eine Config für die Initialisierung des iFrames vorzubereiten, benötigt der Controller der Webkomponente verschiedene Methoden, damit er die nötigen Informationen bekommt und diese validieren kann. Dazu wird imperativ aufgerufener Apex Code mit JavaScript Methoden gemischt. Die Konfiguration selbst ist ein JavaScript Object, in welchem Daten über das zu konfigurierende Produkt oder die URLs der entstandenen REST Endpunkte gespeichert werden. Unter Anderem wird auch die vorbereitete Methode zum Authentifizieren aufgerufen, um einen funktionalen Access-Token zu generieren. Ist die Variable für die Config erstellt, wird mit ihr das in den Head des Seitenquelltextes eingebundene Script aufgerufen. Mit validen Daten erstellt das ZAKEKE System nun das iFrame und fügt es als Kindelement an die mit „zakeke-container“ markierte Stelle im HTML Code.

3.5 Qualitätssicherung durch Unit Tests

Um einen ordnungsgemäßen Ablauf der Integrationslogik zu garantieren, muss der Code für möglichst viele, unterschiedliche Anwendungsfälle getestet werden. Die Funktionalität wird mithilfe von Unit Tests auf Extremwerte geprüft, um sicherzustellen, dass nach dem Deployment auf die Produktionsumgebung im Bestellprozess der Kunden keine plötzlichen Fehler auftreten, welche die Kauferfahrung beeinträchtigen oder behindern können. In Salesforce wird Apex Code getestet, in dem Testklassen angelegt werden, welche die Annotation @isTest besitzen. Alle erstellten Testklassen werden von Salesforce aufgerufen und geprüft, wenn ein Entwickler versucht, Code auf die Produktivumgebung zu deployen. Ist dann keine Codeabdeckung von mindestens 75% getestet, scheitert der Deploymentprozess. Diese von Salesforce festgelegte Vorraussetzung ist notwendig, da Salesforce eine Cloud Anwendung ist und sichergegangen werden soll, dass die Änderungen bei einem Update des Produktivsystems auch ordnungsgemäß funktionieren sollen. Dementsprechend sind Unit Tests für jede erstellte Apex Klasse notwendig, um die geforderte Testabdeckung zu erreichen.

Eine Testklasse besitzt eine mit @testSetup gekennzeichnete Methode, die beim Durchlauf der Tests vor den anderen Testmethoden aufgerufen wird. Sie soll dazu dienen, Testdaten für den Test zu erstellen, da ein Apex Test im Normalfall nicht auf Einträge der Datenbank zugreifen kann. Alle anderen Methoden sind mit @isTest annotiert und besitzen zahlreiche assert() Aufrufe, welche die Korrektheit der im Test gewonnenen Ergebnisse mit den geforderten Ergebnissen abgleicht.

Da die entstandene Logik sowohl eingehende als auch ausgehende HTTP Anfragen besitzt, ist es für das Schreiben der Unit Tests notwendig, sogenannte Mocks anzulegen. Ein Mock simuliert gefälschte HTTP Callouts, um der Testanwendung sichergestellt exakt gewünschte Musterdaten zu geben, während der Unit Test denkt, er hätte eine normale HTTP Anfrage gesendet.

4 Analyse der Lösung

4.1 Zusammenfassung und Auswertung

Alle benötigten Schnittstellen der beiden Systeme können nun miteinander kommunizieren. Ein Kunde des Webshops begegnet der entstandenen Logik zuerst auf der Produktdetailseite eines Produktes, welches als konfigurierbar markiert ist. Dort befindet sich eine neue Schaltfläche mit der Aufschrift „individualisieren“, welche den Nutzer auf die erstellte Konfiguratorenseite umleitet und die relevanten Produktinformationen für die Initialisierung des Konfigurators übermittelt. Auf der Produktdetailseite öffnet sich dann der ZAKEKE Konfigurator und der Kunde kann das gewünschte Produkt mit Schrift und Bildern individualisiert auf sich anpassen. Danach besitzt er die Möglichkeit, ein Vorschaubild seines Designs zu betrachten oder den konfigurierten Artikel direkt mit einem Klick auf „In den Warenkorb“ seinem Warenkorb anzuhängen. Besitzt der Kunde alle Produkte, die er sucht, im Warenkorb, so kann er mit einem Klick auf „zur Kasse“. Daraufhin wird der normale Salesforce Bestellabwicklungsprozess durchlaufen, bei welchem der Kunde Lieferadresse, Rechnungsadresse und Versandart auswählt und dann mit einem Klick auf „Jetzt zahlungspflichtig bestellen“ die Bestellung abschließt. Der Nutzer bekommt eine Bestellübersicht, in welcher er unter Anderem nun auch die ID seiner Designs und den Link zu den für ihn generierten Bildern findet. Die Administratoren des Webshops können auf diese Informationen ebenfalls zugreifen und mit dem personalisierten Druck nach Vorlage der Bilder beginnen. Zusammengefasst sind alle im Vorhinein festgelegten Anforderungen umgesetzt wurden, wie Tabelle 2 zu entnehmen ist.

