Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

2GP ..... Second-Generation-Packages
API ..... Application Programming Interface
ECA ..... External Client Apps
REST ..... Representational State Transfer

1 Einführung

„Keiner von uns ist so klug wie wir alle.“ **— Ken Blanchard**

Das Zitat von Ken Blanchard, einem US-amerikanischem Unternehmer und Autor von Management Büchern, thematisiert die Überlegenheit einer Gruppe gegenüber einem Individuum.[1] Dieselbe Überlegung wurde schon vielfach behandelt. So werden beispielsweise in dem Buch The Wisdom of Crowds von Autor James Surowiecki eine Vielzahl von Fallstudien aufgelistet, um die Existenz einer überlegenen kollektiven Intelligenz zu unterstreichen.[2] Kollaboration zählt auch zu einem zentralen Bestandteil für den Erfolg in einem modernen Unternehmen. Sei es durch Werkzeuge oder Plattformen, welche einen Fokus auf Kollaboration legen oder Bewegungen und Praktiken, welche die Zusammenarbeit innerhalb von Firmen stärken sollen. In der IT-Branche bewährt sich ebenfalls Zusammenarbeit als elementare Praxis. Plattformen wie das Versionsverwaltungssystem Git oder Salesforce, welches Mitarbeiter unterschiedlicher Abteilung verbindet, zählen zu den Marktführern.[3,4] Entwicklerteams für Software werden durch Bewegungen wie DevOps oder Scrum immer integrierter in der Kundenberatung, Betreuung sowie in der Projektverwaltung.

Dieses Zusammenschließen, für einen besseren Erfolg, ist auch seit längerem in der Softwarelandschaft bemerkbar. Die Ersten Protokolle, welche diese Kommunikation erlaubten, entstanden bereits in den Neunzigern. Diese Programmschnittstellen, auch Application Programming Interface (API) genannt, können Anwendungen um Funktionalitäten externer Services erweitern.[5]

Ziel dieser Arbeit ist es, eine prototypische Implementierung einer Schnittstelle in Salesforce vorzunehmen, um auf mögliche Kundenprojekte vorbereitet zu sein. Dabei sollen Umfrageergebnisse exemplarisch an eine Schnittstelle gesendet werden und letztlich durch Salesforce Objekte und Beziehungen repräsentiert werden. Hauptbestandteil der Arbeit bilden dabei die Konzeptionierung und Implementierung der Schnittstelle, sowie der erforderten Übersetzungslogik. Das Senden der Anfragen an die Schnittstelle soll nicht thematisiert werden – die Daten, welche übermittelt werden, schon.

---

[1] [Nur22]
[2] Vgl. [Sur17]
[3] Vgl. [Sal24f]
[4] Vgl. [Exp23]
[5] Vgl. [Mil23]

1.1 Wachstumstrends von Programmierschnittstellen

Die Wichtigkeit des Zusammenschließens von Programmen und Diensten über Schnittstellen wurde jedoch nicht nur früh identifiziert, sondern nimmt auch Heutzutage noch stetig zu. Postman ist eine führende Plattform für das Entwickeln von API anfragen, um diese Effektiv zu testen. Die gleichnamige Firma, die diese Plattform verwaltet, berichtet über die Zunahme von Schnittstellen im State of the API-Report des Jahres 2024. Innerhalb der Umfrage, die über 5600 Personen beantworteten, gaben 74% der Befragten an, das ihre Unternehmen einen API zentrierten Entwicklungsprozess pflegen. Dies ist eine Erhöhung von 8% zum Vorjahr. 48% der Teilnehmer gaben dabei an, mehr Zeit und Ressourcen in die Entwicklung von Programmierschnittstellen zu investieren. Diese dienen nicht nur als Lebensqualitäts-Funktionen, sondern generieren bei 21% der befragten Unternehmen mehr als 75% des Umsatzes.[6]

Auch die dotSource SE verfolgt aktuelle Änderungen auf dem Gebiet der Softwareintegration. Die Ergebnisse der Arbeit verfolgen somit derzeitig wichtige Themen auf dem Markt und bieten einen wichtigen Mehrwert für das Unternehmen. Die von Salesforce vertriebene Sammlung von APIs auf der Postman Plattform ist die Populärste. Naheliegenderweise wird Postman deswegen innerhalb der Arbeit während der Implementation, für das Testen von APIs und Entwickeln von HTTP-Anfragen, genutzt.

APIs Lassen sich anhand ihrer Architektur oder Technologie verschieden Klassifizieren. Zu den Heutzutage am meisten verwendeten APIs zählen die, welche sich nach der Representational State Transfer (REST) Architektur richten. Diese in der Jahrtausendwende von Dr. Roy Fielding entwickelten Designprinzipien sind Grundlage für simple, leicht skalierbare und flexible Programmschnittstellen. Sie sind Programmiersprachen unabhängig und implementieren dadurch in der Praxis weitere häufig verwendete Architekturen wie HTTP-Anfragen oder JSON-Dateien. Innerhalb der Jahre 2021-2022 waren REST-APIs, mit 69,3%, die am stärksten wachsenste Schnittstellenart.[7] Wobei sich Identifizieren lässt das heutige REST-APIs, aufgrund fehlender Standardisierungen, immer weniger dem ursprünglichen Rahmenwerk entsprechen.[8] Hypermedia, als ein Prinzip der REST-Architektur wird dabei besonders häufig vernachlässigt.[9] Das eigentliche Ziel, Schnittstellen ohne Dokumentation,

[6] Vgl. [Pos24]
[7] Vgl. [Mil23]
[8] Vgl. [RBD+16]
[9] Vgl. [NLB21]

über die ein Client in Kenntnis sein muss, funktionsfähig zu machen, ist dabei für moderne APIs von weniger Relevanz, sich für die REST-Architektur zu entscheiden. Dieser Zwiespalt zwischen Theorie und Praxisanwendung von REST macht sich besonders in der Recherche zu den Begriffen REST- und RESTful-API deutlich, wobei man auf verschiedene Interpretationen trifft.[10,11]

Als akkuratere Benennung für moderne REST-APIs würden sich JSON- oder URI-API anbieten, da diese in fast allen modernen Implementationen, aufgrund der einhergehenden Schlichtheit und Flexibilität, enthalten sind.[12] Wegen des starken Wachstums soll dennoch innerhalb der Arbeit eine REST-API entwickelt werden. Da REST aber nur als theoretisches Designkonzept für Schnittstellen dient, sind tatsächliche Ausarbeitungen maßgeblich durch die Implementation der Salesforce Plattform bestimmt. Eine weitere Betrachtung von REST ist folglich nicht zum Bearbeiten der Aufgabe nötig und soll nicht in dieser Arbeit Vorkommen.

1.2 Salesforce

Die Verwendung der Salesforce Plattform gegenüber anderen begründet sich durch zwei treibende Faktoren. Dazu zählen die Gründe, aus welchen sich Kunden der dotSource SE, innerhalb der Customer-Relationship-Management und E-Commerce Branche für Salesforce entscheiden und warum die dotSource SE Kunden dabei unterstützt. Dabei ist Salesforce durch die, in der Einleitung erwähnten Strategie, unterschiedliche Abteilungen unter einer Plattform zu verbinden, seit der Gründung im Jahr 1999 stetig zu einem marktführenden Unternehmen gewachsen. Um diese Zusammenschließung zu arrangieren, ermöglicht Salesforce seinen Anwendern ein breites Spektrum an Konfigurationsmöglichkeiten. Dazu zählen für die Plattform optimierte Programmiersprachen, deren effektive Verwendung nur in Kombination mit viel erfordertem Hintergrundwissen entstehen kann, welches unerfahrene Nutzer mitunter nicht überschauen können.

Die führende Position der Plattform, sowie die Anforderungen an eine Anpassung führen vermehrt zu Kundenanfragen an die dotSource SE. Der Salesforce Kontext der Arbeit, gibt ihr damit eine erhöhte Wichtigkeit, was dessen Bearbeitung ergründen lässt. Des Weiteren werden durch ihn, wie beispielsweise in 1.1, klare Rahmen für die Inhalte der Arbeit gegeben.

