Inhaltsverzeichnis **Seite**

1 Einleitung

Die Welt des elektronischen Verkaufs steht jeden Tag vor neuen Herausforderungen. Die Bedürfnisse der Konsumenten ändern sich und ihre Anforderungen wachsen, deshalb sollen die digitalen Agenturen mit ihnen Schritt halten. Die Herausforderungen der heutigen Zeit kann Salesforce bewältigen, die 1999 mit dem Ziel gegründet wurde, Unternehmen bei der digitalen Transformation zu unterstützen sowie die Beziehungen zu ihren Kunden aufzubauen und zu verstärken[1].

Salesforce-Produkte sind sowohl für den B2B- als auch für den B2C E-Commerce geeignet. B2C (auf Englisch Businness-to-Consumer) ist ein Verkaufsmodell, bei dem Unternehmen ihre Produkte direkt an Endverbraucher verkaufen. Diese Art des Handels zielt darauf ab, das Verständnis für die Bedürfnisse der Kunden zu verbessern, was durch die Erfassung von Kundendaten über Online-Kanäle erreicht werden kann. Zu diesen können Produktsuchen, Einkaufswagen und Wunschlisten gehören. Diese Daten können dann für Produktempfehlungen genutzt werden, die wiederum per E-Mail verschickt werden können[2]. Andere Dokumente können auch per E-Mail übermittelt werden. Beispiele sind Rechnungen, Rückgabe- und Umtauschquittungen, Gutscheine mit Sonderangeboten und andere. Hier kommt die PDF-Generierung ins Spiel. Einer der wichtigsten Vorteile der Verwendung von PDF-Dokumenten ist die Unabhängigkeit vom Endgerät oder Betriebssystem, da Schriften und Layouts im Dokument eingefügt sind. Dadurch wird sichergestellt, dass die Dokumente aktuell und korrekt sind, und das Risiko von Darstellungsfehlern aufgrund von unterschiedlichen Ausgabesystemen verringert.

Ziel der Arbeit ist es, die Methoden der PDF-Generierung in SFCC B2C zu untersuchen, sie zu vergleichen und einen Prototyp zu erstellen. Das Unternehmen dotSource wendet bei seiner Arbeit aktiv agile Methoden an, weshalb eine aktive Kommunikation mit dem Kunden während der Entwicklung des Shops wesentlich ist. Der Entwickler sollte wissen, wie ein PDF-Dokument generiert werden kann, und den Kunden die wichtigsten Methoden erklären, die dafür verwendet werden können. Die Vor- und Nachteile eines bestimmten Verfahrens sollten von dem Entwickler erläutert werden, damit der Kunde sich für die am besten geeignete Lösung für sein Unternehmen entscheiden kann.

[1] Vgl. [Sal25 a]

[2] Vgl. [Sal25 b]

2 Generierungsfunktionen auf externer Umgebung

Eine der Methoden zur PDF-Generierung in SFCC B2C ist die Erstellung einer Generierungsfunktion auf externen Umgebung (z.B. Cloudhosting oder Heroku) als Custom Implementierung, die per API an SFCC angebunden wird.

2.1 Web-Service Framework

Salesforce B2C Commerce bietet ein Web Service Framework, das den stabilen, sicheren und effektiven Betrieb des Applikationsservers unterstützt. Mit dem Web Service Framework gibt es Gelegenheiten, alle Webservices und deren Aufrufe zu verwalten, Limits für die Häufigkeit der Aufrufe und die Anzahl der Fehlversuche in einem bestimmten Zeitraum festzulegen. Dazu kommt noch, dass Dienste aktiviert oder deaktiviert werden können. Das Web Service Framework bietet auch Möglichkeiten, Serveranalysen zu erstellen, den Aufrufstatus und Fehler anzuzeigen sowie die Serverleistung zu untersuchen[3].

Für das Erstellen von Web-Services in SFCC B2C sollten folgende Schritte durchgeführt werden. Zunächst soll der Web-Service im Business Manager konfiguriert werden. Danach folgen die Erstellung und das Testing mit der Hilfe von Logging und Fehlerbehebung des Codes[4] (siehe Anlage 1).

2.1.1 Web Service Konfiguration im Business Manager

Der Business Manager ist eine Kommandozentrale, der für Merchandising, Verwaltung und Website-Entwicklung im B2C Bereich geeignet ist. Er kann sowohl für eine als auch für mehrere Websites verwendet werden. Zu seinen hauptsächlichen Registerkarten gehören „Merchant Tools“ (die am häufigsten von den Händlern genutzt wird) und „Administration“ (die von den Administratoren und Entwicklern genutzt wird). Mit Hilfe dieser Kommandozentrale richten Merchandiser die Konfiguration der Site. Administratoren können Websiteeinstellungen anpassen, Websitedaten importieren sowie exportieren, Code- und Datenänderungen umsetzen. Entwickler sind in der Lage, Websites zu erstellen, die Codeversion zu verändern, Fehler zu beheben, den Seitencache anzupassen und vieles mehr[5].

[3] Vgl. [Tra25 a]

[4] Vgl. [Sal25 c]

[5] Vgl. [Tra25 e]

Der Service kann im Business Manager angelegt werden. Dazu muss zuerst ein Web Service Profil erstellt und Credentials hinzugefügt werden. Sie befinden sich auf der Registerkarte „Services“, die durch Auswahl des Moduls „Operations“ im Fenster „Administration“ aufgerufen werden kann. Credential wird für die Identifizierung verwendet. Für das Hinzufügen von Credentials sind Name, URL zum Service, Nutzer und Kennwort benötigt. Der Name sollte gemäß SFCC-Standard angegeben werden, d.h. er darf keine Benutzerinformationen und keine Leerzeichen enthalten und am besten sollte der Name so ausgewählt werden, damit Details zur Verwendung entnommen werden können. Zum Beispiel kann der Name einen Protokolltyp, einen Sitename und einen Servicename beinhalten. Da SFCC auf die Datensicherheit bedacht ist, wird das eingegebene Passwort nach der Übernahme der Änderungen maskiert und kann nicht über den Business Manager abgerufen werden[6].

Der nächste Schritt besteht darin, ein Serviceprofil zu erstellen, das nicht nur für einen , sondern auch für mehrere Services verwendet werden kann (wenn deren Konfiguration identisch ist). Die wichtigsten Merkmale des Dienstprofils, die im Business Manager eingestellt werden können, sind Timeout, Cirkuit Breaker und Rate Limiter[7].

Timeout bezieht sich auf die maximale Zeit in Millisekunden, die der Server auf Antwort vom Web-Service wartet. Sie umfasst sowohl die Verbindungszeitüberschreitung als auch die Socket-Zeitüberschreitung[8]. Socket ist ein Endpunkt der Kommunikation zwischen zwei laufenden Programmen in einem Netzwerk, der häufig in Client-Server-Programmen verwendet wird. Der Server erstellt einen Socket, verbindet ihn mit einer Netzwerkanschlussadresse und wartet auf eine Anfrage des Clients. Im Gegenzug erstellt der Client ebenfalls einen Socket und versucht, sich mit dem Server-Socket zu verbinden. Wenn die Verbindung erfolgreich ist, beginnt die Datenübertragung[9]. Wenn B2C Commerce innerhalb der angegebenen Wartezeit keine Antwort vom Server erhält, wird der Vorgang mit einem Fehler abgebrochen. Timeout kann mithilfe von API-Methoden eingestellt werden, aber Salesforce empfiehlt, die Zeitüberschreitung in der Serverkonfiguration festzulegen, um später Serveranalysen anzeigen zu können[10].

[6] Vgl. [Tra25 b]

[7] Vgl. [Sal25 c]

[8] Ebenda

[9] Vgl. [Gee25]

[10] Vgl. [Sal25 c]

Der Rate Limiter legt die maximale Anzahl der Anfragen an den Server für eine bestimmte Zeitspanne fest. Wenn die Anzahl der Anfragen den ausgewählten Wert überschreitet, wird von Salesforce B2C eine ServiceUnavailableException gesendet[11].

Der Cirkuit Breaker wiederum prüft die Anzahl der erfolglosen Anfragen an den Server während eines bestimmten Zeitintervalls. Übersteigt die Anzahl den eingestellten Grenzwert, so gibt SFCC eine ServiceUnavailableException aus und bricht den Service-Call ab. Cirkuit Breaker kann auch Benachrichtigungen über Probleme im Zusammenhang mit dem Aufruf von Webdiensten empfangen, wie z. B. HTTP 500 (Internal Server Error) oder 503 (Service Unavailable) Antwortstatus, unbekannter Host, Verbindungsverweigerung, usw[12].

Der letzte Schritt besteht darin, einen Dienst zu erstellen, der beim Aufruf ein neues ServiceConfig Objekt generiert. Dieses Objekt unterstützt Methoden wie getCredential(), getProfile(), getServiceType() und getID()[13], d.h. es enthält alle zuvor ausgefüllten Informationen über den Service. Service Credentials und Service Profil können in den entsprechenden Feldern im Abschnitt Service des Business Manager ausgewählt werden. Dem Service sollte auch ein beschreibender Name zugewiesen werden. Salesforce empfiehlt, den Namen in diesem Format zu schreiben: {cartridge}.{protocol}.{service}.{operation}. Diese Namenskonvention hilft dabei, die Hauptmerkmale dieses Servers und den Zweck seiner Verwendung zu verstehen. Außerdem kann die Angabe der Cartridge (mehr dazu siehe Seite 12) bei der Verwendung verschiedener Cartridges im selben Projekt nützlich sein[14].

Ein wichtiges Merkmal des Servers ist sein Typ, der im Business Manager ausgewählt werden muss. SFCC bietet folgende Arten von Services: HTTP, HTTP-Form, FTP, SFTP, SOAP und Generic. Zusätzlich zu den bereits erhaltenen Methoden erbt das Objekt die Methoden der Basisklasse des ausgewählten Services[15].

[11] Vgl. [Sal25 c]

[12] Vgl. ebenda

[13] Vgl. [Sal25 d]

[14] Vgl. [Sal25 c]

[15] Ebenda

Tabelle 1: Arten von Services

Quelle: [Sal25 c]

Typ	Klasse
HTTP	dw.net.HTTPClient.
HTTP Form	dw.net.HTTPClient.
FTP	dw.net.FTPClient
SFTP	dw.net.SFTPClient
SOAP	dw.ws.webReference2
Generic	Keine Klasse

Im Business Manager muss das Feld „Enabled“ ausgewählt werden, damit der Service aufgerufen werden kann. SFCC unterstützt zwei Modi für die Bedienung von Webdiensten - Live und Mocked. Im ersten Modus wird ein echter, laufender Dienst aufgerufen und eine echte Antwort auf die Anfrage erhalten. Im zweiten Fall wird ein Anruf durchgeführt, der eine simulierte Anfrage an den Server sendet und eine vordefinierte Antwort zurück liefert. Dazu kommt noch, dass Anfragen und Antworten vom Web-Service geloggt werden können[16].

2.1.2 Erstellen des Webservice Scripts

Zunächst sollte ein Objekt vom Typ Service mit der Methode LocalServiceRegistry.createService, die aus dem Paket dw.svc importiert ist, erstellt werden. Diese Methode nimmt zwei Argumente entgegen, nämlich die ID des im Business Manager konfigurierten Dienstes und die Callback-Handler. Jeder Callback besteht aus einer Reihe von Methoden, die je nach dem Stand der Anfrageverarbeitung in der folgenden Reihenfolge ausgeführt werden: initServiceClient(Service), createRequest(Service, Object...), execute(Service, Object) und parseResponse(Service, Object). Die Beschreibung und Verwendung in verschiedenen Arten von Webdiensten sind in der Tabelle 2 dargestellt[17].

Tabelle 2: Callback Methoden für verschiedene Arten von Web-Services

Quelle: [Sal25 c]

Callback-Methoden	Beschreibung	HTTP und HTTP Form Callbacks	FTP und SFTP Callbacks	SOAP Callbacks	Generic Callbacks
initServiceClient()	Erstellt Underlying Client, der für	Nicht erforderlich	Nicht erforderlich	Gibt ein dw.ws.port und	Nicht erforderlich

[16] Vgl. [Sal25 c]

[17] Vgl. Ebenda

	den Anruf verwendet wird			webreferences2 Objekt zurück	
createRequest()	Konfiguriert eine Anfrage für den Service, z.B. Hinzufügen eines Headers oder Bodys zu einer Anfrage	Erforderlich zum Erstellen requestData Objekt. Dieses Objekt wird verwendet, um den Service aufzurufen	Dasselbe zu HTTP und HTTP Form Callbacks	Erforderlich zum Erstellen requestData Objekt. Dieses Objekt muss an die execute() Methode übergeben werden.	Dasselbe zu HTTP und HTTP Form Callbacks
execute()	Führt eine Anfrage an den Service aus	Nicht erforderlich	Wenn die setOperation-Methode nicht im Skript verwendet wird, werden die Operationen im execute() Callback definiert und beim Aufruf eines Webdienstes empfangen	Wird für die zusätzliche Verarbeitung der Anfrage des Services verwendet	Nicht erforderlich
parseResponse()	Konvertiert das Ergebnis eines Serviceaufrufs in ein Objekt	Ermöglicht die Verarbeitung der Antwort des Servers (HTTPClient Klass)	Ermöglicht die Verarbeitung der Antwort des Servers (FTPClient Klass)	Ermöglicht die Verarbeitung der Antwort des Servers	Dasselbe zu SOAP Callbacks

Mock Callbacks können auf zwei Wegen durchgeführt werden. Die erste Möglichkeit besteht darin, ein Mock-Modul im Business Manager auszuwählen und Code zu schreiben, um den Aufruf zu verarbeiten. Die zweite Variante ist es, die Simulation des Aufrufs in einem Skript zu erzwingen[18].

Dann wird der Web-Service mit der Methode Service.call() aufgerufen. Es ist möglich, dieser Funktion Argumente hinzuzufügen, um für den Webdienst erforderlichen Daten zu senden. Wenn Timeout oder Rate Limiter nicht ausgelöst werden, gibt Callback das Ergebnis des Dienstes zurück, dass in der Klasse Result gespeichert wird. Andernfalls sendet SFCC den entsprechenden Fehlercode[19].

[18] Vgl. [Sal25 c]

[19] Ebenda

2.1.3 Sicherheit von Web Services

Mit der Hilfe von PDF-Generierung ist es möglich, verschiedene Arten von Dokumenten zu erstellen, von denen einige ein hohes Maß an Datensicherheit erfordern. Die 1977 erschaffene Open Systems Interconnection (OSI) Architektur beschreibt die einzelnen Aufgaben, die auf jeder der sieben Schichten ausgeführt werden, um die Netzwerkkommunikation zu gewährleisten. Für die Sicherstellung der Kommunikation zwischen einem Dienst eines Drittanbieters und Business Manager spielen 2 Schichten eine zentrale Rolle, nämlich Transport- und Anwendungsschicht. Die folgenden Prozesse finden auf der Transportschicht statt:

- Bei dem Aufruf eines Webdienstes über das HTTPS Protokoll verwendet B2C Commerce automatisch den privaten Schlüssel des Clients, der in Business Manager gespeichert ist. Dazu wird der Hostname verwendet, unter dem der Schlüssel abgelegt ist;
- Import von SSL-Zertifikaten für die Zwei-Faktor-Authentifizierung in die Variable, in der der Schlüssel gespeichert wird;
- Verwendung eines TSL-Zertifikats für die Kommunikation zwischen dem Webservice und B2C Commerce;
- Herstellung einer Verbindung zwischen B2C Commerce und dem Drittanbieterdienst[20].

Nachdem der Kommunikationskanal auf der Transportschicht etabliert ist, folgt die Kommunikation auf der Anwendungsschicht, bei der die folgenden Prozesse ablaufen:

- Speicherung von Zertifikaten in einer Cartridge zur automatischen Verschlüsselung oder Entschlüsselung von SOAP-Nachrichten;
- Senden einer verschlüsselten oder signierten SOAP Nachricht an einen Webdienst eines Dritten. Die Verschlüsselung oder Signierung der Nachricht wird mit einem in der Cartridge gespeicherten X509-Zertifikat durchgeführt[21].

[20] Vgl. [Tra25 c]

[21] Vgl. [Tra25 c]

2.2 REST API

„Nach [IBM25 a] ist eine REST-API eine Anwendungsprogrammierschnittstelle (API), die den Designprinzipien des REST-Architekturstils (Representational State Transfer) entspricht“. Zu diesen Prinzipien gehören:

- Einheitliche Schnittstelle (API-Anfragen für dieselbe Ressource sollten gleich aussehen und dieselben Daten enthalten, die nur über einen URI (Uniform Resource Identifier) verfügbar sind.)
- Client-Server-Entkopplung (der Client und der Server sollten keine Informationen übereinander enthalten, d. h. sie sollten völlig unabhängig voneinander sein. Die einzige Information, die der Server enthalten sollte, ist die URI der Ressource)
- Zustandslosigkeit (jede API-Anfrage müsste alle erforderlichen Informationen für ihre Bearbeitung enthalten. Der Server sollte keine zusätzlichen oder gespeicherten Informationen enthalten, um die Anfrage zu bearbeiten)
- Cachefähigkeit (für die Verbesserung der Leistung auf der Client-Seite und die Erhöhung der Skalierbarkeit auf der Server-Seite sollten die Ressourcen auf einer der beiden Seiten gecacht werden)
- Mehrschichtige Systemarchitektur (API-Anfragen und -Antworten werden auf verschiedenen Ebenen der Architektur gestellt, sodass der Client nicht sieht, mit wem der Server interagiert und umgekehrt)[22]

REST API verwendet HTTP-Anfragen für den Datenaustausch. Zu den wichtigsten HTTP-Methoden gehören GET (Abrufen eines Datensatzes), POST (Erstellen eines neuen Datensatzes), PUT (Aktualisieren eines Datensatzes) und DELETE (Löschen eines Datensatzes)[23].

2.3 Cloud Computing und Cloudhosting

Für die Erleichterung des Prozesses der Erstellung einer Webanwendung kann Cloud Computing hilfreich sein. Gemäß der sogenannten NIST-Definition ist Cloud Computing einen „allgegenwärtigen, bequemen, bedarfsgerechten Netzwerkzugriff auf einen gemeinsamen Pool konfigurierbarer Recherressourcen, die schnell und mit minimalen Verwaltungsaufwand

[22] Vgl. [IBM25 a]

[23] Vgl. ebenda

oder Interaktion mit Service Providern bereitgestellt, aber auch wieder freigegeben werden können“ ([Kra22], S. 11). Außerdem hat NIST ein Modell des Cloud Computings bereitgestellt, das in 3 Hauptkomponenten unterteilt ist: Service Merkmale, Service Modelle und Deployment Modelle (siehe Anlage 2)[24].

Es gibt 5 Hauptmerkmale des Cloud Computing: On-Demand Self-Service, Netwerkzugriff, Elastizität, Messung der Ressourcennutzung und Ressourcen-Pooling.

Tabelle 3: Hauptmerkmale von Cloud Computing
Quelle: [Kra22], S. 13

Service Merkmale	Beschreibung
On-Demand Self-Service	Nutzer ist in der Lage, automatisch auf Ressourcen zuzugreifen, ohne dass eine zusätzliche Kommunikation mit dem Cloud-Service-Provider notwendig ist
Netwerkzugriff	Der Zugang zu den Ressourcen erfolgt über öffentliche Netze mit standardisierten Internetprotokollen
Elastizität	Ressourcen können jederzeit schnell zugewiesen und ebenso schnell wieder freigegeben werden, und zwar in der Menge, die Benutzer benötigt
Messung der Ressourcennutzung	Cloud-Systeme überwachen automatisch die Nutzung von Ressourcen, indem sie ihren Verbrauch messen (z. B. Speicherverbrauch, Verarbeitungszyklen)
Ressourcen-Pooling	Damit der Cloud-Service-Provider mehrere Nutzer in einem Multi-Tenant-Modell bedienen kann, sollten die Nutzerressourcen zunächst gepoolt werden. Außerdem werden dem Benutzer physische und virtuelle Ressourcen dynamisch zugewiesen, sodass er oft keine Kontrolle oder Kenntnis über den Standort der Ressourcen hat.

Eines der Service-Modelle des Cloud Computings ist Infrastructure as a Service (IaaS), bei dem ein Provider physische und virtuelle Hardware anbietet. Zu dieser Hardware gehören Server, Speicher und Netzwerkinfrastruktur, die der Nutzer über eine Self-Service-Schnittstelle verwalten kann. Dieses Modell bietet dem Nutzer schnellen Zugang zu Verarbeitungs-, Speicher-, Netzwerk- und anderen grundlegenden Rechenressourcen. Dies ermöglicht es dem Kunden, Betriebssysteme und Anwendungen unabhängig voneinander einzusetzen und auszuführen. Der Kunde hat zwar die Kontrolle über die Anwendung, den Speicherplatz und in einigen Fällen auch in begrenztem Umfang über bestimmte Netzkomponenten, jedoch in der Regel nicht über die zugrunde liegende Cloud-Infrastruktur[25].

[24] Vgl. [Kra22], S. 12

[25] Vgl. [Kra22], S. 15

Das IaaS Modell wird häufig für das Cloudhosting verwendet. Cloudhosting basiert auf einem Netz von miteinander verbundenen physischen und virtuellen Servern, die Flexibilität und Skalierbarkeit bieten. Es nutzt ebenfalls die Technik der Virtualisierung, bei der die Elemente eines Computers, wie Prozessor, Arbeitsspeicher und Speicher, auf einer abstrakten Ebene auf mehrere virtuelle Maschinen verteilt werden. Der Hauptvorteil der Virtualisierung bei dem Cloudhosting ist die effiziente Nutzung der Ressourcen[26].

2.4 Heroku

Heroku ist eine Cloud Platform, mit der Webanwendungen erstellt, bereitgestellt und überwacht werden können. Die Anzahl der Anwendungen, die auf Heroku laufen, nimmt täglich zu. Daher ist es nicht verwunderlich, dass die Plattform eine bestimmte Architektur haben muss, um viele gleichzeitig laufende Programme zu unterstützen. Heroku ist in mehrere Hauptsegmente unterteilt, dazu gehören Router, Dynos, sowie zusätzliche Elemente wie Logplex undverschiedene andere Anwendungen und Dienste[27] (siehe Anlage 3).

Dynos ist ein Container, der eine Umgebung für die Ausführung einer Anwendung bereitstellt. Jeder Dynos ist von den anderen isoliert und bereitet nur seine eigene Anwendung vor. Routers sind ein Teil der Heroku-Software, die für die Kommunikation zwischen dem Benutzer, der den Code ausführt, und den Dynos verwendet werden. Sie enthalten Informationen über alle bereitgestellten Anwendungen und ihre auf der Plattform gespeicherten Adressen, sowie die entsprechenden externen URLs. Logplex ist ein Tool, das einen konstanten Datenstrom von verschiedenen Heroku-Segmenten (Router, Dynos, etc.) an einem Ort gleichzeitig bereitstellt[28]

[26] Vgl. [IBM25 b]

[27] Vgl. [MS13]

[28] Vgl. ebenda

3 Externe Service für die PDF-Generierung

Die zweite Methode der PDF-Generierung besteht in der Nutzung eines Drittanbieterdienstes, der über eine API mit SFCC B2C verbunden wird. Das Funktionsprinzip dieser Methode ist ähnlich wie das oben beschriebene, aber anstatt selbst eine Anwendung zur PDF-Generierung zu erstellen, ist ein vorgefertigtes Programm verwendet. Das Problem der Generierung von PDF-Dateien ist weit verbreitet, weshalb verschiedene Anbieter dieses Dienstes im Internet gefunden werden können.

Ein Beispiel ist die Plattform PDF Generator API, mit der PDF-Dokumente auf einfache Weise erstellet werden können. Ein Merkmal dieser Plattform ist die Verwendung eines browserbasierten Editors, mit dem das Erscheinungsbild eines Dokuments durch Verschieben und Platzieren der Elemente selbst gestaltet werden kann. Das spart Zeit und Geld bei der Erstellung einer Vorlage. Der Dienst kann für Dokumente wie Rechnungen, Lieferscheine, Vertragsunterlagen oder Etiketten genutzt werden[29]. Die Plattform bietet eine ausführliche Dokumentation mit Beispielen, Anleitungen und Tutorials.

Der Dienst PDF-Generator API basiert auf einer Web API-Architektur, die es ermöglicht, ihn in einer Vielzahl von Programmiersprachen (z. B. JavaScript, Python, Ruby, PHP und Java) zu verwenden. Er unterstützt JSON-Daten und UTF-8 Codierung. Die API verwendet auch Rate Limiter und erlaubt bis zu zwei Anfragen pro Sekunde und bis zu 60 pro Minute zu stellen. Wird diese Grenze überschritten, gibt der Dienst einen Fehler mit dem Code 429 zurück. Um die Sicherheit bei der Authentifizierung zu gewährleisten, verwendet der Dienst JSON Web Tokens (JWT)[30]. Die Plattform bietet eine Reihe von Tarifen, die von der Größe des Unternehmens und seinen Zielen abhängen:

- Enterprise (6990$ pro Jahr);
- Premium (3990$ pro Jahr);
- Basic (990$ pro Jahr);
- Starter (590$ pro Jahr);
- Low Usage (290$ pro Jahr)[31].

[29] Vgl. [PDF25 a]

[30] Vgl. [PDF25 b]

[31] Vgl. [PDF25 c]

4 Integration per Third-Party-Library

Da SFCC B2C keine Standard-Funktion für die PDF-Generierung bietet, ist die Integration einer Third-Party-Lösung eine gute Möglichkeit, diese Aufgabe zu lösen. Dabei handelt es sich um die dritte Methode der PDF-Erstellung - die Anwendungen einer Third-Party-Library, die in SFCC B2C integriert wird, um die Grundfunktionen der Website zu erweitern.

4.1 Cartridges in SFCC

Cartridge ist ein Container, in dem Code platziert und bereitgestellt werden kann. Er wird dazu verwendet, um die Funktionalität einer Website zu erweitern oder mit anderen Drittsystemen zu interagieren. Nach Funktionalität unterscheidet Salesforce „generic“ und „application“ Cartridges. Der Hauptunterschied besteht darin, dass die Funktionalität von „generic“ Cartridges für viele Websites verwendet wird, während „application“ Cartridges für eine Website mit spezifischen Anforderungen verwendet werden. Cartridges können Controllers, Templates, Scripts, Form Definition, statische Inhalte (Bilder, CSS-Dateien und clientseitige JavaScript-Dateien) und WSDL Dateien enthalten[32].

In B2C Commerce existiert der Begriff Cartridge Stack. Dabei handelt es sich um einen Stack, der aus „base“, „plugin“ und „custom“ Cartridges in einer bestimmten Reihenfolge besteht. Ganz unten befindet sich die „base“ Cartridge, deren Code zu Beginn geladen wird und deren Funktionalität durch „plugin“ und „custom“ Cartridges erweitert wird, welche sich oberhalb befinden und somit später geladen werden. Die „Base“ Cartridge beinhaltet den Code der SFCC B2C Referenzimplementierung (SFRA). „Plugin“ Cartridges erweitern die Funktionalität der „base“ oder integrierter Cartridges, zum Beispiel für Zahlungsanbieter wie PayPal. Der letzte Cartridge-Typ sind „custom“ Cartridges, mit der die Website mit neuen Funktionen ausgestattet werden können, um individuellen Anforderungen des Shopbetreibers zu erfüllen. Die beste Methode, um neue Änderungen hinzuzufügen oder die Funktionalität von „base“ Cartridge zu erweitern, ist das Erweitern oder Überschreiben, je nachdem, was die Aufgabe der zusätzlichen Implementierung ist[33].

Damit ein neuer Cartridge auf die Website hochgeladen werden kann, muss sie zunächst zum Cartridge-Pfad hinzugefügt werden, der sich im Business Manager befindet. Der Cartridge

[32] Vgl. [Tra25 d]

[33] Ebenda

Pfad legt die Reihenfolge fest, in der der Code in den Cartridges geladen wird. Somit können einzelne Dateien, Klassen oder Methoden überschrieben werden[34].

4.2 jsPDF

Da die Erzeugung professioneller PDF-Dokumente recht komplex ist , wurde die Lösung jsPDF geschaffen. jsPDF bietet die Möglichkeit, eine Vielzahl von Dokumenten wie Zertifikate, Rechnungen, Berichte und mehr zu generieren[35]. Der Hersteller stellt eine ausführliche und aktuelle Dokumentation für diese Bibliothek zur Verfügung, sowie auf GitHub hochgeladenen Code mit Anwendungsbeispielen. Diese Bibliothek unterstützt die folgenden Funktionen:

- Änderung von Blattgröße, Ausrichtung und Einheiten
- Einstellung der Schriftart
- Hinzufügung und Entfernung von Seiten
- Hinzufügung von Formen (Linien, Kreise, Rechtecke)
- Hinzufügung und Einrichtung von Bildern
- Schreiben des Textes von rechts nach links
- Generierung des Erstellungsdatums des Dokuments
- Änderung der Farbe von Objekten
- Verwendung von AcroForm (Technik zum Hinzufügen interaktiver Elemente zum Dokument, wie z. B. Optionsfelder, Kontrollkästchen, Textfelder usw.)
- Hinzufügung der Anmerkungen von Dokument (z. B. Seitenzahlen, Name und URL)
- Einstellung von Sprache und mehr[36].

Die Bibliothek wurde umgeschrieben, um als „Third-Party-Library“ verwendet werden zu können. Sie kann auf derselben Ebene wie Projekt-Cartridges platziert und daher nicht im Cartridge-Pfad geschrieben werden. Das wird die Erfahrung der Entwickler im Backend verbessern und auch dazu beitragen, die Entwicklung zu beschleunigen[37].

[34] Vgl. [Tra25 d]

[35] Vgl. [Par25 a]

[36] Vgl. [Par25 b]

[37] Vgl. [Git25]

5 Aufsetzen des Prototyps

5.1 Generierungsfunktionen auf einer externen Umgebung

Die Erstellung eines Prototyps der ersten Methode der PDF-Generierung besteht aus mehreren Hauptschritten:

1. Erstellung einer REST API, die Daten von SFCC B2C entgegennimmt, diese Daten auf einem separaten Server in ein PDF-Dokument umwandelt und dieses Dokument zurück an SFCC B2C sendet.

Zur Erstellung der Anwendung wurden die JavaScript Runtime Environment Node.js sowie die express.js Bibliothek verwendet. Zu Beginn wurde ein neues Projekt erstellt und der Befehl **npm init** in die Befehlszeile eingegeben, um eine package.json Datei mit dem Namen der Anwendung, der Version, der Beschreibung sowie dem Namen der Datei, die die Hauptanwendung enthält, usw. zu erstellen. Danach wurde die express.js Bibliothek mit dem Befehl **npm install express** geladen.

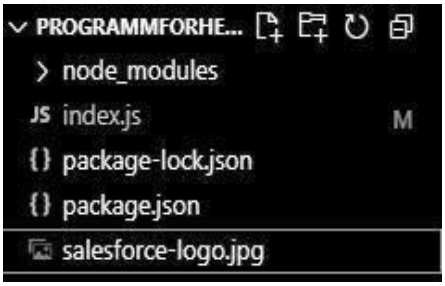


Abbildung 1: Programmstruktur für den Web Service

Zu Beginn des Programms werden alle erforderlichen Bibliotheken importiert und der Port bestimmt, auf dem das PDF-Dokument erstellt werden soll. Danach wurde eine Funktion mit der POST-Methode erstellt, da die Anwendung nicht nur Daten von einem anderen Server empfängt, sondern auch das generierte Dokument erzeugt und zurücksendet. Die Anwendung nimmt eine SFCC B2C Anfrage und deren Body mit allen für die Rechnung erforderlichen Informationen entgegen. Es wird eine neue Datei vom Typ „application/pdf“ mit der Codierung ISO-8859-1 erstellt, in die die empfangenen Informationen geschrieben werden. Schließlich wird die Datei in Form von Byte-Streams zurückgesendet. Das Zurücksenden der Datei an den Server ist ein wichtiger Teil des Prozesses, da diese Datei an eine E-Mail angehängt werden kann, die über die SFCC-Vorlage oder ein anderes System (z.B. SF Marketingcloud) in

Übereinstimmung mit dem Corporate Style des Kunden versendet wird. In der letzten Phase wurde eine Funktion erstellt, die gewährleistet, dass der Server über den entsprechenden Port arbeitet (siehe Anlage 4 - 5).

2. Testen der Anwendung

Um die API zu testen, wurde Postman verwendet. Darin wurde die entsprechende Methode ausgewählt, die Server URL eingegeben und der Body hinzugefügt. Der Dienst wurde von der IDE gestartet (Abb. 5) und im Fenster „Response“ von Postman wurde ein erfolgreich generiertes PDF-Dokument empfangen (siehe Anlage 6).

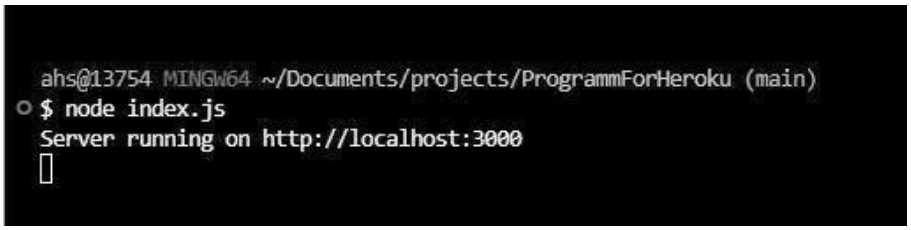


Abbildung 2: Starten des Servers aus der IDE

3. Hochladen der Anwendung auf die Plattform und Vorbereitung für den Start

Zunächst sollte das erstellte Projekt in GitHub Repository hochgeladen werden. Dann wird dieses Repository als Ressource für die Erstellung eines Services auf einer Plattform (z. B. Heroku oder Koyeb) verwendet. Bei der Erstellung eines Dienstes können solche Felder wie: Builder, Instance, Scaling, Ports, Health checks usw. konfiguriert werden. Danach wird der Service automatisch erstellt und eingesetzt. Wenn Änderungen am Projekt vorgenommen und in Repository übertragen werden, führt die Plattform automatisch einen neuen Build und ein Deployment des Dienstes mit den neuen Änderungen durch (siehe Anlage 7).

4. Einrichten der Webserver-Konfiguration im Business Manager

Abbildung 3: Web Servise Credentials hinzufügen

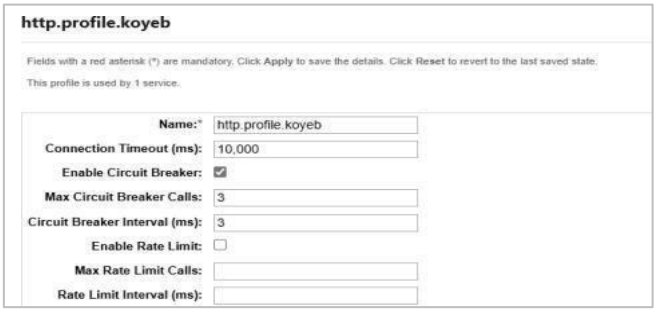


Abbildung 5: Web Service Profile Konfiguration

Abbildung 4: Web Service Konfiguration

5. Schreiben von Code in IDE

In der Anlage 8 ist der Code für die Erstellung eines Webdienstes dargestellt.
Der Code, der alle für die Rechnung erforderlichen Daten sammelt, den Webdienst aufruft und die Rechnung im IMPEX Directory speichert, ist in der Anlage 9 - 10 abgebildet. Da es nicht möglich ist, die Datei in diesem Verzeichnis direkt im Backend zu speichern, wurde Job von Business Manager konfiguriert und verwendet (siehe Abbildung 5):

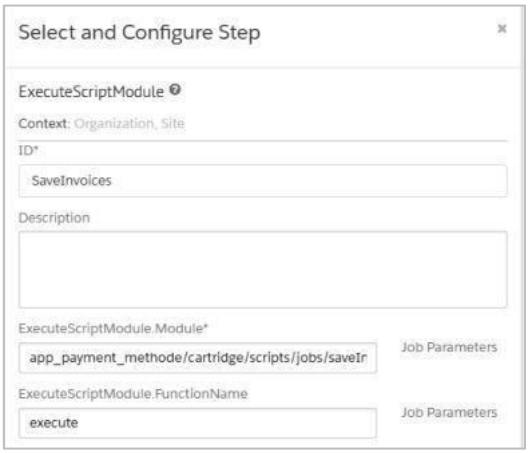


Abbildung 6: Job Steps

6. Starten und Testen der Anwendung

Der Job im Business Manager war eingeführt (Abb. 11) und Rechnungen wurden erfolgreich generiert (siehe Anlage 11).

Abbildung 7: Job Ergebnis

5.2 Integration per Third-Party-Library

Die Erstellung eines Prototyps für die PDF-Generierung mit Hilfe von jsPDF besteht aus folgenden Schritten:

1. Hinzufügung der Bibliothek zum Projekt

Zunächst wurde die Bibliothek jsPDF vom GitHub Repository heruntergeladen und dem Projekt auf derselben Ebene wie andere Cartridges hinzugefügt.

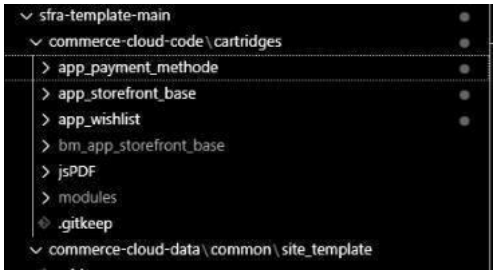


Abbildung 8: Programmstruktur in IDE

2. Erstellung des Codes

- Das ISML Template aus app_storefront_base wurde überschrieben, um die Schaltfläche „Download PDF“ in die Storefront einzufügen (siehe Anlage 12).

- Eine Funktion wurde erstellt, um die integrierte jsPDF-Bibliothek zu importieren, ein neues Dokument zu erstellen und Daten in dieses Dokument zu schreiben. Als Ergebnis gibt sie das erzeugte Dokument als String zurück (Anlage 13-14).

- In der Anlage 15 ist eine Funktion abgebildet, die Daten für eine Rechnung sammelt und die oben genannte Funktion aufruft

- Der Code, der in einem Controller geschrieben wurde, ruft eine Funktion zum Sammeln und Erzeugen eines Dokuments auf und lädt dieses Dokument, wenn die Schaltfläche „Download PDF“ angeklickt wird (siehe Anlage 16). Als Ergebnis wurde die Rechnung (siehe Anlage 17) heruntergeladen.

6 Vergleich von PDF-Generierung Methoden

Bei der Entwicklung einer Aufgabe müsste am besten geeignete Methode der PDF-Generierung ausgewählt werden. Aus diesem Grund werden in der Tabelle 5 die wichtigsten Unterschiede von PDF-Erzeugung Methoden analysiert.

Tabelle 4: Vergleich von PDF-Generierungs - Methoden

Vergleichende Eigenschaften	Generierungsfunktion auf externen Umgebungen (Heroku/Cloudhosting)	Externe Service (PDF Generator API)	Third-Party-Library (jsPDF)
Komplexität der Integration	Der Integrationsprozess ist recht kompliziert. Erstens muss es geprüft werden, ob die gewählte Plattform allen technischen Anforderungen laut der Aufgabenstellung entspricht. Zweitens ist es schwierig, eine API zu erstellen. Das Programm müsste gewährleisten, dass die Aufgabe effizient und mit einem hohen Maß an Sicherheit durchgeführt wird, insbesondere wenn es um die Verwendung privater oder finanzieller Informationen geht. Dazu kommt noch, dass eine Dokumentvorlage erzeugt werden sollte. Dieser Vorgang nimmt viel Zeit in Anspruch, da die Objekte unabhängig voneinander durch Koordinaten positioniert werden müssen. Es kann zu Schwierigkeiten bei der Codierung und Byte Stream kommen, und der Inhalt der Datei wird möglicherweise nicht richtig angezeigt oder dargestellt. Außerdem muss der Service im Business Manager sowie die Anwendung in externer Umgebung konfiguriert werden.	Das Prinzip der Integration ist ähnlich wie bei der ersten Methode, aber es besteht keine Notwendigkeit, selbst ein Programm zur PDF-Generierung von Dokumenten zu erstellen. Oftmals bieten Dienstleister vorgefertigte Vorlagen oder Code zur Erstellung einer Vorlage an, die geändert oder erweitert werden kann. Dies spart viel Zeit bei der Dokumentenerstellung.	Der Integrationsprozess ist im Vergleich zu früheren Methoden einfacher. Eine klare Dokumentation ist im GitHub-Repository zu finden. Die Bibliothek sollte dem Projekt nur auf der gleichen Ebene wie die anderen Cartridges hinzugefügt werden, und in diesem Fall müsste sie nicht einmal im Cartridgespfad geschrieben werden. Das Dokument wird „on-the-fly“ generiert und als String zurückgegeben. Die regelmäßige Erstellung einer Vorlage nimmt jedoch auch einige Zeit in Anspruch.
Wiederverwendbarkeit	Möglich	Möglich	Unmöglich
Kosten (für das Unternehmen)	- 4000$ pro Monat (Dev Starter Package. Eine Sammlung von Produkten und Ressourcen, die es Entwicklern ermöglichen, schnell Anwendungen zu erstellen) - 40000$ pro Monat (Prod Starter Package. Eine Reihe von Tools und Ressourcen zur Entwicklung globaler Anwendungen und zur Gewährleistung eines hohen Sicherheitsniveaus)[38]	- 6990$ pro Jahr (der Preis hängt von dem Anbieter und den Funktionen, die Anbieter bietet)	Die Lizenz ist kostenfrei

[38] Vgl. [Sal25 e]

7 Fazit

Die PDF-Generierung ist ein integraler Bestandteil des Prozesses der Erstellung eines Online-Shop. Sie wird zur Erzeugung verschiedener Arten von Dokumenten verwendet. Dadurch wird sichergestellt, dass die Daten aktuell sind und das Risiko von Darstellungsfehlern im Dokument aufgrund von unterschiedlichen Ausgabesystemen verringert wird. In der Vergangenheit war dieser Prozess für Entwickler recht schwierig, da SFCC B2C keine integrierten Funktionen zur Bewältigung dieser Aufgabe verfügte. Dies hat sich jedoch im Laufe der Zeit geändert, da alternative Methoden zur PDF-Generierung gefunden wurden. Dazu gehören die Erstellung einer Funktion, die das Dokument in einer externen Umgebung generiert, die Nutzung eines Service-Anbieter oder die Erzeugung des „on-the-fly“ Dokuments mit Hilfe von Third-Party-Bibliothek direkt im Code des Shopsystems.

Jede dieser Verfahren hat ihre eigenen Besonderheiten und Vorteile und kann für das Projekt verwendet werden. Es ist jedoch wichtig zu verstehen, welche Methode in einem bestimmten Fall am besten zu wählen ist. Die Wahl hängt von vielen Faktoren ab, z. B. der Komplexität der Integration, dem Zeit- und Kostenrahmen, der Möglichkeit, die Datei für den E-Mail-Versand zu verwenden, der Effizienz und der Flexibilität bei der Nutzung. Dieses Wissen ist für dotSource wichtig, um die Bedürfnisse seiner Kunden so weit wie möglich zu erfüllen und dadurch treue Kunden zu gewinnen. Während der Entwicklung eines Projekts kommunizieren die Entwickler von dotSource aktiv mit den Kunden über die Besonderheiten der Erstellung von Elementen der E-Commerce-Website. Aus diesem Grund sollen die Entwickler die bestehenden Methoden der Dokumentenerstellung, ihre Merkmale, Vor- und Nachteile und Unterschiede in der Verwendung im Detail erklären, damit die Kunden am besten geeignete Option für ihren Unternehmen wählen kann.

Quellenverzeichnis

[Gee25] GeeksForGeeks: "Socket in Computer Network". Abgerufen am 31.03.2025 von: https://www.geeksforgeeks.org/socket-in-computer-network/

[Git25] GitHub: "JS Libraries for Salesforce Commerce Cloud B2C". Abgerufen am 15.04.2025 von: https://github.com/taurgis/salesforce-commerce-cloud-libraries?tab=readme-ov-file#js-libraries-for-salesforce-commerce-cloud-b2c

[IBM25 a] IBM: "Was ist eine REST-API?". Abgerufen am 23.04.2025 von: https://www.ibm.com/de-de/topics/rest-apis

[IBM25 b] IBM: "What is cloud hosting?". Abgerufen am 11.04.2025 von: https://www.ibm.com/think/topics/cloud-hosting

[Kra22] Kratzke, N.: "Cloud-native Computing: Software Engineering von Diensten und Applikationen für die Cloud", Carl Hanser Verlag, München, 2022

[MS13] Middleton, N., Schneeman, R.: "Heroku: Up and Running", O'Reilly, Peking, Cambridge, Farnham, Köln, 2013

[Par25 a] Parallax: "jsPDF". Abgerufen am 15.04.2025 von: https://parall.ax/products/jspdf

[Par25 b] Parallax: "jsPDF". Abgerufen am 15.04.2025 von: https://raw.githack.com/MrRio/jsPDF/master/docs/index.html

[PDF25 a] PDF Generator API: "Create PDF documents and manage document templates". Abgerufen am 22.04.25 von: https://pdfgeneratorapi.com/

[PDF25 b] PDF Generator API: "PDF Generator API (4.0.10)". Abgerufen am 22.04.25 von: https://docs.pdfgeneratorapi.com/v4/

[PDF25 c] PDF Generator API: "Simple, predictable pricing". Abgerufen am 22.04.25 von: https://pdfgeneratorapi.com/pricing

[Sal25 a] Salesforce: "The History of Salesforce". Abgerufen am 21.03.2025 von: https://www.salesforce.com/news/stories/the-history-of-salesforce/

[Sal25 b] Salesforce Blog: "Erfolgreich im B2C E-Commerce: So gelingt‘s". Abgerufen am 21.03.2025 von: https://www.salesforce.com/de/blog/b2c-e-commerce/

[Sal25 c] Salesforce Developers: "Web Services". Abgerufen am 28.03.2025 von: https://developer.salesforce.com/docs/commerce/b2c-commerce/guide/b2c-webservices.html

[Sal25 d] Salesforce, "Class ServiceConfig". Abgerufen am 31.03.2025 von: https://salesforcecommercecloud.github.io/b2c-dev-doc/docs/current/scriptapi/html/index.html?target=class_dw_svc_ServiceConfig.html

[Sal25 e] Salesforce: "Heroku Pricing". Abgerufen am 24.04.2025 von: https://www.salesforce.com/editions-pricing/heroku/

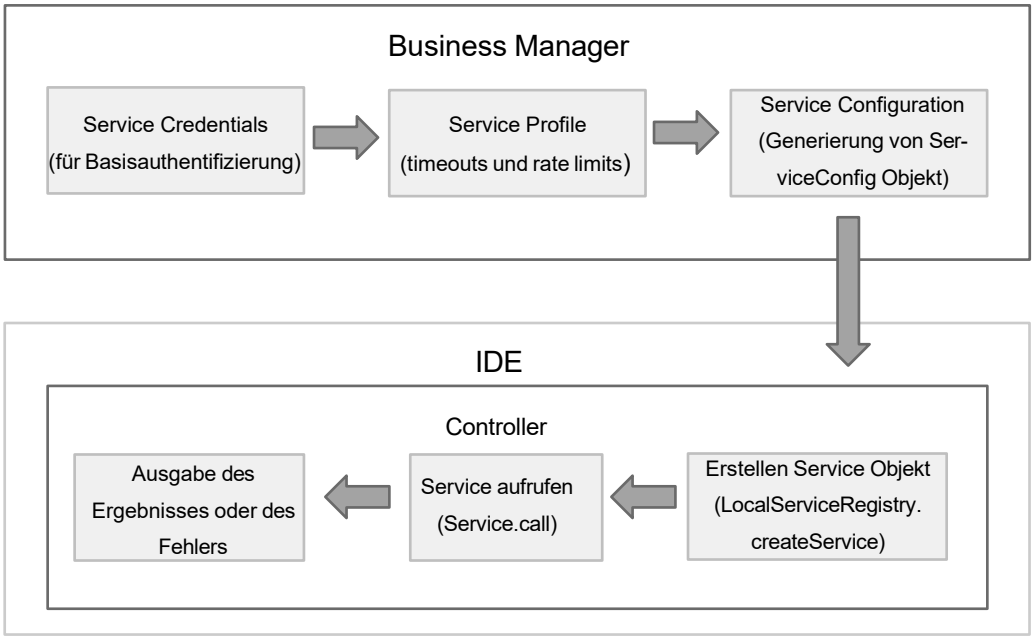
[Tra25 a] Trailhead: "Learn About the Web Services Framework". Abgerufen am 28.03.2025 von: https://trailhead.salesforce.com/content/learn/modules/b2c-integration-approaches/b2c-learn-web-services-framework

[Tra25 b] Trailhead: "Create a Web Service". Abgerufen am 31.03.2025 von: https://trailhead.salesforce.com/content/learn/modules/b2c-integration-approaches/b2c-create-web-service

[Tra25 c] Trailhead: "Ensure Integration Security". Abgerufen am 17.04.2025 von: https://trailhead.salesforce.com/content/learn/modules/b2c-integration-approaches/b2c-ensure-integration-security

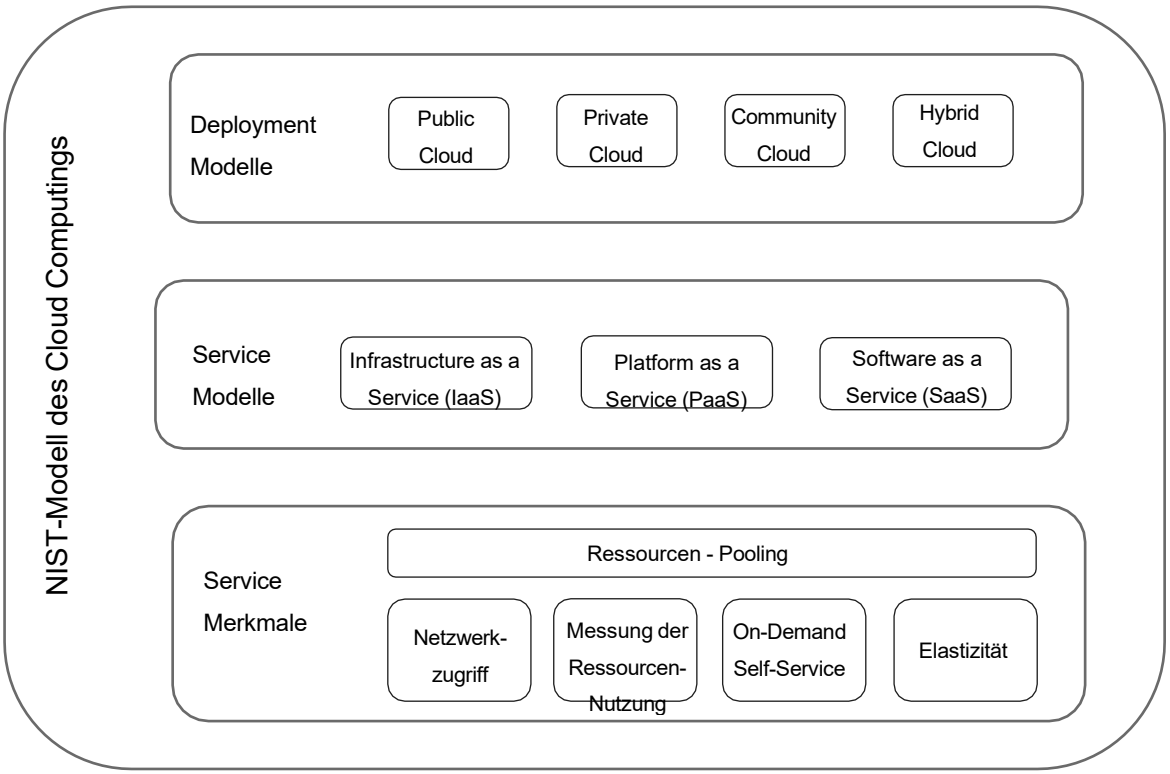
[Tra25 d] Trailhead: "Get to Know B2C Commerce Cartridges". Abgerufen am 15.04.2025 von: https://trailhead.salesforce.com/content/learn/modules/b2c-cartridges/b2c-cartridges-explore?trail_id=develop-for-commerce-cloud

[Tra25 e] Trailhead: "Get Started with Commerce Cloud Business Manager". Abgerufen am 28.04.2025 von: https://trailhead.salesforce.com/de/content/learn/modules/cc-digital-for-developers/cc-business-manager



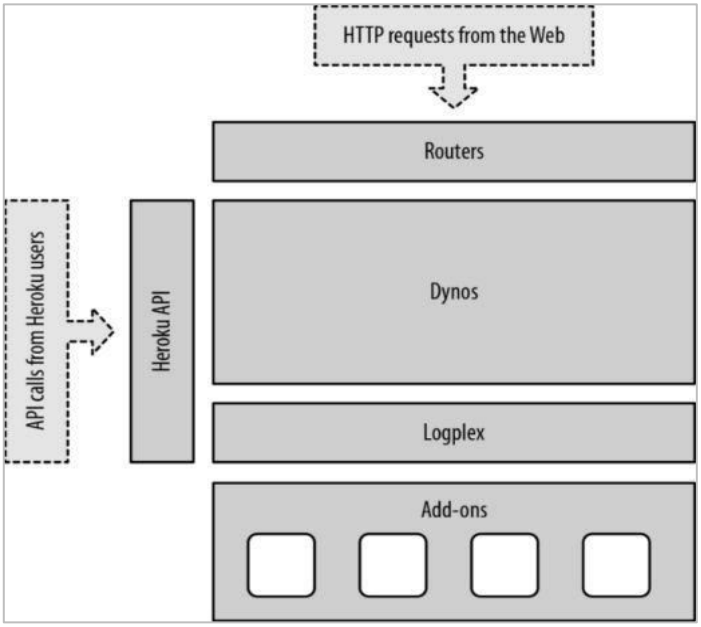
Anlage 1: Verfahren zur Erstellung eines Web Services

Quelle: Eigene Darstellung nach [Sal25 c]



Anlage 2: NIST-Modell des Cloud Computings

Quelle: Eigene Darstellung nach [Kra22], S. 12

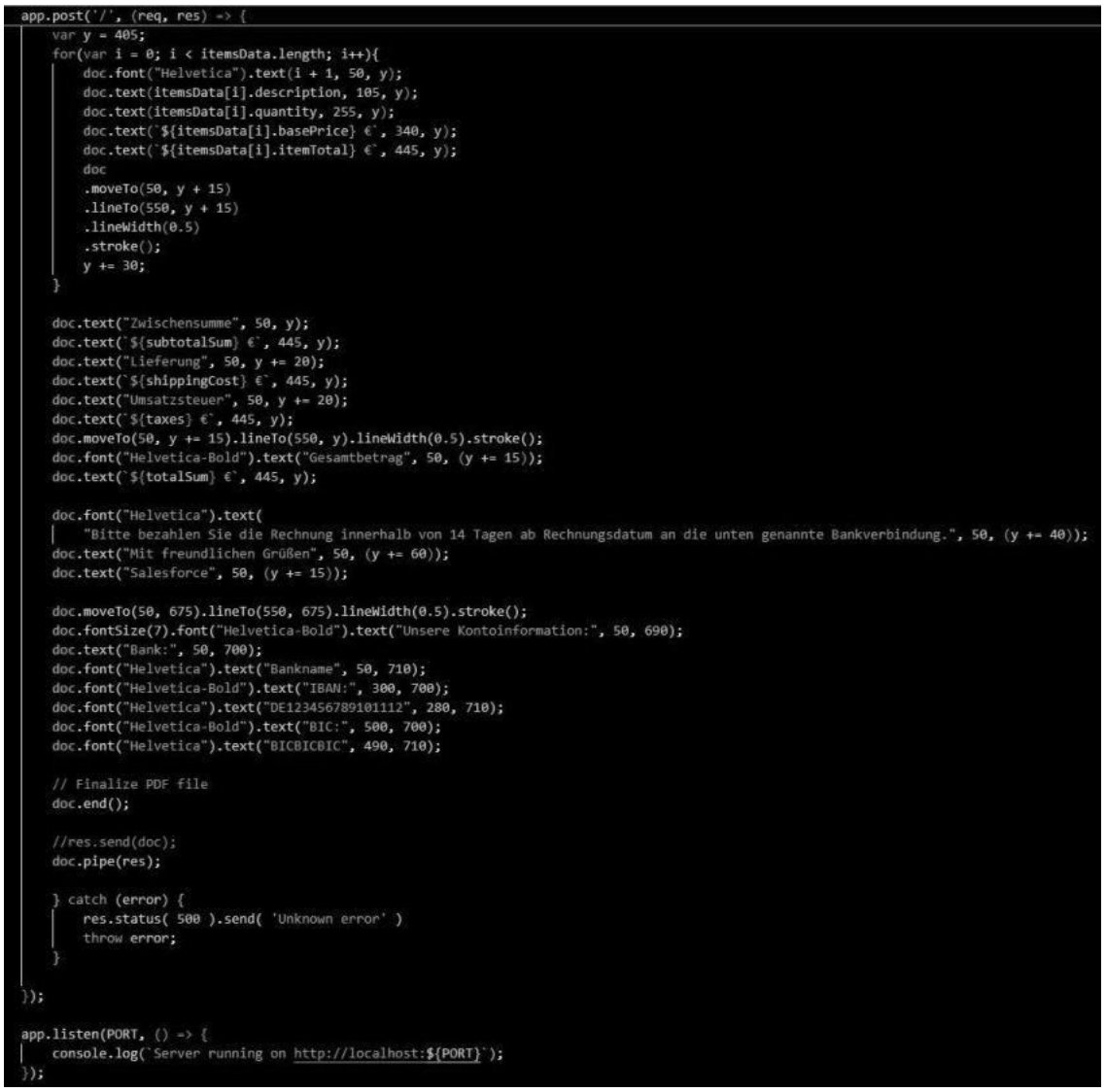


Anlage 3: Heroku Architektur

Quelle: [MS13]

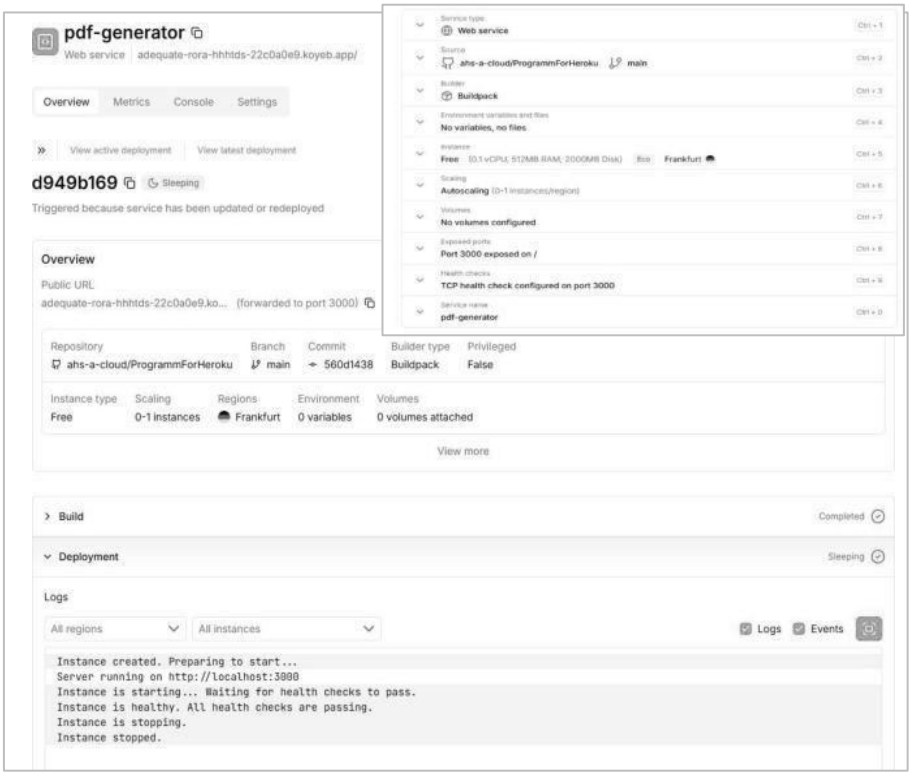
```
const express = require('express');
const app = express();
const path = require('path');

app.use(express.json());

const PORT = process.env.PORT || 3000;

app.post('/', (req, res) => {
    // get request values inside req.body
    const {firstName, lastName, adress, city, postalCode, invoiceNumber, itemsData, taxes, date, shippingCost, subtotalSum, totalSum} = req.body;

    try {

    // Include pdfkit library and fs module of Node.js
    const PDFDocument = require("pdfkit");
    const fs = require("fs");

    // Create a document
    const doc = new PDFDocument();

    // Pipe its output
    const filePath = path.join(__dirname, `InvoiceNr${invoiceNumber}.pdf`);
    doc.pipe(fs.createWriteStream(filePath));

    res.setHeader('Content-Type', 'application/pdf; charset=iso-8859-1');
    doc.image('salesforce-logo.jpg', 460, 30, { fit: [100, 100] });

    const fullName = firstName + " " + lastName;
    doc.fontSize(10).text(fullName, 50, 160);
    doc.text(adress, 50, 175);
    doc.text(`${postalCode}, ${city}`, 50, 190);

    doc.fontSize(10).text("Salesforce GmbH", 450, 160);
    doc.text("Erika-Mann-Str. 31", 450, 175);
    doc.text("80636, München ", 450, 190);
    if(date.month < 10) doc.text(`Datum: ${date.day}.0${date.month}.${date.year}`, 450, 205);
    else doc.text(`Datum: ${date.day}.${date.month}.${date.year}`, 450, 205);

    doc.fontSize(13).text(`Rechnung Nr. ${invoiceNumber}`, 50, 310);
    doc
        .fontSize(10)
        .text(
        "Vielen Dank für Ihren Auftrag. Wir berechnen Ihnen folgende Lieferung bzw. Leistung:", 50, 335 );
    doc.font("Helvetica-Bold").text("Nr.", 54, 370);
    doc.text("Bezeichnung", 105, 370);
    doc.text("Anzahl", 240, 370);
    doc.text("Einzelpreis", 320, 370);
    doc.text("Gesamtpreis", 420, 370);
    doc.moveTo(50, 390).lineTo(550, 390).lineWidth(0.5).stroke();
```

Anlage 4: Code der Generierungsfunktion in einer externen Umgebung



```
app.post('/', (req, res) => {
    var y = 405;
    for(var i = 0; i < itemsData.length; i++){
        doc.font("Helvetica").text(i + 1, 50, y);
        doc.text(itemsData[i].description, 105, y);
        doc.text(itemsData[i].quantity, 255, y);
        doc.text(`${itemsData[i].basePrice} €`, 340, y);
        doc.text(`${itemsData[i].itemTotal} €`, 445, y);
        doc
        .moveTo(50, y + 15)
        .lineTo(550, y + 15)
        .lineWidth(0.5)
        .stroke();
        y += 30;
    }

    doc.text("Zwischensumme", 50, y);
    doc.text(`${subtotalSum} €`, 445, y);
    doc.text("Lieferung", 50, y += 20);
    doc.text(`${shippingCost} €`, 445, y);
    doc.text("Umsatzsteuer", 50, y += 20);
    doc.text(`${taxes} €`, 445, y);
    doc.moveTo(50, y += 15).lineTo(550, y).lineWidth(0.5).stroke();
    doc.font("Helvetica-Bold").text("Gesamtbetrag", 50, (y += 15));
    doc.text(`${totalSum} €`, 445, y);

    doc.font("Helvetica").text(
        "Bitte bezahlen Sie die Rechnung innerhalb von 14 Tagen ab Rechnungsdatum an die unten genannte Bankverbindung.", 50, (y += 40));
    doc.text("Mit freundlichen Grüßen", 50, (y += 60));
    doc.text("Salesforce", 50, (y += 15));

    doc.moveTo(50, 675).lineTo(550, 675).lineWidth(0.5).stroke();
    doc.fontSize(7).font("Helvetica-Bold").text("Unsere Kontoinformation:", 50, 690);
    doc.text("Bank:", 50, 700);
    doc.font("Helvetica").text("Bankname", 50, 710);
    doc.font("Helvetica-Bold").text("IBAN:", 300, 700);
    doc.font("Helvetica").text("DE123456789101112", 280, 710);
    doc.font("Helvetica-Bold").text("BIC:", 500, 700);
    doc.font("Helvetica").text("BICBICBIC", 490, 710);

    // Finalize PDF file
    doc.end();

    //res.send(doc);
    doc.pipe(res);

    } catch (error) {
        res.status( 500 ).send( 'Unknown error' )
        throw error;
    }

});

app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

Anlage 5: Code der Generierungsfunktion in einer externen Umgebung

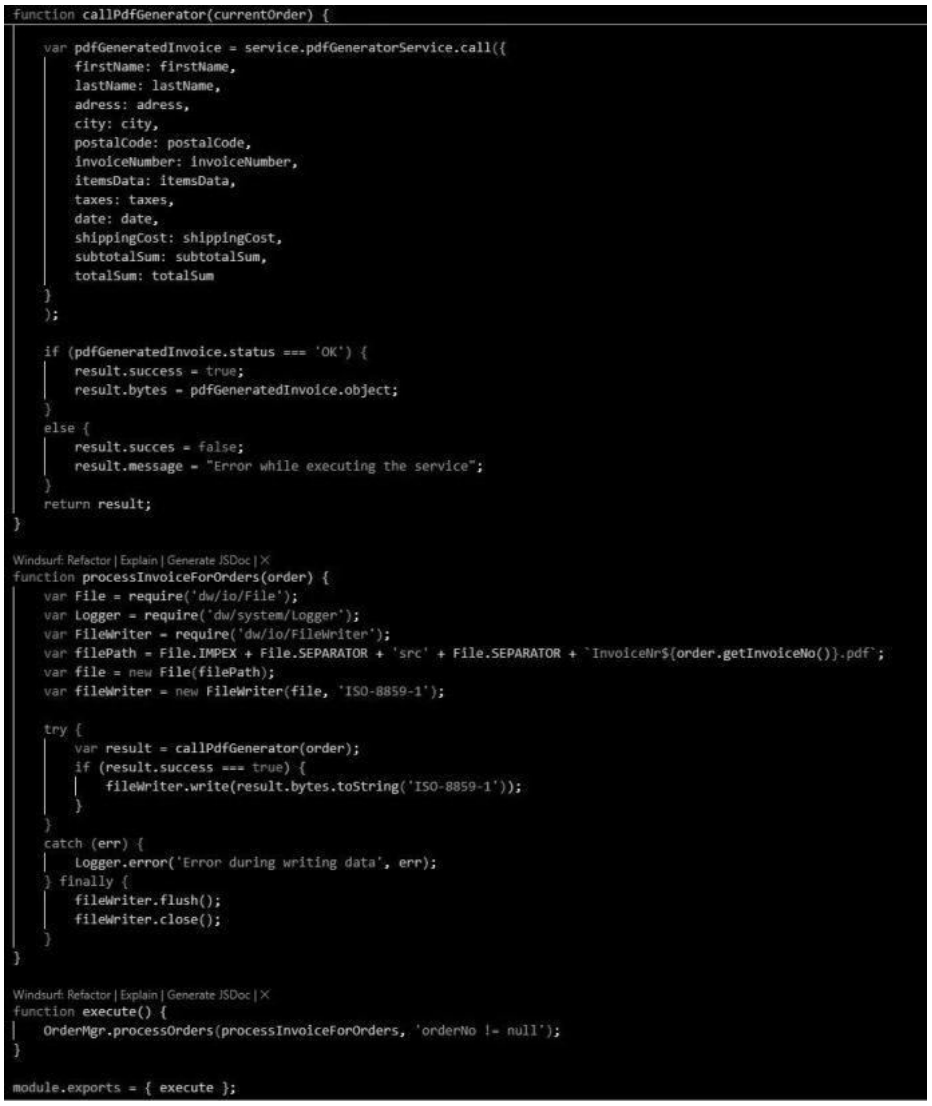


Anlage 7: Erstellter und eingesetzter Service im Koyeb

```
'use strict';

var LocalServiceRegistry = require('dw/svc/LocalServiceRegistry');

var pdfGeneratorService = LocalServiceRegistry.createService('http.rest.heroku', {
    Windsurf: Refactor | Explain | Generate JSDoc | X
    createRequest: function (svc, params) {
        svc.setRequestMethod('POST');
        svc.addHeader('Content-Type', 'application/json');
        var body = {
            firstName: params.firstName,
            lastName: params.lastName,
            adress: params.adress,
            city: params.city,
            postalCode: params.postalCode,
            invoiceNumber: params.invoiceNumber,
            itemsData: params.itemsData,
            taxes: params.taxes,
            date: params.date,
            shippingCost: params.shippingCost,
            subtotalSum: params.subtotalSum,
            totalSum: params.totalSum
        }

        return JSON.stringify(body);
    },


    Windsurf: Refactor | Explain | Generate JSDoc | X
    parseResponse: function (svc, client) {
        if(client.statusCode == 200) {
            var bytes = client.getBytes();
            return bytes;
        }

        else {
            return "Error";
        }
    }
});

module.exports = {
    pdfGeneratorService: pdfGeneratorService
}
```
Anlage 8: Code für die Erstellung eines Webdienstes

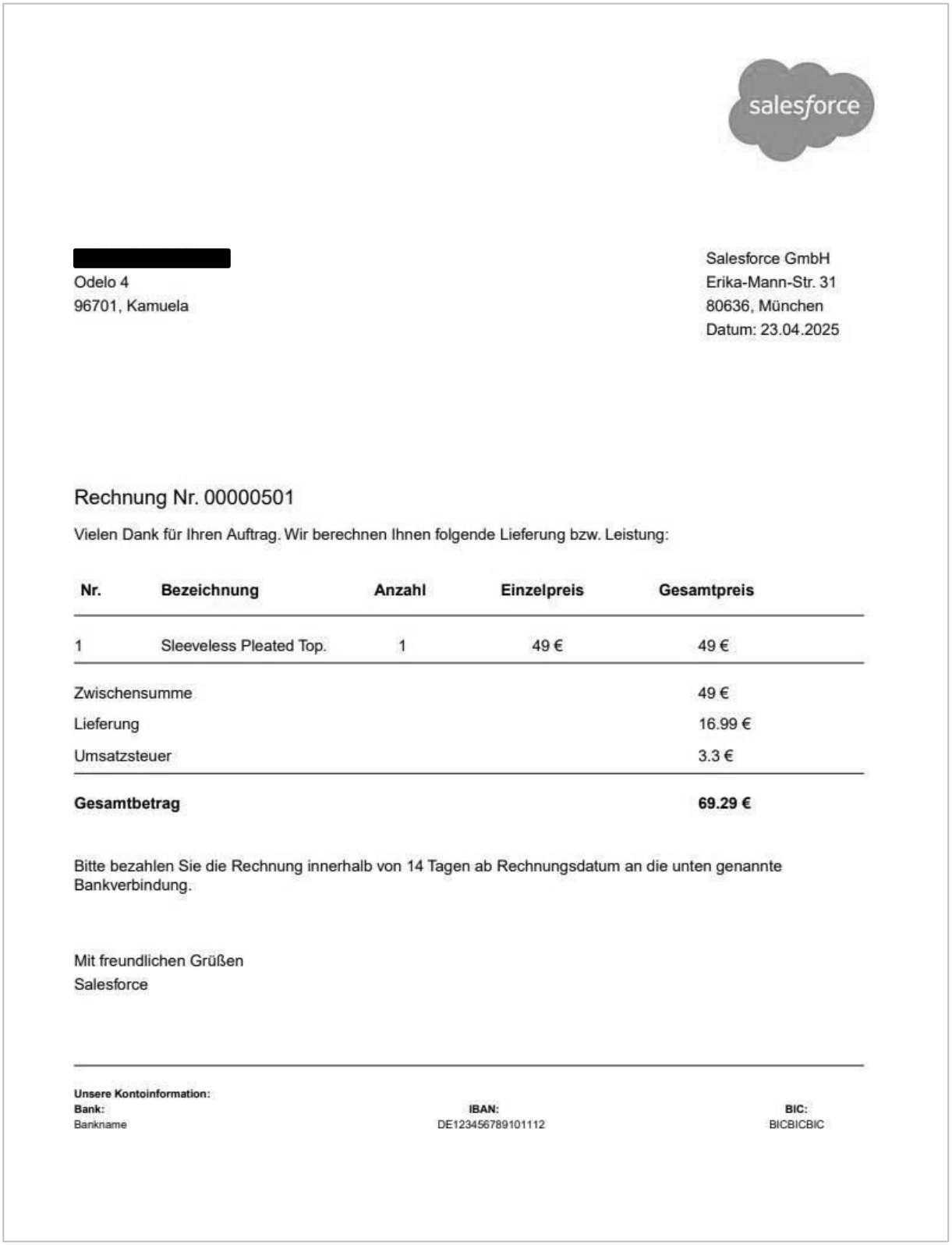
```
'use strict';

var ShippingMgr = require('dw/order/ShippingMgr');
var service = require('*/cartridge/services/pdfGenerator');
var OrderMgr = require('dw/order/OrderMgr');

Windsurf: Refactor | Explain | Generate JSDoc | X
function callPdfGenerator(currentOrder) {
    var result = {};
    var customInfo = currentOrder.getBillingAddress();
    var firstName = customInfo.getFirstName();
    var lastName = customInfo.getLastName();
    var phone = customInfo.getPhone();
    var email = currentOrder.getCustomerEmail();
    var adress = customInfo.getAddress1();
    var city = customInfo.getCity();
    var postalCode = customInfo.getPostalCode();
    var invoiceNumber = currentOrder.getInvoiceNo();

    // Products info
    var items = currentOrder.getAllProductLineItems();
    var itemsData = [];
    var iterator = items.iterator();
    var subtotalSum = 0;
    while (iterator.hasNext()) {
        var item = iterator.next();
        var basePrice = item.getBasePrice().getValue();
        if (basePrice == 0) { continue; }
        itemsData.push({
            description: item.getProductName(),
            basePrice: basePrice,
            quantity: item.getQuantity().getValue(),
            itemTotal: (basePrice * item.getQuantity())
        });
        subtotalSum += basePrice;
    }

    // Shipping
    var shipment = currentOrder.defaultShipment;
    var shippingMethod = shipment.shippingMethod;
    var shippingModel = ShippingMgr.getShipmentShippingModel(shipment);
    var shippingCost = shippingModel.getShippingCost(shippingMethod).getAmount().getValue();

    // Date
    const today = new Date();
    const date = {
        day: today.getDate(),
        month: today.getMonth() + 1,
        year: today.getFullYear(),
    };

    var taxes = currentOrder.getTotalTax().getValue();
    var totalSum = (subtotalSum + shippingCost + taxes).toFixed(2);

    var customInfo = currentOrder.getBillingAddress();
```

Anlage 9: Code für Webdienst Aufruf und Speicherung die Rechnung im IMPEX



```
function callPdfGenerator(currentOrder) {

    var pdfGeneratedInvoice = service.pdfGeneratorService.call({
        firstName: firstName,
        lastName: lastName,
        adress: adress,
        city: city,
        postalCode: postalCode,
        invoiceNumber: invoiceNumber,
        itemsData: itemsData,
        taxes: taxes,
        date: date,
        shippingCost: shippingCost,
        subtotalSum: subtotalSum,
        totalSum: totalSum
    }
    );

    if (pdfGeneratedInvoice.status === 'OK') {
        result.success = true;
        result.bytes = pdfGeneratedInvoice.object;
    }
    else {
        result.succes = false;
        result.message = "Error while executing the service";
    }
    return result;
}

Windsurf: Refactor | Explain | Generate JSDoc | X
function processInvoiceForOrders(order) {
    var File = require('dw/io/File');
    var Logger = require('dw/system/Logger');
    var FileWriter = require('dw/io/FileWriter');
    var filePath = File.IMPEX + File.SEPARATOR + 'src' + File.SEPARATOR + `InvoiceNr${order.getInvoiceNo()}.pdf`;
    var file = new File(filePath);
    var fileWriter = new FileWriter(file, 'ISO-8859-1');

    try {
        var result = callPdfGenerator(order);
        if (result.success === true) {
            fileWriter.write(result.bytes.toString('ISO-8859-1'));
        }
    }
    catch (err) {
        Logger.error('Error during writing data', err);
    } finally {
        fileWriter.flush();
        fileWriter.close();
    }
}

Windsurf: Refactor | Explain | Generate JSDoc | X
function execute() {
    OrderMgr.processOrders(processInvoiceForOrders, 'orderNo != null');
}

module.exports = { execute };
```

Anlage 10: Code für Webdienst Aufruf und Speicherung die Rechnung im IMPEX



[redacted]
Odelo 4
96701, Kamuela

Salesforce GmbH
Erika-Mann-Str. 31
80636, München
Datum: 23.04.2025

Rechnung Nr. 00000501

Vielen Dank für Ihren Auftrag. Wir berechnen Ihnen folgende Lieferung bzw. Leistung:

Nr.	Bezeichnung	Anzahl	Einzelpreis	Gesamtpreis
1	Sleeveless Pleated Top.	1	49 €	49 €

Zwischensumme	49 €
Lieferung	16.99 €
Umsatzsteuer	3.3 €
Gesamtbetrag	**69.29 €**

Bitte bezahlen Sie die Rechnung innerhalb von 14 Tagen ab Rechnungsdatum an die unten genannte Bankverbindung.

Mit freundlichen Grüßen
Salesforce

Unsere Kontoinformation:

Bank:	**IBAN:**	**BIC:**
Bankname	DE123456789101112	BICBICBIC

Anlage 11: PDF-Generierte Rechnung mit Hilfe von Generierungsfunktion auf auf externen Umgebung

```
<div class="invoice">
    <a href="${dw.web.URLUtils.https('Invoice-GenerateInvoice', 'orderID', pdict.order.orderNumber, 'orderToken', pdict.orderToken).toString()}"
    class="btn btn-primary btn-block download-invoice" role="button" aria-pressed="true">
        Download PDF
    </a>
</div>
```

Anlage 12: Code für die Einfügung der Schaltfläche