Anforderung	**Beschreibung**
Datenmodellerweiterung der Datenbank	Die Felder **IsCustomizeable**, **DesignId** und **DesignFilesUrl** wurden den Salesforce Standardobjekten **CartItem**, **OrderItem** und **Product** hinzugefügt.
Authentifizierungslogik	Das notwendige OAuth Verfahren wurde in Salesforce implementiert und übernimmt die notwendigen Genehmigungsprozesse.

Visuelle Benutzeroberflächen	Mithilfe von zwei Lightning Web Components kann der Benutzer nun auf der Shopseite den Konfigurator bedienen.
Backend Logik	Alle erforderlichen Endpunkte können vom Zakeke System angesteuert werden. Die empfangenen Daten werden in den Bestellprozess eingebunden und erscheinen nach erfolgreichen Bestellabschluss in der Übersicht.

Tabelle 2: Auswertung technischer Anforderungen

4.2 Wiederverwendbarkeit der gewonnenen Erkenntnisse

Die in der Durchführung gewonnenen Erkenntnisse und Erfahrung für die Integration des ZAKEKE Produktkonfigurators in Salesforce Commerce beinhalten viele theoretische Fakten, die wichtig werden, wenn eine Produktkonfiguratorenlösung gefordert ist und die Entwickler entscheiden müssen, ob eine Drittanbieterlösung wie Beispielsweise die Dienste von ZAKEKE oder eine eigene Implementierung notwendig ist. Informationen über die Limits, Probleme aber auch bedeutende Vorteile von ZAKEKE, sowie Einschränkungen durch den Salesforce Standard, welche gesammelt wurde, geben eine neue Überlegungsgrundlage, um eine Entscheidung besser treffen zu können.

Durch das im Rahmen dieser Projektarbeit angefertigte Proof of Concept ist bei zukünftigen Projekten, die sich mit ähnlichen Integrationen beschäftigen, eine genauere Aufwandsschätzung möglich und viele Konzeptfragen können als Mustervorlage und Anregungen genutzt werden. Selbstverständlich werden kommende Projekte auch eigene Anpassungen vornehmen müssen, um die Lösung funktional in das andere Kundensystem zu integrieren, doch sind viele Aspekte der geschaffenen Schnittstellen generisch und direkt wiederverwendbar angelegt. So kann auf diese Arbeit zukünftig als Dokumentation zugegriffen werden.

4.3 Fazit

4.3.1 Durchführung im Vergleich zum geschätzten Aufwand

Die Umsetzung der Schnittstellenimplementierung ist ein langwieriger Prozess, bei dem auf viele kleine Details zu achten ist. Trotz generischer ZAKEKE Dokumentation allgemeine Integrationskonzepte der API verbergen sich viele Hindernisse und Problemstellungen in der Salesforce Implementierung, welche individuelle Lösungsansätze erfordern. Einige Zeit fließt in die Erstellung von Konzepten und Lösungswegen. Es ist notwendig, vor dem Beginn einen Überblick über das Gesamtbild der Aufgabenstellung zu erlangen. Vom Datenmodell über die benötigten Schnittstellen bis hin zu Anpassungen der Salesforce Standard Automatisierungsprozesse müssen zahlreiche softwarearchitektorischen Entscheidungen bereits vor Beginn der Implementierung getroffen werden. Viele Lösungsansätze davon müssen individualisiert an die Kundenanforderungen und an das bestehende Webshopsystem des Salesforce Kunden angepasst werden, was schnell zu überraschend auftauchenden Problemstellungen führen kann. Ein Beispiel dazu aus der Durchführung wäre folgender Fall:

Das bestehende Salesforce Umfeld nutzt eine benutzerdefinierte Logik, um jeden im Warenkorb befindlichen Artikel vor dem Kauf noch einmal auf die Verfügbarkeit im Lager zu prüfen. Legt ein Kunde nun mehrere konfigurierte Produkte in den Warenkorb, wirft der Prozess Fehler, da dies alles unterschiedliche Artikel sind, die aber auf dem selben Produkt basieren. So sind wieder Anpassungen im Flow zum Prüfen des Lagerbestands notwendig, um ihn auf die Verarbeitung der neuen Funktionalitäten zu erweitern.