---

[10] Vgl. [Gee22]
[11] Vgl. [TES+15], S. 11
[12] Vgl. Ebenda, S. 46

Zu den Programmiersprachen der Plattformen gehören beispielsweise Apex und ein Frontend-Programmiergerüst mit dem Namen Lightning Web Components.[13] Da sich die Aufgabe aber um eine Schnittstelle zwischen Programmen und nicht zwischen Programmen und Endanwender handelt, ist Letztere nicht Bestandteil der Arbeit.

Apex als in der Arbeit verwendete Programmiersprache weist eine Java ähnliche Syntax auf. Abweichungen gegenüber Java, welche nicht zu einem maßgeblich besseren Verständnis des Quellcodes beitragen, sowie Erklärungen von Grundkonzepten der Programmierung, sollen ebenfalls nicht Bestandteil der Arbeit sein.

Die Verwendung von Schnittstellen innerhalb der Salesforce Plattform erfordert Apex REST, um die API zur Verfügung zu stellen und External Client Apps (ECA) , um die Anfragen an die Schnittstelle zu Autorisieren. Diese verwendeten Technologien werden in den folgenden Kapiteln kurz erläutert.

1.2.1 Endpunkt Bereitstellung

Die Salesforce Plattform bietet Entwicklern eine Vielzahl an bereits vorgefertigten APIs und auch die Möglichkeit eigene Schnittstellen zu erstellen.[14] Daraus resultiert die Fragestellung welcher der beiden Kommunikationspartner, Salesforce und Drittanwendung, eine Erweiterung des Quellcodes vornimmt. Soll also der Kommunikationspartner seine Anfrage der Standard API von Salesforce anpassen, oder soll Salesforce eine API, welche der Anfrage der Drittanwendung entspricht, bereitstellen?

Die Anpassung auf Seite der externen Anwendung erfordert die Möglichkeit die REST-Anfrage überhaupt verändern zu können. Ist diese Grundlage nicht erfüllt, so muss eine eigene Salesforce Apex REST Schnittstelle geschrieben werden. Weitere Überlegungen, um die Fragestellung zielführend zu beantworten, können sein:

1. Wer soll die Wartungsarbeiten, für Änderungen in der Kommunikation, betreiben?
2. Soll die zur Einbindung benötigte Struktur der Salesforce Objekte außerhalb der Plattform bekannt sein?
3. Welche Aspekte der Kommunikation sind standardisiert und welche anpassungsfähig?

[13] Vgl. [Sal25c]
[14] Vgl. [Sal25b]

Für den weiteren Verlauf der Arbeit werden die Fragen so beantwortet, dass die Anfrage der Drittanwendung nicht abänderbar ist, sodass eine individuelle eigene Apex REST Schnittstelle sinnvoller zu implementieren ist. Die Erstellung einer Schnittstelle ist unter Salesforce, im Vergleich zu anderen Frameworks wie beispielsweise Springboot, welches beim Erstellen von REST-Schnittstellen zusätzliche Bibliotheken benötigt,[15] sehr vereinfacht. So werden große Teile des Verwaltungs- und Konfigurationsaufwandes bereits von Salesforce übernommen. Ein Salesforce Plattform-Entwickler kümmert sich deswegen hauptsächlich um die Erstellung der Schnittstellen Logik.

Dazu wird eine Apex Klasse erstellt und mit einer RestResource Annotation versehen. Dieser Annotation wird ein individuelles URL-Mapping mitgegeben, um verschiedene Endpunkte voneinander separat ansprechen zu können. Anschließend können Methoden, ebenfalls durch Annotationen, den verschiedenen HTTP-Anfragemethoden zugeordnet werden.

```
1. @RestResource(urlMapping='/Account/*')
2. global with sharing class MyRestResource {
3.
4.     @HttpDelete
5.     global static void doDelete() {
6.         RestRequest req = RestContext.request;
7.         RestResponse res = RestContext.response;
8.         String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
9.         Account account = [SELECT Id FROM Account WHERE Id = :accountId];
10.        delete account;
11.    }
12.
13.    @HttpGet
14.    global static Account doGet() {
15.        RestRequest req = RestContext.request;
16.        RestResponse res = RestContext.response;
17.        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
18.        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id = :accountId];
19.        return result;
20.    }
21.
22.    @HttpPost
23.    global static String doPost(String name,
24.        String phone, String website) {
25.        Account account = new Account();
26.        account.Name = name;
27.        account.phone = phone;
28.        account.website = website;
29.        insert account;
30.        return account.Id;
31.    }
32. }
```

Abbildung 1: Beispiel Code-Schnipsel der benötigten Annotationen
Quelle: [Sal25a]

[15] Vgl. [Sal25a]

Beim Aufrufen der Schnittstelle werden die jeweiligen Methoden ausgeführt und mittels eines Returns eine Antwort an den Anfragenden zurückgeschickt. Alternativ lassen sich Antworten über das Objekt RestContext.response verwalten. Dieses wird wie in der doDelete Methode in Abbildung 1 standardmäßig zurückgesendet, wenn sie den Rückgabetyp Void hat. Die Abbildung zeigt die oben erwähnten Annotationen, welche zur Aufsetzung der Schnittstelle nötig sind.

In dem Beispiel wird eine API über folgende URL bereitgestellt:

https://instanz.salesforce.com/services/apexrest/Account/

Die Adresse lässt sich in drei Abschnitte unterteilen:

1. Der erste Teil der Anfrage gibt die Webadresse der Salesforce Instanz an.
2. Der zweite ist der Standardpfad zu allen Apex-REST-Schnittstellen.
3. Der dritte ist das in der RestResource Annotation enthaltene Mapping.

Über die DELETE, GET und POST HTTP-Methoden können Anfragen mit einen Access Token diese API verwenden und Account Objekte auf der Plattform verwalten. Die Beschaffung des Access Tokens ist im Gedanken einer erhöhten Sicherheit jedoch mit etwas mehr Aufwand verbunden. Dies erfordert eine Korrekte Autorisierung gegenüber Salesforce.

1.2.2 Zugriffs Autorisierung

Innerhalb einer Salesforce Instanz wird oftmals mit sensiblen Kundendaten gearbeitet. Sollten diese falsch gesichert werden und unautorisiert abgreifbar sein, könnten schwere rechtliche Folgen für den Betreiber der Instanz entstehen. Zusätzlich könnten Angreifer die Schnittstelle verwenden um gezielt Plattform-Limitierungen auszunutzen, wie das Erstellen von Salesforce Objekten, um den Dienst zu blockieren.

Die Sicherung einer Schnittstelle ist also von maßgeblicher Bedeutung. Nur durch einen zuverlässigen Autorisierungsprozess sind die eben beschriebenen möglichen Probleme zu vermeiden. Eine Autorisierung innerhalb von Salesforce geschieht über industriestandardisierte und weitverbreite Protokolle wie OpenID Connect oder OAuth.[16]

[16] Vgl. [Sal24a]

Diese werden unter Salesforce eigenen Frameworks, dazu zählen Connected Apps und die seit dem Sommer 2024 Update neueren External Client Apps, in Form von verschieden Flows den Entwicklern bereitgestellt. Ein Flow ist eine, auf den Anwendungsfall abgestimmte, Umsetzung des OAuth 2.0 Protokolls. Zusätzlich kann, innerhalb einer Benutzeroberfläche des Frameworks, konfiguriert werden, welche konkreten Teile der Plattform der externen Anwendung zugänglich gemacht werden sollen.[17]

Im Zusammenhang mit der Arbeit sollen die External Client Apps als mögliche Neuerung testweise implementiert werden. Eine Evaluation, welches der Frameworks für Entwicklungsprozesse innerhalb der dotSource SE effizienter ist, würde den Rahmen der Arbeit sprengen und soll folglich nicht erwähnt werden. Eine Auswahl und Begründung des benötigten Flows erfolgt innerhalb der Konzeptionierung.