```
'use strict';
var ShippingMgr = require('dw/order/ShippingMgr');
var jsPDF = require('jsPDF');

Windsurf: Refactor | Explain | Generate JSDoc | X
function generateInvoice(data) {
    var info = data;
    var doc = new jsPDF();

    var fullName = data.firstName + " " + data.lastName;
    doc.setFont('DejaVu', 'normal');
    doc.setDrawColor(103, 173, 253);
    doc.setLineWidth(0.4);
    doc.setTextColor(103, 173, 253);
    doc.setFontSize(20);
    doc.text("Salesforce", 170, 15);
    doc.line(10, 20, 200, 20);
    doc.setTextColor(0, 0, 0);
    doc.setFontSize(11);
    doc.text(fullName, 10, 30);
    doc.text(data.adress, 10, 36);
    doc.text(`${data.postalCode}, ${data.city}`, 10, 42);

    doc.text("Salesforce GmbH", 170, 30);
    doc.text("Erika-Mann-Str. 31", 170, 36);
    doc.text("80636, München ", 170, 42);
    if(data.date.month < 10) doc.text(`Datum: ${data.date.day}.0${data.date.month}.${data.date.year}`, 170, 48);
    else doc.text(`Datum: ${data.date.day}.${data.date.month}.${data.date.year}`, 170, 48);

    doc.setFontSize(13);
    doc.text(`Rechnung Nr. ${data.invoiceNumber}`, 10, 80);
    doc.setFontSize(11);
    doc.text("Vielen Dank für Ihren Auftrag. Wir berechnen Ihnen folgende Lieferung bzw. Leistung:", 10, 90 );
    doc.setFont("DejaVu", "bold");
    doc.text("Nr.", 10, 100);
    doc.text("Bezeichnung", 40, 100);
    doc.text("Anzahl", 90, 100);
    doc.text("Einzelpreis", 130, 100);
    doc.text("Gesamtpreis", 170, 100);
    doc.setDrawColor(0, 0, 0);
    doc.setLineWidth(0.1);
    doc.line(10, 105, 200, 105);

    doc.setFont('DejaVu', 'normal');
    var y = 115;
    for(var i = 0; i < data.itemsData.length; i++){
        doc.text((i + 1).toString(), 11, y);
        doc.text(data.itemsData[i].description, 37, y);
        doc.text(data.itemsData[i].quantity.toString(), 95 , y);
        doc.text(`${data.itemsData[i].basePrice.toString()} €`, 135, y);
        doc.text(`${data.itemsData[i].itemTotal.toString()} €`, 180, y);
        doc.line(10, y + 5, 200, y + 5);
        y += 15;
    }
```

Anlage 13: Erstellen einer Rechnung mit Hilfe von jsPDF

```
    doc.text("Zwischensumme", 10, y );
    doc.text(`${data.subtotalSum.toString()} €`, 180, y);
    doc.text("Lieferung", 10, y += 10);
    doc.text(`${data.shippingCost.toString()} €`, 180, y);
    doc.text("Umsatzsteuer", 10, y += 10);
    doc.text(`${data.taxes.toString()} €`, 180, y);
    doc.line(10, y += 7, 200, y);
    doc.setFont('DejaVu', 'bold');
    doc.text("Gesamtbetrag", 10, y += 10);
    doc.text(`${data.totalSum.toString()} €`, 180, y);

    doc.setFont('DejaVu', 'normal');
    doc.text(
        "Bitte bezahlen Sie die Rechnung innerhalb von 14 Tagen ab Rechnungsdatum an die unten genannte Bankverbindung.", 10, (y += 20));
    doc.text("Mit freundlichen Grüßen", 10, (y += 20));
    doc.text("Salesforce", 10, (y += 10));

    doc.line(10, 270, 200, 270);
    doc.setFontSize(7);
    doc.setFont('DejaVu', 'bold');
    doc.text("Unsere Kontoinformation:", 10, 280);
    doc.text("Bank:", 12, 285);
    doc.setFont('DejaVu', 'normal');
    doc.text("Bankname", 10, 290);
    doc.setFont('DejaVu', 'bold');
    doc.text("IBAN:", 100, 285);
    doc.setFont('DejaVu', 'normal');
    doc.text("DE123456789101112", 94, 290);
    doc.setFont('DejaVu', 'bold');
    doc.text("BIC:", 170, 285);
    doc.setFont('DejaVu', 'normal');
    doc.text("BICBICBIC", 167, 290);
    return doc.output();
}
```

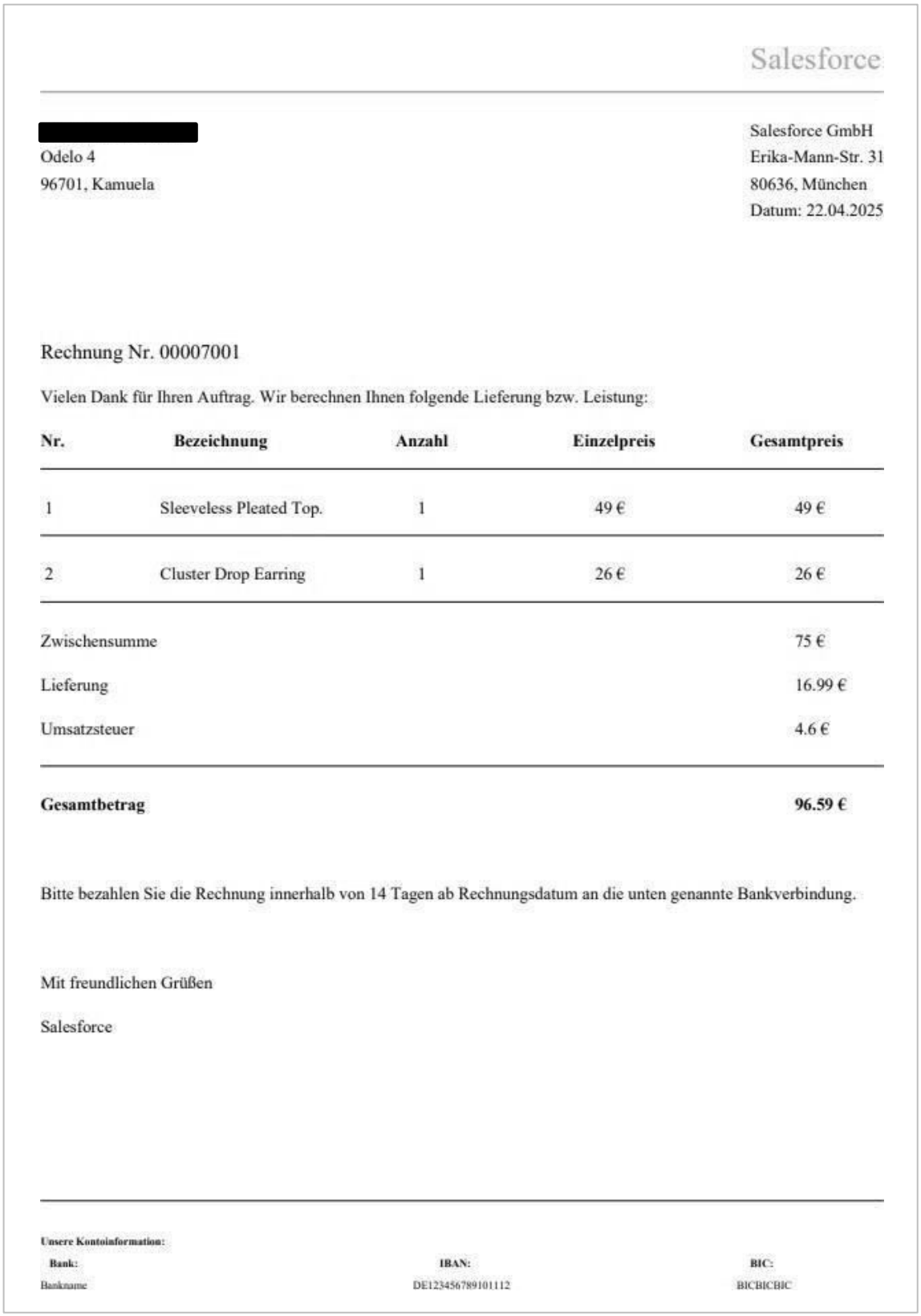
Anlage 14: Erstellen einer Rechnung mit Hilfe von jsPDF

```
function storeInvoiceIinfo (currentOrder) {

    var customInfo = currentOrder.getBillingAddress();

    var data = {
        firstName: firstName,
        lastName: lastName,
        adress: adress,
        city: city,
        postalCode: postalCode,
        invoiceNumber: invoiceNumber,
        itemsData: itemsData,
        taxes: taxes,
        date: date,
        shippingCost: shippingCost,
        subtotalSum: subtotalSum,
        totalSum: totalSum
    }

    var doc = generateInvoice(data);
    return doc;
}

module.exports = {
    storeInvoiceIinfo: storeInvoiceIinfo
}
```

Anlage 15: Funktion zur Vorbereitung der Daten für die PDF-Generierung

```
'use strict';

/**
 * @namespace Invoice
 */

var server = require('server');

var OrderMgr = require('dw/order/OrderMgr');
var Transaction = require('dw/system/Transaction');
var Resource = require('dw/web/Resource');

server.get('GenerateInvoice', function (req, res, next) {
    var invoiceHelper = require('*/cartridge/scripts/orderCustom/invoiceGeneratorHelper');
    if (!req.querystring.orderToken || !req.querystring.orderID) {
        res.render('/error', {
            message: Resource.msg('error.confirmation.error', 'confirmation', null)
        });

        return next();
    }

    var order = OrderMgr.getOrder(req.querystring.orderID, req.querystring.orderToken);
    if (!order || order.customer.ID !== req.currentCustomer.raw.ID) {
        res.render('/error', {
            message: Resource.msg('error.confirmation.error', 'confirmation', null)
        });

        return next();
    }

    var result = invoiceHelper.storeInvoiceIinfo(order);

    response.setBuffered(false);
    response.setContentType('application/pdf; charset=iso-8859-1');
    res.setHttpHeader('Content-Disposition', 'attachment; filename="invoice.pdf"');
    var resWriter = response.getWriter();

    // Iterate through the data and print as needed
    resWriter.print(result);

    resWriter.flush();
    resWriter.close();

    next();
});

module.exports = server.exports();
```

Anlage 16: Code im Controller



Salesforce

[redacted]
Odelo 4
96701, Kamuela

Salesforce GmbH
Erika-Mann-Str. 31
80636, München
Datum: 22.04.2025

Rechnung Nr. 00007001

Vielen Dank für Ihren Auftrag. Wir berechnen Ihnen folgende Lieferung bzw. Leistung:

Nr.	Bezeichnung	Anzahl	Einzelpreis	Gesamtpreis
1	Sleeveless Pleated Top.	1	49 €	49 €
2	Cluster Drop Earring	1	26 €	26 €

Zwischensumme	75 €
Lieferung	16.99 €
Umsatzsteuer	4.6 €
Gesamtbetrag	**96.59 €**

Bitte bezahlen Sie die Rechnung innerhalb von 14 Tagen ab Rechnungsdatum an die unten genannte Bankverbindung.

Mit freundlichen Grüßen

Salesforce

Unsere Kontoinformation:

Bank:	IBAN:	BIC:
Bankname	DE123456789101112	BICBICBIC

Anlage 17: PDF-Generierte Rechnung mit Hilfe von jsPD