Solche spontanen Problematiken sind schwer im Vorhinein abzuschätzen, was den eigentlich erwarteten Aufwand beträchtlich steigen lässt. Dazu kommen spontane Wünsche des Kunden für die Visualisierung, die Fehlerverwaltung oder Änderungen der Preislogik, welche im Nachhinein zur ursprünglichen Kalkulation hinzugefügt werden müssen. Außerdem ist zu beachten, dass bei einem so standardnahen Projekt manchmal das Problem auftreten kann, dass Änderungen an einer Salesforce Komponente notwendig sind, deren Quellcode ein Entwickler jedoch nicht erreichen und verändern kann. In so einem Fall ist eine benutzerdefinierte Umsetzung notwendig, die bis dahin führen kann, eine Standardkomponente selbst nachbauen zu müssen, um den Kundenwunsch zu realisieren.

4.3.2 Mögliche Schwachstellen und Erweiterungsmöglichkeiten

Da das Kundenprojekt sehr Salesforce standardnah ist und wenig Budget für das Selbstschreiben von Shopfunktionalitäten besteht, welche schon im Standard existieren und genutzt werden können, ist die Visualisierung der Funktionalitäten von ZAKEKE stark limitiert. Darstellungen im Warenkorb sind sehr aufwendig, da hier die Standardkomponenten greifen und ein Entwickler hieran nichts ändern kann. So wird nur im Namen des Warenkorbartikels angezeigt, dass das Produkt individualisiert ist, aber es gibt keine übersichtlichen Tags oder Informationen über die genaue DesignID des Artikels. Durch Selbstbauen der Warenkorbkomponente wären viele neue und benutzerfreundlichere Funktionalitäten möglich, wie Beispielsweise ein Knopf neben dem Artikel, um das Design noch einmal vor dem Checkout zu ändern, statt es immer löschen und neu anlegen zu müssen. Auch könnten mithilfe von zusätzlichem Aufwand in der Bestellübersicht mehr Informationen über die Designs dargestellt werden. Manch ein Webshopbetreiber könnte sich eine Übersicht auf der Seite des eingeloggten Nutzers wünschen, auf welcher eine Liste von selbst erstellten Designs angezeigt werden soll, um diese schnell zum Warenkorb hinzufügen zu können. Es gibt viele Möglichkeiten, das Projekt zu erweitern, doch sind diese Abhängig von den Forderungen und den Wünschen des Kunden und mit Mehraufwand verbunden.

4.4 „Lesson Learned“ für zukünftige Integrationen

Das Beste, was für zukünftige Projekte mit API Integrationen mitgenommen werden kann, ist es, nach der Konzeptionierungsphase direkt mit der Implementierung des Authentifizierungsverfahrens zu beginnen. Daten zur Authentifizierung und Informationen über den Nutzen sind während der Programmierung lange als feste Konstanten hard-coded gewesen, was regelmäßige Änderungen an den Daten gefordert hat. Nach der Implementierung der Logik hat sich das System selbst um eigene Token und die richtigen HTTP Header gekümmert, und Änderungen waren zentral an einer Stelle möglich, statt überall aktualisiert werden zu müssen.

5 Abschließende Worte

Diese Projektarbeit ist ein gutes Beispiel dazu, wie zwei vollkommen verschiedene Anwendungen mithilfe einer API miteinander kommunizieren und synergieren können. Es beweist, dass selbst die unterschiedlichsten Systeme miteinander arbeiten können, wenn sich auf einheitliche Schnittstellen geeinigt wird und diese klar kommuniziert werden. Die verwendete Herangehensweise und das entstandene Konzept sind eine von unzählig vielen Möglichkeiten, die Problemstellung zu lösen, doch gerade das zeigt die Vielfalt von dem, was in der heutigen Zeit in der Softwareentwicklung möglich ist. Die Integration ist ein einzigartiges Werk, angepasst auf die Wünsche des Kunden, was in sich selbst ein eigenes kleines System ist und immer weiterentwickelt werden kann. Wie andere Entwickler nun die gewonnenen Erkenntnisse anwenden, ist abhängig von ihren eigenen kreativen Ideen und Interpretationen des Problems. Schnittstellenintegrationen werden immer ein Thema bleiben, welches Softwareentwickler wieder und wieder begleitet. Und dennoch: jedes Mal entsteht eine neue, einzigartige Lösung wie diese.