1.3 Schnittstellenentwicklung bei dotSource SE

Die dotSource SE bietet schon seit längerem die Entwicklung von, auf Kundenanforderungen maßgeschneiderte, Anpassungen der Salesforce Plattform als Teil des Service-Portfolios an. Schon heute gibt es Aufträge, welche die Entwicklung einer Schnittstelle erfordern, um externe Tools von Kunden integrieren zu können. Dabei werden Endpunkte mittels Connected Apps autorisiert und zur Verfügung gestellt. Als Grundlage des Handelns der dotSource SE dient dabei die dotSource DNA, ein Manuskript mit wichtigen Grundsätzen des Unternehmens. In diesem finden sich die in der Einleitung besprochene Zusammenarbeit, firmenintern und zum Kunden hin, verankert.

1.4 Zielstellungen der Arbeit

Aus diesem soll, mittels dieser Arbeit, dem Gedanken der Weiterentwicklung nachgegangen werden. Das Ziel der Arbeit ist es eine Einbindung einer externen Anwendung an eine Salesforce Instanz zu ermöglichen. Das Ergebnis der Arbeit dient Entwicklern in der Salesforce-Abteilung der dotSource SE, als Beispiel und Referenz für mögliche Kundenanfragen. Dazu werden aus den genannten Gründen (siehe 1.1) APIs verwendet. Zudem soll eine neue Technologie von Salesforce implementiert und deren Vorteile kurz erläutert werden. Das Nachfolgende Konzept soll thematisieren warum sich für die implementierten Bestandteile entschieden wurde.

---

[17] Vgl. [Sal25f]

2 Konzeptionierung

Da die Ergebnisse dieser Arbeit nur als Referenz dienen, liegen keine Spezifikationen an die Schnittstelle vor. Deswegen mussten vorher Spezifikationen konstruiert werden, wie sie auch in der Praxis vorkommen könnten. Die Anfrage stellt einen zentralen Baustein in der Aufgabenstellung dar, dient aber nicht zur Bearbeitung und wird folglich nur in der Konzeption erläutert und nicht in der Implementierung. Als Beispielfall wird eine Schnittstelle entworfen, welche Umfrage-Ergebnisse als JSON verschickt. Diese sollen in eine Salesforce Instanz integriert werden.

2.1 Schnittstellen Anfrage

Als erster Schritt musste dazu definiert werden welche Objektstruktur letztlich entstehen soll. Aus der Anfrage soll ein Umfrage Objekt, mehrere zugeordnete Frage Objekte, mehrere Nutzer-Antwort Objekte, welche zu der jeweiligen Frage gehören, der Umfrageteilnehmer als Kontakt und letztlich ein Zuordnungsobjekt, welches die Antworten mit der Umfrage und dem Kontakt verknüpft, entstehen. Die Abbildung verdeutlicht diese Zusammenhänge noch einmal in einem UML-Diagramm.

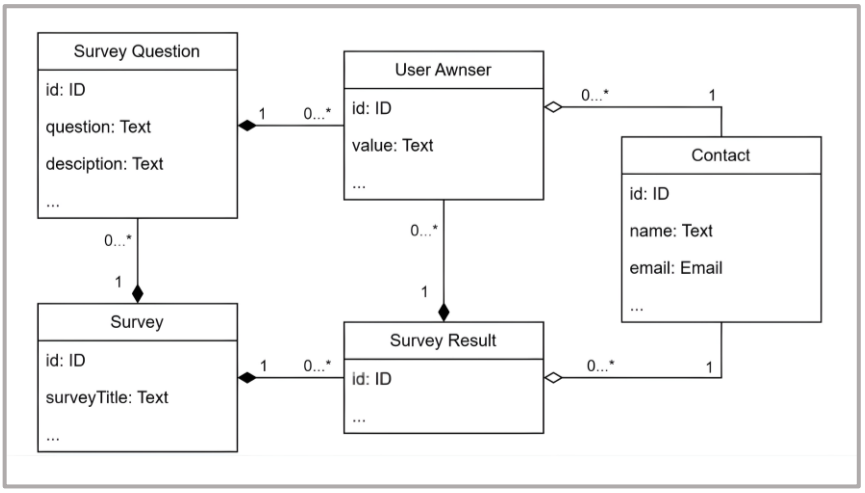

Abbildung 2: initiales UML-Diagramm der zu enstehenden Objekte

Zudem stellt das Diagramm genauere Informationen zu den Beziehungen dar. Ein Umfrageobjekt besteht aus null oder mehreren Fragen. Letztere müssen konkret einer Umfrage zugeordnet sein. Ähnlich verhalten sich das SurveyResult Zuordnungsobjekt und das UserAwnser Objekt. Beide Objekte sind mit ihrem jeweiligen Gegenstück in der Umfrage-Frage-Beziehung verbunden, wobei diese ebenfalls auch jeweils nur einer zugeordnet werden können.

Zwischen dem Kontakt und dem Zuordnungsobjekt, sowie den fragen, ist eine Eins-zu-Mehr-Beziehung vorhanden, da ein Nutzer an mehreren verschiedenen Umfragen teilnehmen und Fragen beantworten soll.

Alle Beziehungen, außer denen zum Nutzer hin, sind als Komposition dargestellt. Damit soll verdeutlicht werden das, sobald eine Umfrage gelöscht wird, alle damit zugehörigen Bestandteile, bis auf den Nutzer, ebenfalls gelöscht werden.

Da dieser Aufbau keine Mehr-zu-Mehr- oder Eins-zu-Eins-Beziehungen aufweist, lässt sich eine solche Struktur in Salesforce problemlos nachbauen. Grundlage dafür sind die Master-Detail- und Lookup-Relationships der Salesforce Plattform. Der Schema Builder, ein von Salesforce vertriebenes Visualisierungstool bestätigt die Implementierung nach UML-Vorlage:

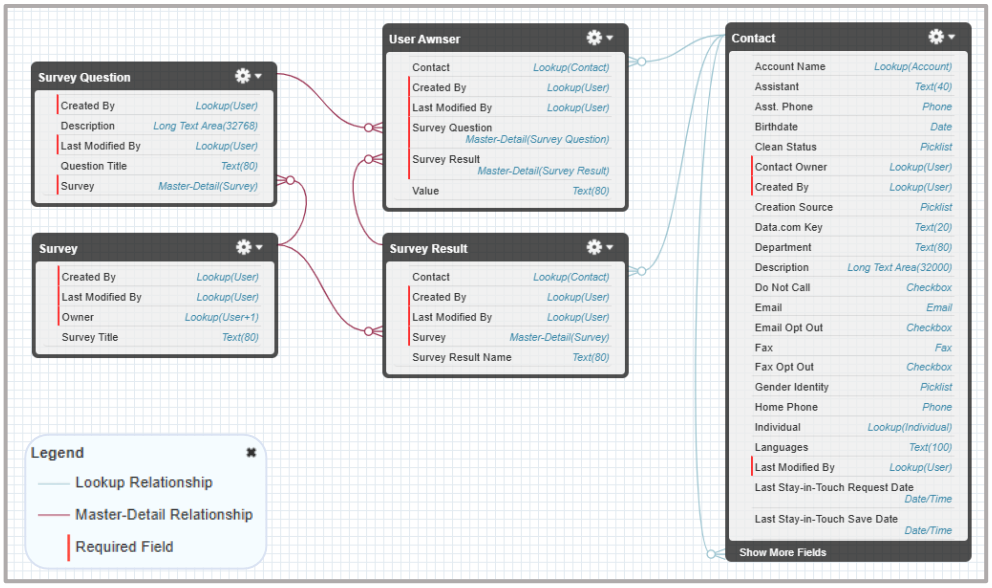

Abbildung 3: Umsetzung des UML-Diagramms in Salesforce

Mit dieser Grundlage kann die JSON für den Body der Anfrage konstruiert werden. Dabei müssen alle Felder des UML-Diagramms vorhanden sein. Zusätzlich werden weitere Meta-Informationen mitgeliefert, um der JSON realitätsnähe mitzugeben, da APIs Dritter standardmäßig mit, für die Salesforce Instanz uninteressanten, Informationen gefüllt sind.

Die für den weiteren Verlauf verwendete JSON liegt der Arbeit im Anhang bei. Diese Enthält neben verschiedenen Datentypen, wie Booleans, Nummern, Texten, Null Einträgen, Zeilenumbrüche und durch das externe System verwaltete IDs, eine verschachtelte Datenstruktur, um einen komplexeren Fall zu repräsentieren.

Die eben vollzogenen Schritte dienen nicht zur Bearbeitung der Aufgabe, dennoch dienen sie als Voraussetzung für die Aufgabenstellung. In der Praxis werden die Schritte durch den Kunden erledigt. Erfolgt dies nicht so muss in Kooperation mit dem Kunden eine Modellierung der Salesforce Objekte und eine Analyse der Schnittstellenanfrage vollzogen werden. Nur durch eine Konkretisierung welche Teile relevant sind und wo hingehören kann eine effektive Integration gewährleistet werden.

Eine Erweiterung der vom Kunden gegebenen Salesforce Objekte Struktur ist denkbar, da Wunschvorstellung oftmals nicht die ideale Art sind externe Daten zu repräsentieren. Dabei ist abzuwägen, ob eine bessere Repräsentation der Daten zu einer besseren Funktionalität der Schnittstelle beiträgt, oder die Vorstellungen des Kunden ausreichend sind. Diese Abwandlung sowie die Schnittstellen Logik wird im folgenden Kapitel thematisiert.

2.2 Aufbau der Schnittstellen Logik

Ist eine Änderung der initialen Vorstellung zielführend so müssen mehrere Aspekte betrachtet werden. Dazu zählen mitunter folgende Fragen:

1. Wie sollen verschiedene Datentypen gespeichert werden?
2. Wie werden Änderungen der Daten verwaltet?

Frage eins erfordert für den in 2.1 beschriebenen Fall, die Berücksichtigung wie die Antworten gespeichert werden. Ein Objekt innerhalb Salesforce besteht aus Feldern mit konkreten Datentypen. Eine dynamische Typ-Veränderung des Feldes Value, welches die Antwort des Nutzers speichern soll ist daher nicht möglich. Eine Zuweisung könnte über Record Types erfolgen, bei denen bestimmte Felder je nach Typ aktiviert bzw. unverfügbar gemacht werden können. So könnte das Objekt intern mehrere Felder wie ValueNumber ValueText ValueMultilineText enthalten, bekommt aber je nach Typ nur das jeweilige entsprechende Feld zugewiesen.

Wichtig ist hierbei, dass diese nur in Richtung der Benutzer der Salesforce Instanz limitiert sind. Administratoren könnten trotzdem nicht freigegebene Felder mit Daten füllen. Grund dafür ist die Implementierung innerhalb der Salesforce Anwendung.[18] Record Types deaktivieren also nicht die Felder basierend auf dem Typ, sondern verstecken sie nur.

[18] Vgl. [Sal24e]

Alternativ könnten mehrere separate Objekte entstehen, zwischen welchen der Endpunkt auswählt und nur der Antwort entsprechende erstellt. Beide Lösungen sind aufgrund vieler benötigter Objekte, oder der fehlenden Limitierung für Entwickler und Admins, nicht wünschenswert. Aufgrund dessen werden innerhalb dieser Arbeit die Antworten der Nutzer standardmäßig als mehrzeiliger Text gespeichert. Daten müssen also durch die Schnittstelle in den dementsprechenden Datentyp umgewandelt werden.

Die zweite Frage beschäftigt sich mit einer möglichen Versionierung der Daten innerhalb der Salesforce Instanz und wirft eine Vielzahl an Folgefragen für die Schnittstelle auf. Damit eine Versionierung stattfinden kann, müssen Daten beispielsweise richtig in der Salesforce Instanz identifiziert werden. Da Drittanwendungen oftmals mit Identifikationsnummern, welche sich von der Salesforce Plattform unterscheiden, agieren, müsste hier mit Salesforce Lösungen gehandelt werden, um diese Ebenfalls zu repräsentieren.

Dabei müsste die Priorisierung von Kundenanforderung oder Repräsentation, der von der Anfrage übermittelten Daten, erneut evaluiert werden. Dennoch beendet eine Korrekte Identifizierung nicht die benötigten Berücksichtigungen für eine Versionierung. Für das gewählte Beispiel könnten weitere Fragen sein:

1. Was passiert, wenn eine Frage gelöscht wird?
2. Was passiert mit den Nutzerantworten auf diese Frage?
3. Was passiert, wenn eine Frage hinzugefügt wird?
4. Was passiert mit Nutzerantwortbögen, die diese Frage nicht beinhalten?
5. Was passiert, wenn sich der gesendete Datentyp einer Frage ändert.

Eine ausgiebige Versionierung, sowie Datentypenverwaltung, würde hierbei den Rahmen der Arbeit deutlich sprengen und soll folglich nicht Bestandteil der Implementation sein. Dennoch sind diese Überlegungen für eine gute Schnittstelle relevant und müssen in der Praxis mit dem Kunden thematisiert werden. Dabei kann eine als einfach erscheinende Schnittstelle deutlich komplexer werden, als initial vermutet.

Für die Implementation wurde sich für eine Änderung der in 2.1 beispielhaft definierten Struktur entschieden. Diese Beispielstruktur sollte mögliche Dopplungen identifizieren und auswerten, sowie Objekte mit korrekten Beziehungen anlegen. Eine Wiederholung von Datensätzen konnte somit minimiert werden. Statt ursprünglich vier Objekten, werden nun zwei Objekte angelegt, um die Umfrage und deren Nutzerantwort zu repräsentieren. Nun werden auf Basis jeder neuen Anfrage nur neue Objekte erstellt.

Die einzige weiterhin bestehende Beziehung, ist zu einem Kontakt, dessen Identifikation durch die Drittanwendung geschieht und damit nicht zum Umfang der Schnittstelle beiträgt. Die Struktur und die Logik der Schnittstelle werden dadurch maßgeblich vereinfacht und schneller. Ein Nachteil einer Implementierung dieses Designs in der Praxis könnte, der von der Instanz erhöhte Speicherverbrauch, sein, da Daten redundant vorliegen.

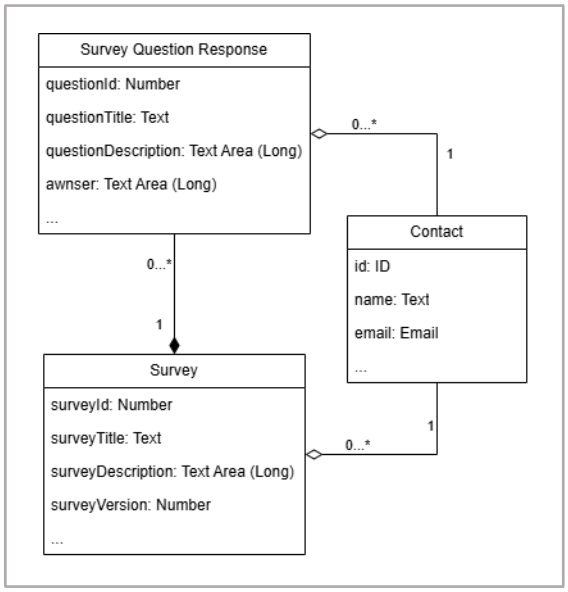

Abbildung 4: überarbeitetes UML-Diagramm mit vereinfachter Struktur

Eine weitere Vereinfachung der Struktur ist nicht möglich. Um Fragen und Umfragen in einem Objekt zu speichern, müsste die Konfiguration über Felder geschehen. Da dabei nicht dynamisch Felder angelegt oder gelöscht werden können, müsste eine feste maximale Anzahl an Fragen vordefiniert werden, was sich für Umfragen aus einer Externen Anwendung schlecht vorgeben lässt.

Da die Struktur, der zu erstellende Objekte, verfeinert und finalisiert wurde, lässt sich die Schnittstellen Logik konzeptionieren. Eine Schnittstelle lässt sich am besten in drei Quellcode-Bestandteile gliedern. Der Klasse für die Bereitstellung, welche die Annotationen enthält, eine Modell-Klasse, in der die Daten der JSON temporär abgelegt werden und letztlich eine Mapper-Klasse, welche die Daten aus dem Modell den eigentlichen Feldern des Objektes zuordnet.

Die Salesforce Implementierung eines JSONs Parsers entpackt alle Daten in Felder dessen Name gleich dem Schlüssel der JSON ist. Wird in der JSON beispielsweise ein Feld Id mitgegeben, wird es standardmäßig dem, durch Salesforce verwaltetem, Feld zugewiesen und bei Erstellung des Objektes für die Cloud automatisch überschrieben.[19]

Vorteile der Quellcode-Dreiteilung bestehen also in der voneinander getrennten Annahme der JSON-Daten und der Erzeugung der Objekte. Damit wird vermieden das Felder, welche deckungsgleiche Namen, aber andere Verwendungszwecke haben, versehentlich gefüllt werden. Auf den Datentyp wird beim Parsen automatisiert geachtet. Enthält die Anfrage ein Feld mit einer Zahl und soll aber im Modell dem Typ String zugordnet werden, so wird die Zahl automatisch in einen String konvertiert.

Da Daten nicht ausgegeben und nur angenommen werden müssen, benötigt die Klasse der Schnittstellenbereitstellung nur zwei Annotationen. Dazu zählen RestResource und HttpPost. Die POST-Http-Methode sendet Daten an einen Server und eignet sich ideal für die Integration. Die Klasse wird in der Implementation also aus nur einer Funktion bestehen, welche verwaltet, was passiert, wenn eine POST-Anfrage an die Schnittstelle geschickt wird. Die Modell-Klasse wird den Body der Anfrage so nah wie möglich repräsentieren. Sie besteht daher aus Unterklassen, Eigenschaften und einer statischen Funktion zum Parsen, welche ein Modell-Objekt erstellt. Die Mapper-Klasse besteht je nach Verwendungszweck aus mehreren statischen Funktionen. In der Praxis werden diese Funktionen je nach Inhalt der Anfrage angesteuert. Diese Unterteilung für die Bearbeitung der Aufgabe nicht relevant, da sich die Anfrage nicht ändert. Basierend auf der gegebenen Anfrage wird sie in der Implementation aus zwei Funktionen bestehen. Eine die das Umfragen Objekte und eine die die Antworten auf Fragen als Objekte erstellt.

2.3 Sicherheitskonzept

Wie bereits in 1.2.2 beschrieben, ist der Kunde gut beraten die Schnittstelle so sicher wie möglich zu implementieren. Während die Herausgabe von Daten an Unbefugte, aufgrund der eben beschriebenen Schnittstellenlogik, keine mögliche Problematik ist, ist die Sicherung trotzdem von großer Relevanz. Da Salesforce auf Cloud Technologie agiert, obliegen Instanzen vorgegebenen Limitationen.

[19] Vgl. [Sal25e]

Eine Vernachlässigung der Schnittstellensicherheit könnte zur übermäßigen Erstellung von Datensätzen ausgenutzt werden. Dadurch werden Limitierungen ausgereizt und wichtige Prozesse können nicht zum gewünschten Zeitpunkt ausgeführt werden.

Bevor die nachfolgenden Kapitel erläutern, für welche Implementierung sich entschieden wurde, werden an dieser Stelle wichtige Hintergründe prägnant erklärt, um die Vorteile grob nachvollziehen zu können.

Für die Entwicklung auf der Salesforce Plattform gibt die Firma zwei wesentliche Entwicklungs-Modelle an. Das eine richtet sich nach unterschieden zwischen Salesforce Instanzen und das andere Modell nach dem Gedanken kleine modulare Pakete zu entwickeln und diese zu installieren.[20] Innerhalb letzterem gibt es verschiedene Versionen an Paketen. Die neueren Second-Generation-Packages (2GP), enthalten viele Verbesserungen und werden deswegen im Paket-Entwicklungs-Modell den älteren vorgezogen. Dazu zählen unteranderem eine flexiblere Versionierung, eine verbesserte Verwaltung und eine verstärkte Modularisierung einzelner Pakete. In der Praxis werden beide Modelle gemischt verwendet, um von den jeweiligen Stärken zu profitieren und die schwächen somit zu minimieren.[21]

2.3.1 Vorteile der External Client App

Diese Entwicklungs-Modelle sind über die Zeit entstanden und wurden seither weiterentwickelt. Innerhalb der Salesforce Plattform existieren derzeitig zwei Rahmenwerke, welche weitverbreitete Autorisierungsprotokolle verwenden. Die Connected Apps sind dabei deutlich früher entstanden und schwerer in moderne Entwicklungsprozesse zu integrieren. Aus diesem Grund besitzen sie aber Funktionalitäten, welche noch nicht in den ECAs verfügbar sind. Dazu zählen beispielsweise einige OAuth Flows oder stark situationsbedingte Optionen. Zudem Profitieren Connected Apps von einer bereits ausgiebigeren Dokumentation.[22]

Zum Zeitpunkt der Arbeit richtet sich die Entscheidung, welches Framework verwendet werden soll, nach dem Verwendungszweck der Schnittstelle. Da ECAs mit dem Package Development Model im Fokus entwickelt wurden, liegen ihre Vorteile in der zeitgleichen Verwendung mit Managed Packages.

---

[20] Vgl. [Sal25g]
[21] Vgl. [Sal25d]
[22] Vgl. [Sal24b]

Wenn der Betreiber der Plattform, welche die Umfrageergebnisse an die Schnittstelle schickt, ein Integrationspaket für seine Kunden bereitstellen möchte, eignen sich ECAs besonders. Am stärksten profitieren nämlich Projekte, in denen ein Paket entsteht, welches mehreren Nutzern zur Verfügung gestellt werden soll. Damit könnten alle Kunden der Umfrageanwendung dieses Paket auf der Salesforce Instanz installieren und Umfrageergebnisse nahtlos auf der Plattform erhalten.

Eine Implementation, unter welcher die Vorteile des 2GP für eine External Client App Anklang finden, würde den Rahmen der Arbeit sprengen und ist folglich nicht Bestandteil der Arbeit. Dennoch scheinen External Client Apps eine vielversprechende Neuerung, wenn sich Salesforce weiterhin bemüht, diese regelmäßig zu verbessern und Funktionalitäten der Connected Apps zu übertragen. Um das neue Framework kennenzulernen, wird in der Implementierung folglich eine lokale ECA angelegt, was die Verwendung ohne Pakete erlaubt.

2.3.2 Auswahl des OAuth Flows

Wie bereits in der Einleitung beschrieben stellt Salesforce, innerhalb der Frameworks, auf den Anwendungsfall abgestimmte Implementierungen des OAuth Protokolls, als Flows zur Verfügung. Für External Client Apps existieren dabei Folgende:

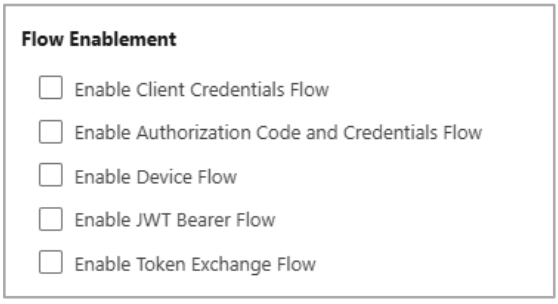

Abbildung 5: UI-Element aus Salesforce zur Konfiguration von Flows

Die Abbildung zeigt nicht alle Flows, welche in einer ECA verfügbar sind. Ein weiterer ist beispielsweise der Web Server Flow, welcher nicht deaktiviert werden kann. Dieser ist ähnlich zum Autorisation Code und wird hauptsächlich verwendet für Nutzer spezifische Anmeldung mit einem Login. Der Device Flow ist für Geräte mit eingeschränkter Eingabe oder Internetfunktionalität. Der Token Exchange Flow ist für Autorisierung mit externen Autorisierungsmethoden.

Der JSON Web Token (JWT) Bearer Flow, sowie der Client Credentials Flow, bieten sich für Maschine-zu-Maschine Verwendungszwecke und somit auch für den Beispielfall idealerweise an. Der JWT verwendet dabei eine etwas komplexere Autorisierung und ist somit in der Lage, Aktionen im Kontext eines Nutzers auszuführen.[23] Der Client Credentials Flow verwendet hingegen einen durch den Instanz-Administrator angegebenen Integrations-Nutzer. Dieser agiert bei Fragen unabhängig vom aufrufenden Nutzer der Schnittstelle. Somit wird kein direkter Nutzerkontext hergestellt.[24]

Für den Beispielfall ist folglich der Client Credentials Flow der zu implementierende. Eine Autorisierung eines Servers, welcher nach Abschluss einer Umfrage, Anfragen an die Schnittstelle schickt, ist somit gegeben. Alle Flows geben einen Access Token als Rückgabe an den Aufrufer. Dieser muss in die Anfragen an die Schnittstelle mitgegeben werden und läuft standardmäßig aus. Eine periodische Erneuerung muss daher von der Anwendung dritter verwaltet werden. Die Aktualisierung des Access Tokens ist folglich nicht Bestandteil der Arbeit.

Damit wurden verschiedene Flows gegenübergestellt und anhand des Verwendungszweckes der passende Flow-Typ ausgewählt. Diese Konzepte sollen innerhalb der nächsten Kapitel implementiert werden.

[23] Vgl. [Sal24d]
[24] Vgl. [Sal24c]

3 Implementierung

Um die Implementierung so verständlich wie möglich zu erläutern, sind die folgenden Kapitel so angeordnet, das kein Verweis auf Quellcode entsteht, welcher noch nicht erklärt wurde. Dabei ist zu beachten das alle Bestandteile für die gewünschte Funktionalität innerhalb einer Salesforce Instanz existieren müssen. Es existiert also kein Kern, der iterativ erweitert werden kann. Stattdessen sind alle Abschnitte Teile des Kerns.

3.1 External Client App

Nicht nur in dieser Arbeit, sondern auch in der Praxis sollte die Erstellung der External Client App der erste Schritt in der Umsetzung einer kundenspezifischen Schnittstelle sein. Nur so kann die Entwicklung iterativ in kleinen Abschnitten getestet und evaluiert werden.

Die Implementation einer ECA ist dabei intuitiv über das User Interface von Salesforce durchsetzbar. Dies geschieht aufgrund des 2GP-Fokus in zwei verschiedenen Bereichen, Einstellungen und Richtlinien. Die Einstellungen dienen dabei als globale Einstellungen, die Richtlinien als Instanz-Einstellungen. Durch diese Unterteilung kann, bei der Verwendung der ECA in einem Paket, besser zwischen benötigten und verwaltbaren Einstellungen unterschieden werden. Diese Unterteilung wird ebenfalls in einer lokalen ECA vorgenommen, auch wenn dabei beide Gruppen als Instanz-Einstellung zählen.

Dazu wurden zuerst grundlegende Informationen, wie der Name der App, eine Beschreibung und das sie als lokale und nicht paketierbare App dient, ausgefüllt. Anschließend wird das OAuth Plugin aktiviert. Unter diesen wurde der API-Scope definiert. Scopes dienen als zusätzliche Restriktion, um die Funktionen, welche die Schnittstelle ausführen darf, zu begrenzen. Zudem wird eine durch Salesforce vorausgesetzte, aber durch den Server-zu-Server Kontext der Schnittstelle unbrauchbare, Callback-URL und der Client Credentials Flow aktiviert. Die restlichen Konfigurationen richten sich nach den Salesforce Standard Konfigurationen des Plugins. Der Client Credentials Flow muss zusätzlich in den Richtlinien aktiviert, und einem Nutzer zugeordnet werden, welcher den, durch die Schnittstelle, definierten Code ausführt. Dieser wurde nach bewährter Salesforce Vorschrift angelegt und hat ausschließlich Zugriff auf die zu erstellenden Objekte.

Die ECA ist somit in der Lage, Anfragen, welche den client_credentials Grant-Type verwenden und die hinterlegte Client ID und Secret an https://instanz.salesforce.com /services/oauth2/token übermitteln, zu Autorisieren.

3.2 Modell-Klasse

Die Modell-Klasse hat die Primärfunktion die Daten des JSON-Bodys der Anfrage anzunehmen und in ein verarbeitbares Format zu bringen. Wie im Konzept bereits erklärt, besitzt die Klasse einen einfachen Aufbau, welcher die Daten der JSON repräsentieren, soll:

```
1. public with sharing class SurveyModel {
2.     public SurveyData surveyData;
3.     public Metadata metadata;
4.     public class SurveyData {
5.         public Integer id;
6.         public String name;
7.         public Integer revision;
8.         public String description;
9.         public boolean completed;
10.        public String participant;
11.        public List<QuestionData> questionData;
12.    }
13.    public class QuestionData {
14.        public Integer qId;
15.        public String title;
16.        public String description;
17.        public String value;
18.    }
19.    public class Metadata {
20.        public String requestId;
21.        public String timestamp;
22.        public String source;
23.        public String version;
24.    }
25.    public static SurveyModel parse(String json){
26.        return (SurveyModel) System.JSON.deserialize(json, SurveyModel.class);
27.    }
28. }
```

Abbildung 6: Quellcode der Modell-Klasse

Um aus dem JSON-Body verschachtelte Objekte, welche mit geschweiften Klammern dargestellt werden, richtig entpacken zu können, braucht die Klasse mehrere Unterklassen. Das Erstellen von Objekten dieser Klasse geschieht über die statische parse Methode und nicht wie normalerweise über einen Konstruktor. Grund dafür ist der Fakt das ein Konstruktor keinen Rückgabetyp enthalten darf und ausschließlich für die richtige Zuweisung an die Eigenschaften der Klasse fungiert. Diese geschieht hier aber nicht manuell, sondern automatisch durch den Parser. Die Konvertierung verschiedener Datentypen als Antworten auf eine Frage, wird wie im Konzept erläutert über den JSON-Parser von Salesforce innerhalb der Modell-Klasse verwaltet. Durch den Datentyps String des value Feldes der SurveyData Klasse, werden alle beliebigen Antworten beim Parsen in Strings konvertiert.

Die Konvertierung von einem String in die verschiedenen Arten von Textfeldern innerhalb der Salesforce Plattform, wie Text, Text Area und Text Area (Long) geschieht beim Laden in die Salesforce Instanz und benötigt daher nicht die Verwaltung durch einen Entwickler.

3.3 Mapper-Klasse

Als nächstes wird die Mapper-Klasse implementiert, welche nun ein Objekt der SurveyModel Klasse verarbeitet und dessen Eigenschaften den jeweiligen Feldern des Salesforce Objektes zuordnet. Diese entstehenden Salesforce Objekte werden dann wieder an den Aufrufer der Methode übergeben.

```
 1. public with sharing class SurveyMapper {
 2.     public static Survey__c createSurvey(SurveyModel requestData){
 3.         Survey__c newServey = new Survey__c(
 4.             Name = requestData.surveyData.name,
 5.             Survey_Id__c = requestData.surveyData.id,
 6.             Survey_Version__c = requestData.surveyData.revision,
 7.             Survey_Description__c = requestData.surveyData.description,
 8.             Contact__c = requestData.surveyData.participant
 9.         );
10.         return newServey;
11.     }
12.
13.     public static List<Survey_Question_Response__c> createSurveyResponses(SurveyModel
requestData, Survey__c surveyObject){
14.         List<Survey_Question_Response__c> responses = new
List<Survey_Question_Response__c>();
15.         for (SurveyModel.QuestionData question : requestData.surveyData.questionData) {
16.             responses.add(new Survey_Question_Response__c(
17.                 Name = question.title,
18.                 Question_Id__c = question.qId,
19.                 Question_Description__c = question.description,
20.                 Contact__c = requestData.surveyData.  ,
21.                 Survey__c = surveyObject.Id,
22.                 Awnsered_Value__c = question.value
23.             ));
24.         }
25.         return responses;
26.     }
27. }
```

Abbildung 7: Quellcode der Mapper-Klasse

Die zweite statische Funktion weicht von diesem Konzept jedoch leicht ab. Sie ist nicht ausschließlich vom SurveyModel abhängig, sondern erwartet einen zusätzlichen Parameter eines Survey__c Objekts. Durch dieses kann eine Beziehung zum Umfragen Objekt hergestellt werden. Der Rückgabetyp der Funktion ist dabei eine Liste, und erstellt folglich alle in der Anfrage enthaltenen Survey Question Response Objekte. Dabei wird über die questionData Liste des Modells geschleift und basierend auf jedem Eintrag, ein neues Survey Question Response Objekt der Liste hinzugefügt.

3.4 Apex REST-Klasse

Zuletzt muss die Apex REST-Klasse implementiert werden. Die Abbildung 8 zeigt dabei die Klasse nicht vollständig.

```
 1. @RestResource(urlMapping='/Survey/*')
 2. global with sharing class ApexRestApi {
 3.
 4.     @HttpPost
 5.     global static void insertObjects() {
 6.         try{
 7.             SurveyModel requestData =
SurveyModel.parse(RestContext.request.requestBody.toString());
 8.             Survey__c surveyObj = SurveyMapper.createSurvey(requestData);
 9.             insert surveyObj;
10.             List<Survey_Question_Response__c> responseObjs =
SurveyMapper.createSurveyResponses(requestData,surveyObj);
11.             insert responseObjs;
…
```

Abbildung 8: Ausschnitt aus der API-Klasse welcher die Funktionen der anderen aufruft

In ihr ist lediglich zu sehen wie die Klasse die in 1.2.1 beschriebenen Annotationen einsetzt und die Methoden der anderen Klassen implementiert. Zur Erstellung der Objekte, über eine autorisierte POST-Anfrage an die URL https://instanz.salesforce.com/services/apexrest/Survey/*, wird der Body der Anfrage erstmal durch die statische Methode des Modells geparst und in einer Instanz der Klasse gespeichert. Anschließend wird durch den Mapper ein Umfrage Objekt, basierend auf dem Modell, erstellt. Dieses wird direkt nach der Erstellung in die Datenbank geladen um das Id Feld, für die Beziehung, zu befüllen. Nachdem dieses angelegt wurde, kann aus dem Modell und dem Umfrage Objekt die Fragen erstellt werden. Diese werden anschließend ebenfalls in die Datenbank geladen.

Im Anhang findet sich die Gesamte Klasse. Dabei enthält sie Antworten, welche die Schnittstelle auf Anfragen schickt und implementiert eine Fehlerbehandlung, welche eine private Unterklasse, ein Enum und eine Hilfsmethode erfordert.

4 Fazit

Ziel dieser Arbeit war es, eine Simulierte externe Anwendung in eine Salesforce Instanz zu integrieren. Dabei dienen die Ergebnisse der Arbeit der Vorbereitung auf aktuelle Markttrends. Dazu musste eine benutzerdefinierte Schnittstelle unter Salesforce entstehen. Diesbezüglich wurden Vorarbeiten, welche mit dem Kunden geklärt werden müssen, erläutert, neue Salesforce Technologien prägnant erhoben, sowie bewährte Vorverfahren implementiert.

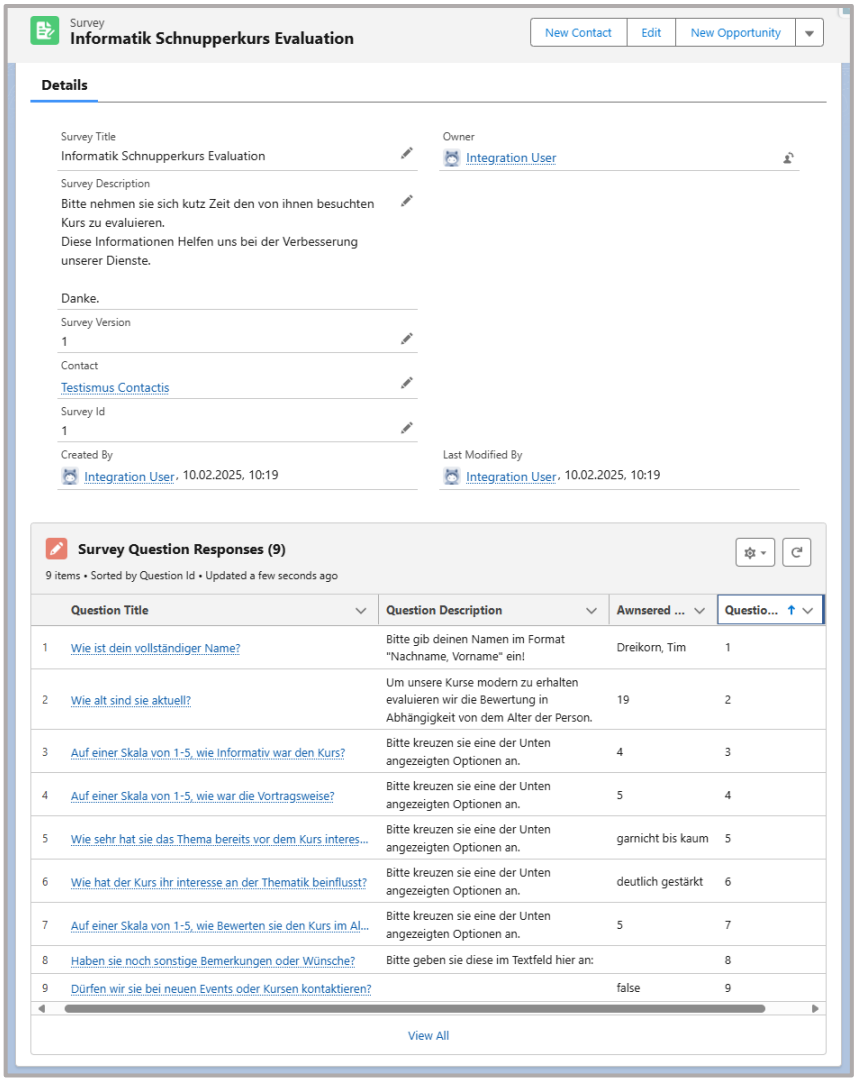

Abbildung 9: Übersicht über die durch die Schnittstelle erstellten Daten

Die innerhalb der Arbeit verwendete Vorgehensweise kann dabei bei nachfolgenden Projekten ebenfalls umgesetzt werden, um zufriedenstellende Lösung zu erzielen. Dazu zählen eine Voranalyse der Zielstellung und den Wünschen des Kunden, um die zu erstellende Objekte und den für den Anwendungsfall spezifischen Flow auszuwählen, sowie die Quellcode Struktur der Schnittstelle. Insofern dabei mehrere Schnittstellen entstehen sollen, können die privaten

Bestandteile der Apex REST-Klasse ausgelagert werden in API-Helper Klassen, welche diese für alle Schnittstellen standardisiert bereitstellen. Die Erläuterung, der Vorteile von ECAs, muss nicht nachfolgend angewandt werden. Stattdessen sollten Salesforce-Entwickler bemüht daran sein, selbständig neue Technologien der Plattform zu kennen. Das Wissen ist essenziell für eine effektive Kundenberatung und Auswahl der Umsetzungen. Die in der Arbeit vermittelten Arbeitsschritte lassen sich außerdem um die Entwicklung von Codetests vor der Implementierung erweitern, um manuelle Überprüfungen auf Korrektheit zu minimieren.

Die der derzeitig steigende Interkonnektivität von Anwendungen, kann somit verfolgt werden. Basierend auf Kundenanforderungen muss zwischen Connected Apps oder ECAs entscheiden, sowie der Korrekte Flow ausgewählt werden. Der SOLL-Zustand wurde erreicht.

Literaturverzeichnis

[Exp23] Exploding Topics: Most Visited Websites In The World (November 2024), 2023, https://explodingtopics.com/blog/most-visited-websites, Abgerufen am: 08.01.2025

[Gee22] GeeksForGeeks: Know the Difference Between REST API and RESTful API, GeeksforGeeks, 2022

[Mil23] Miller, C.: From SOAP to REST: Tracing The History of APIs - Treblle, Treblle Blog, 2023

[NLB21] Neumann, A.; Laranjeiro, N.; Bernardino, J.: An Analysis of Public REST Web Service APIs, IEEE Transactions on Services Computing, 2021, S. 957 – 970

[Nur22] Nurture an Engaged and Satisfied Workforce | Vantage Circle HR Blog: Die 35 besten Zitate zur Teamarbeit inspirieren zur Zusammenarbeit, 2022, https://www.vantagecircle.com/de/blog/zusammenarbeit/, Abgerufen am: 08.01.2025

[Pos24] Postman: 2024 State of the API Report, 2024, https://voyager.postman.com/doc/postman-state-of-the-api-report-2024.pdf, Abgerufen am: 21.02.2025

[RBD+16] Rodríguez, C.; Baez, M.; Daniel, F., et al.: REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. in Bozzon, A. (Hrsg.): Web engineering, Springer, Cham, Heidelberg, S. 21 – 39

[Sal24a] Salesforce: Authorize Apps with OAuth, 2024, https://help.salesforce.com/s/articleView?id=xcloud.remoteaccess_authenticate.htm&type=5, Abgerufen am: 19.02.2025

[Sal24b] Salesforce: Comparison of Connected Apps and External Client Apps Features, 2024, https://help.salesforce.com/s/articleView?id=xcloud.connected_apps_and_external_client_apps_features.htm&type=5, Abgerufen am: 19.02.2025

[Sal24c] Salesforce: OAuth 2.0 Client Credentials Flow for Server-to-Server Integration, 2024, https://help.salesforce.com/s/articleView?id=xcloud.remoteaccess_oauth_client_credentials_flow.htm&type=5, Abgerufen am: 19.02.2025

[Sal24d] Salesforce: OAuth 2.0 JWT Bearer Flow for Server-to-Server Integration, 2024, https://help.salesforce.com/s/articleView?id=xcloud.remoteaccess_oauth_jwt_flow.htm&type=5, Abgerufen am: 19.02.2025

[Sal24e] Salesforce: Tailor Business Processes to Different Record Types Users, 2024, https://help.salesforce.com/s/articleView?id=platform.customize_recordtype.htm&type=5, Abgerufen am: 19.02.2025

[Sal24f] Salesforce: Analyst Reports, 2024, https://www.salesforce.com/company/recognition/analyst-reports/, Abgerufen am: 21.03.2024

[Sal25a] Salesforce: Apex REST Basic Code Sample | Apex Developer Guide | Salesforce Developers, 2025, https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_rest_code_sample_basic.htm, Abgerufen am: 17.02.2025

[Sal25b] Salesforce: API Library | Salesforce Developers, 2025, https://developer.salesforce.com/docs/apis#browse, Abgerufen am: 18.02.2025

[Sal25c] Salesforce: Developer Centers | Salesforce Developers, 2025, https://developer.salesforce.com/developer-centers, Abgerufen am: 18.02.2025

[Sal25d] Salesforce: Comparison of First- and Second-Generation Managed Packages | Second-Generation Managed Packaging Developer Guide | Salesforce Developers, 2025, https://developer.salesforce.com/docs/atlas.en-us.pkg2_dev.meta/pkg2_dev/sfdx_dev_dev2gp_comparison.htm, Abgerufen am: 19.02.2025

[Sal25e] Salesforce: JSON Class | Apex Reference Guide | Salesforce Developers, 2025, https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_class_System_Json.htm#apex_System_Json_methods, Abgerufen am: 19.02.2025

[Sal25f] Salesforce: Salesforce Developers, 2025, https://developer.salesforce.com/docs/platform/mobile-sdk/guide/oauth-scope-parameter-values.html, Abgerufen am: 19.02.2025

[Sal25g] Salesforce: Salesforce Developers, 2025, https://developer.salesforce.com/docs/platform/sfvscode-extensions/guide/development-models.html#, Abgerufen am: 19.02.2025

[Sur17] Surowiecki, J.: Die Weisheit der Vielen, Plassen Verlag, Kulmbach, 2017

[TES+15] Tilkov, S.; Eigenbrodt, M.; Schreier, S., et al.: REST und HTTP, 3., aktualisierte und erweiterte Auflage, dpunkt.Verlag, Heidelberg, 2015

Anlagenverzeichnis

```
{
    "surveyData" :{
        "id" : 1,
        "name" : "Informatik Schnupperkurs Evaluation",
        "revision" : 1,
         "description" : "Bitte nehmen sie sich kutz Zeit den von ihnen besuchten Kurs zu evaluieren.\nDiese Informationen Helfen uns bei der Verbesserung unserer Dienste.\n\nDanke.",
        "completed" : true,
        "participant" : "003Qy00000Hj8NpIAJ",
        "questionData" : [{
            "qId" : 1,
            "title" : "Wie ist dein vollständiger Name?",
            "description" : "Bitte gib deinen Namen im Format \"Nachname, Vorname\" ein!",
            "value" : "Dreikorn, Tim"
        },{
            "qId" : 2,
            "title" : "Wie alt sind sie aktuell?",
             "description" : "Um unsere Kurse modern zu erhalten evaluieren wir die Bewertung in Abhängigkeit von dem Alter der Person.",
            "value" : 19
        },{
            "qId" : 3,
            "title" : "Auf einer Skala von 1-5, wie Informativ war den Kurs?",
            "description" : "Bitte kreuzen sie eine der Unten angezeigten Optionen an.",
            "value" : 4
        },{
            "qId" : 4,
            "title" : "Auf einer Skala von 1-5, wie war die Vortragsweise?",
            "description" : "Bitte kreuzen sie eine der Unten angezeigten Optionen an.",
            "value" : 5
        },{
            "qId" : 5,
            "title" : "Wie sehr hat sie das Thema bereits vor dem Kurs interessiert?",
            "description" : "Bitte kreuzen sie eine der Unten angezeigten Optionen an.",
            "value" : "garnicht bis kaum"
        },{
            "qId" : 6,
            "title" : "Wie hat der Kurs ihr interesse an der Thematik beinflusst?",
            "description" : "Bitte kreuzen sie eine der Unten angezeigten Optionen an.",
            "value" : "deutlich gestärkt"
        },{
            "qId" : 7,
            "title" : "Auf einer Skala von 1-5, wie Bewerten sie den Kurs im Allgemeinen?",
            "description" : "Bitte kreuzen sie eine der Unten angezeigten Optionen an.",
            "value" : 5
        },{
            "qId" : 8,
            "title" : "Haben sie noch sonstige Bemerkungen oder Wünsche?",
            "description" : "Bitte geben sie diese im Textfeld hier an:",
            "value" : null
        },{
            "qId" : 9,
            "title" : "Dürfen wir sie bei neuen Events oder Kursen kontaktieren?",
            "description" : "",
            "value" : false
        }]
    },
    "metadata" : {
        "requestId" : "abc123456789",
        "timestamp" : "2025-01-21T14:30:00Z",
        "source" : "web-client",
        "version" : "1.0"
    }
}
```


Anlage 1: JSON-Body der Schnittstellenanfrage

```apex
@RestResource(urlMapping='/Survey/*')
global with sharing class ApexRestApi {

    @HttpPost
    global static void insertObjects() {
        try{
            SurveyModel requestData =
SurveyModel.parse(RestContext.request.requestBody.toString());
            Survey__c surveyObj = SurveyMapper.createSurvey(requestData);
            insert surveyObj;
            List<Survey_Question_Response__c> responseObjs =
SurveyMapper.createSurveyResponses(requestData,surveyObj);
            insert responseObjs;
            setResponse(
                200,
                new ResponseBodyData(Status.SUCCESS,''),
                'application/json'
            );
            return;
        } catch (JSONException exc) {
            setResponse(
                400,
                new ResponseBodyData(Status.FAILED,exc.getMessage()),
                'application/json'
            );
            return;
        } catch (Exception exc) {
            setResponse(
                500,
                new ResponseBodyData(Status.FAILED,exc.getMessage()),
                'application/json'
            );
            return;
        }
    }

    private static void setResponse(
        Integer statusCode,
        ResponseBodyData responseData,
        String contentType
    ){
        RestContext.response.statusCode = statusCode;
        RestContext.response.responseBody = Blob.valueOf(JSON.serialize(responseData));
        RestContext.response.addHeader('Content-Type', ContentType);
    }

    private class ResponseBodyData {
        public Status status;
        public String message;
        public ResponseBodyData(Status status, string message) {
            this.status = status;
            this.message = message;
        }
    }

    private enum Status {
        SUCCESS,
        FAILED,
        ERROR
    }

}
```

Anlage 2: Vollständige API-Klasse mit Fehlerbehandlung und Serverantworten