Literaturverzeichnis

[INZ15] INZTITUT Medienkompetenz: *Software Architektur,* 2015, https://www.inztitut.de/blog/glossar/software-architektur/

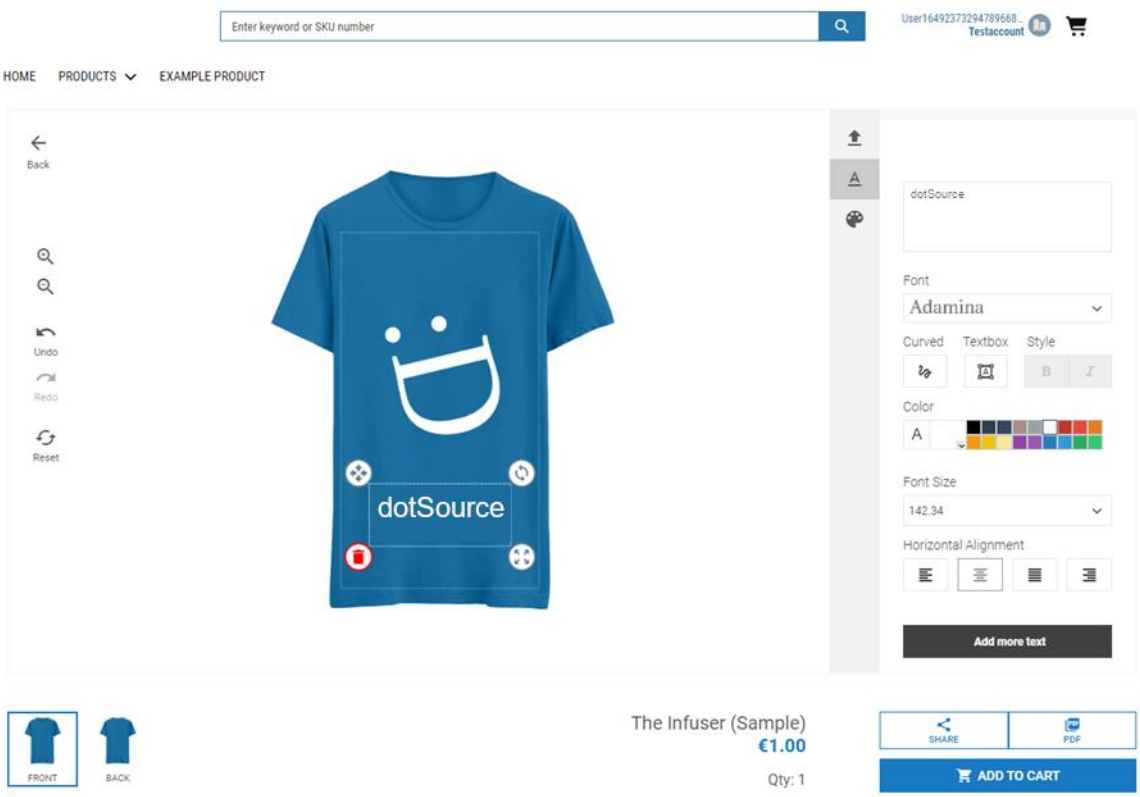
[ION17] IONOS SE: *OAuth (Open Authorization)*, 2017, https://www.ionos.de/digitalguide/server/sicherheit/was-ist-oauth/

[Sal22] salesforce.com, inc.: *Get Started with Salesforce,* 2022, https://resources.docs.salesforce.com/latest/latest/en-us/sfdc/pdf/basics.pdf

[Zak20] Zakeke: *Zakeke Dokumentation, API Integration, Customizer-Seite.*, 2020, https://zakeke.zendesk.com/hc/de-de/articles/360025244873-Customizer-Seite

Anlagenverzeichnis

Anlage 1: Konfiguratorenseite nach der Implementierung



Ehrenwörtliche Erklärung

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Projektarbeit/Studienarbeit/Bachelorarbeit mit dem Thema:

...

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Projektarbeit/Studienarbeit/Bachelorarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift