- Hochschule für angewandte Wissenschaften –

Seminar: Verteilte Enterprise Anwendungen mit Java EE

Dozent: Michael Theis

Seminararbeit

APEX – „Java mit Salesforce“

Stefan Spirkl

Sommersemester 2018

Matrikelnummer: 52623616

Abstract:

Moderne Cloud CRM Systeme wie Salesforce bieten einen enormen und kaum noch zu überschauenden Funktionsumfang. Wer diese enorme Funktionsbreite jedoch sinnvoll nutzen möchte, muss die Salesforce Organisation erst aufwändig auf seine Unternehmensprozesse abstimmen. Ab einer gewissen Komplexität der Anforderungen kommt man dann nicht mehr umhin, auch APEX – die Salesforce Variante von Java – einzusetzen. Für die Entwicklung von eigener Anwendungen auf der Force.com Plattform ist Apex genauso mächtig wie alternativlos. Die vorliegende Seminararbeit soll dem interessierten Leser dazu einen ersten Einblick geben.

Inhaltsverzeichnis

I. Motivation

1. Braucht's das?

Bei der Fülle an verschiedenen Skript- und Programmiersprachen, verschiedenen proprietären Systemen und Entwicklungsparadigmen und einer stets wechselnden Techstack-Landschaft kann ohnehin oft überlastete Informatiker kritisch hinterfragen, ob es wirklich nötig ist, sich auch noch mit Apex zu beschäftigen.

Schließlich gibt es mit der AppExchange[1] ja eine große Auswahl an fertigen Apps die in die eigene Salesforce Org geladen werden können. Mit dem Process Builder[2] hat Salesforce zudem ein mächtiges Werkzeug geschaffen um auch ohne Apex komplexe Geschäftsanwendungen umzusetzen.

Apex ist als nur für die Entwicklung auf der proprietären (Sales)Force(.com) Plattform geeignet – davon abgesehen kann man mit Apex nichts machen [3]. Der Overlap zur täglichen Arbeit zu anderen Sprachen ist gering. Sich mit Apex zu beschäftigen, heißt immer auch, sich mit Salesforce zu beschäftigen.

Auch wenn das sicher keiner zugibt - ein Apex-Entwickler ist ungefähr zu gleichen Teilen Salesforce-Administrator wie Java-Entwickler. [4]

Eine Notwendigkeit, sich ohne konkrete Motivation mit Apex zu beschäftigen, kann klar verneint werden. Am Ende des Tages sind Salesforce-Entwicklungskentnisse eine Spezialisierung, die man wollen oder zumindest bewusst in Kauf nehmen muss.

2. Warum Apex lernen?

Nach der etwas kritischeren Einleitung, sollen nun Gründe genannt werden, sich mit Force.com zu beschäftigen und etwas tiefer in die Apex-Welt einzusteigen.

Denn in der Salesforce-Entwicklung ist es meiner Erfahrung nach immer noch möglich, mit wenig Aufwand, echte Verbesserungen für ein Unternehmen zu bewirken und den Alltag von Mitarbeitern leichter zu machen.

Dem Apex Entwickler wird daher von Mitarbeitern und Geschäftsführung erhöhte Dankbarkeit für technisch triviale Dinge entgegengebracht. Dies geht einher mit einem hohen „Business Value" pro Codezeile und oft kleinen – aber für das Unternehmen sehr wertvollen – Anwendungen.

Durch die Konzentration vieler Unternehmen auf die Force.com Plattform ergibt sich ein stark gestiegener Bedarf an Entwicklern und Administratoren [5]. Der Nachfragestau bei der Umsetzung von Komponenten sorgt für solide Gehälter auch bei „mittelmäßigen" im Bereich um die 60.000-80.000€ [6] in Deutschland(München), in den USA(Boston) etwa 95.000-130.000$ [7].

Wer des Büroalltags Müde geworden ist, hat gute Chancen, mit einer eigenen App auf der AppExchange passives Einkommen zu generieren:

> *"There are 2,948 apps listed on the public AppExchange. Of these apps, 1,650 are not freebies. $1.5B in license revenues were generated via AppExchange transactions for the ecosystem last year. Hence >$900K USD was generated per paid app annually. „* --Stephen Cummins [8]

Der Durchschnittliche Umsatz pro App liegt also knapp über 900.000$ und damit deutlich über Consumer Apps wie dem iOS Appstore (8.100$) oder Android Playstore (4.900$) [9]. Die Entwicklung

von Business-Apps ist nicht nur technisch oft interessanter, sondern bedient auch eine weit größere Nachfrage im Vergleich zu Consumer-Apps.

Bei diesen Zahlen wurde die 15% Commission von Salesforce noch nicht abgezogen. Der Median der Einnahmen wird zudem erwartungsgemäß unter dem Durchschnittswert liegen. Die Konzentration auf einige wenige „Top Apps“ dürfte bei Salesforce jedoch deutlich weniger ausgeprägt sein, als bei Apps für Privatanwender.

Es gibt im Salesforce App-Ökosystem noch unbesetzte Nischen und Bedarf an branchenspezifischen Lösungen und viel Potential, für gute Apps auch Kunden zu finden. Anders als bei Consumer-Apps sind hier auch die Deckungsbeiträge im Produkt gegeben, um potentielle Kunden aktiv anzusprechen und diese in einem Vertriebsprozess von dem Business Value der App zu überzeugen.

II. Grundlagen

1. Was ist Apex?

> “Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on Salesforce servers in conjunction with calls to the API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. [10]

Im Wesentlichen ist Apex also zusammen mit Visualforce eine Möglichkeit, eine Salesforce Umgebung anzupassen und um eigene Komponenten zu erweitern. Apex ist sehr stark an Java angelehnt. Jede Aktion die ein Salesforce Benutzer ausführen kann, kann auch mit Apex durchgeführt werden und mehr.

Apex wird benötigt um Business-Logik bzw. generell Serverseitigen Code auszuführen. Apex ist abzugrenzen von der Syntax von Visualforce und Aura, die eher der von HTML/XML ähnelt und die Präsentations-Schicht innerhalb einer Salesforce Anwendung abbildet.

Die wesentlichen Elemente wie Aufbau von Klassen, Kontrollstrukturen und Tests sind in Apex identisch zu Java. Entsprechend müssen neue Apex Entwickler nur unwesentlich „eine neue Programmiersprache lernen“, sondern müssen vor allem das Force.com Datenmodell und die Eigenheiten der Plattform verstehen.

2. Was ist die Force.com Plattform?

> *Force.com lets developers rapidly create and deploy trusted cloud applications that are rock solid, secure, and scalable without having to worry about provisioning hardware or application stacks. To help you go faster, Force.com delivers out-of-the-box tools and services to automate your business processes, integrate with external applications, deliver mobile experiences and more. [11]*

Kurz gesagt ist Force.com das PaaS (Platform as a Service) System für alle Salesforce Anwendungen. Es gibt keine Möglichkeit als nicht-Salesforce Anwendung darauf aufzusetzen und es gibt keine Möglichkeit als Salesforce Anwendung nicht darauf aufzusetzen.

Die Force.com Plattform ist also die Cloud-Plattform auf der Apex ausgeführt wird und stellt Infrastruktur für Salesforce selbst und alle dafür entwickeltem Anwendungen bereit.

Zu den Vorteilen gehört, direkt mit den Salesforce Daten arbeiten zu können, durch die vorgegebene Struktur mit weniger Code mehr zu erreichen und sich um einige Security Themen nicht kümmern zu müssen. Außerdem können die Apps auf andere Salesforce Orgs übertragen werden. [12]

Die Unterscheidung zwischen „Salesforce“ und „Force.com“ ist eher Marketing- als technischer Natur, da Salesforce mittlerweile nicht nur reines CRM anbietet und auch die non-CRM Anwendungen auf der Force.com Plattform aufbauen (ausgenommen Zukäufe / von Salesforce aufgekaufte Technologieprodukte). Daher kann man „für Salesforce entwickeln“ und „auf Force.com entwickeln“ synonym verwenden.

3. Entwicklungsumgebungen

Nur sehr kurz sei auf einige Entwicklungsumgebungen für Apex eingegangen, da dies wohl ein typisches Thema für den Einstieg sein dürfte.

Name	Kurzbeschreibung	Website
 Mavensmate	Speziell für Salesforce entwickelter Code-Editor, wird leider nicht mehr aktiv weiterentwickelt, dennoch sehr beliebt.	https://github.com/joeferraro/MavensMate
 Eclipse	Force.com Plugin für die bekannte eclipse IDE.	https://developer.salesforce.com/docs/atlas.en-us.eclipse.meta/eclipse/ide_install.htm
 Developer Console	Browserbasierte, von Salesforce bereitgestellte IDE mit vielen Features, für den Anfang ausreichend.	https://www.yoursalesforceorg.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage

4. Was sind die Unterschiede von Apex zu Java? (Auswahl) [13]

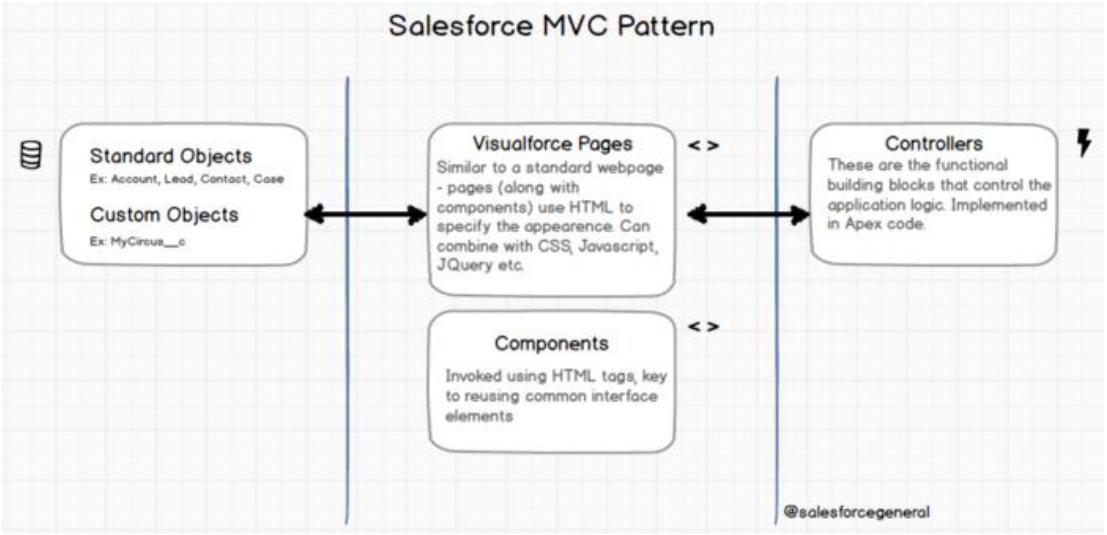
Item	Java	Apex
Packages	Ja	Kein Package System
Case sensitive	Groß-Klein Schreibung wird beachtet	Groß-Klein Schreibung wird nicht beachtet
Limits	Keine Begrenzung der maximalen Zahl an Zeilen und Ausführzeit; Limits sind bis auf absolute Ausnahmen „Soft“ (z.B. Hauptspeicher ausgeschöpft -> Nutzung Swap Partition -> Programm bremst, aber läuft weiter)	Governor Hard Limits z.B. maximal 3 Millionen ausgeführte Codezeilen pro Run (durchaus erreichbar z.B. in $O(N^2)$).
Switch Kontrollstruktur	Ja	Nicht vorhanden, Rückgriff auf else if erforderlich.

III) Technik

1. Model View Controller

Salesforce verwendet ein klassisches MVC-Modell, bei dem eine Anwendung in die drei Teile UI, Logik und Daten aufgeteilt wird.

Die folgende Skizze veranschaulicht dies [14]:



a. Model

In der linken Spalte sehen wir die Datenhaltung (Model) – das ist schlicht die Datenbank unserer Salesforce Org. Sehr theoretisch betrachtet könnte man diese Schicht mit einigem Aufwand auch tauschen und eine eigene Datenbank einsetzen, in der Praxis wird man diese einfach als von Force.com gegeben betrachten und lediglich eigene Custom Objects definieren.

Das Model enthält sowohl die Standardobjekte von Salesforce – die „üblichen Verdächtigen" im CRM, wie Account oder Contact. Eigene Objekte (Costum Objects) werden in fast allen Fällen benötigt werden, um den Anforderungen des eigenen Geschäftsmodells Rechnung zu tragen oder um Relationen zwischen anderen Feldern abzubilden.

Ein Versicherungsunternehmen mag zum Beispiel ein Custom Objekt „Versicherungstyp" haben, in dem zum Beispiel ein Element „Krankenversicherung Beamte" mit Fields wie Monatsbeitrag, Vertragsbeginn, Mindestlaufzeit und Kündigungsfrist abgelegt ist.

Daneben hat das Unternehmen möglicherweise ein Custom Objekt „AccountVersicherungstyp", welches die Relation zwischen einem „Account" (einem Kunden des Versicherungsunternehmens) und einem „Versicherungstyp" (welcher Vertrag) abbildet – also welcher Kunde welchen Vertrag abgeschlossen hat.

Custom Objekts werden stets mit dem Suffix __c versehen, um sie von den Standardobjekten in Salesforce zu unterscheiden. [15]

b) View

In der mittleren Spalte sehen wir den Presentation-Layer im Salesforce-MVC. Dieser besteht je nach Implemtierung aus einer Lightning-Komponente (.CMP) oder einer VisualForce Page (.VFP).

Darin enthalten sind die Elemente, die für den Endbenutzer in der Salesforce Org sichtbar sein wollen. Die Syntax ist dem Aura Framework[16] entnommen und ist im Wesentlichen wie XML/HTML, jedoch erweitert um Foreach Schleifen und die Übernahme von Variablen aus dem Controller mit der {!variableName} Syntax.

Typische Elemente in einem Salesforce View sind Listen von Objekten (z.B. Accounts), Textausgaben oder Charts. Häufige Eingabeelemente sind Inputfelder, Checkboxes und Dropdowns.

Bei der Wahl zwischen Lightning-Komponente und Visualforce Page ist im Regelfall die Lightning-Komponente vorzuziehen.

Die Entwicklung im Visualforce Page Konstrukt ist dann sinnvoll wenn auch Salesforce Classic Endnutzer unterstützt werden sollen oder eine Ausführung im iFrame und damit eingehende Isolierung des im View ausführbaren Javascript Code vom regulären Salesforce DOM erzwungen werden soll. Da der Iframe auf einer seperaten Subdomain ausgeführt wird, wird dieser durch die Cross-Domain Policy stark von der eigentlichen Seite entkoppelt. Auch ist es dadurch möglich, z.B. in einer Sidebar einen full reload durchzuführen, während der Nutzer mit dem Rest der Seite noch weiterarbeiten kann.

Häufig sind diese „Features" aber eher ein Nachteil, weshalb man sich standardmäßig für eine Lightning-Komponente entscheiden sollte.

c) Controller

Im Controller erarbeiten wir die eigentliche Geschäftslogik in Apex. Der Controller liest Eingaben vom Frontend, verarbeitet diese und sendet ggf. eine Rückmeldung zurück an den Nutzer. Ein Controller kann von mehreren Komponenten oder Visualforce-Pages verwendet werden.

Der Controller ruft mittels SOQL Objekte aus der Datenbank ab und führt mit DML (Data Modification Language) Updates und Inserts in der Datenbank aus.

2. APEX-Syntax [17]

Nachstehend ein Beispiel für einen Controller in Apex:

```
public with sharing class anyController {
    public String anyVariable { get; set; }
    public Id getAccountID() {
        return ApexPages.currentPage().getParameters().get('id');
    }
}
```

[…]

3. SOQL und DML

Aus nicht bekannten Gründen hat sich Salesforce dagegen entschieden, einfach SQL als Datenbanksystem zu übernehmen. Stattdessen wurde die eigene deklarative Abfragesprache SOQL(Salesforce Object Query Language) ins Leben gerufen.

SOQL vermisst einige Funktionen, die in SQL üblich und beliebt sind, darunter Wildcards, Unions, verschachtelte Abfragen mit IN und sogar manche JOINs. [18]

Einfache Dinge wie zum Beispiel eine Abfrage über verschiedene „Tabellen“ (Objekte) hinweg mit verschiedenen Feldtypen sind dadurch nicht mehr ohne weiteres möglich.

Ein SOQL Query kann wie folgt im Apex Controller ausgeführt werden:

```
List<AccountRelation__c> leftColumnList = [SELECT account1Id__c FROM
AccountRelation__c WHERE Account2__c = :accountId];
```

SOQL ist zudem eine reine Abfragesprache und ermöglicht ausschließlic Lesezugriff. Um auf die Datenbank zu schreiben muss wiederrum auf die DML (Data Manipulation Language) [19] zurückgegriffen werden. DML unterstützt die Anweisungen insert, update, merge, delete und rollback und bedient damit ausschließlich Schreibzugriffe auf die Datenbank.

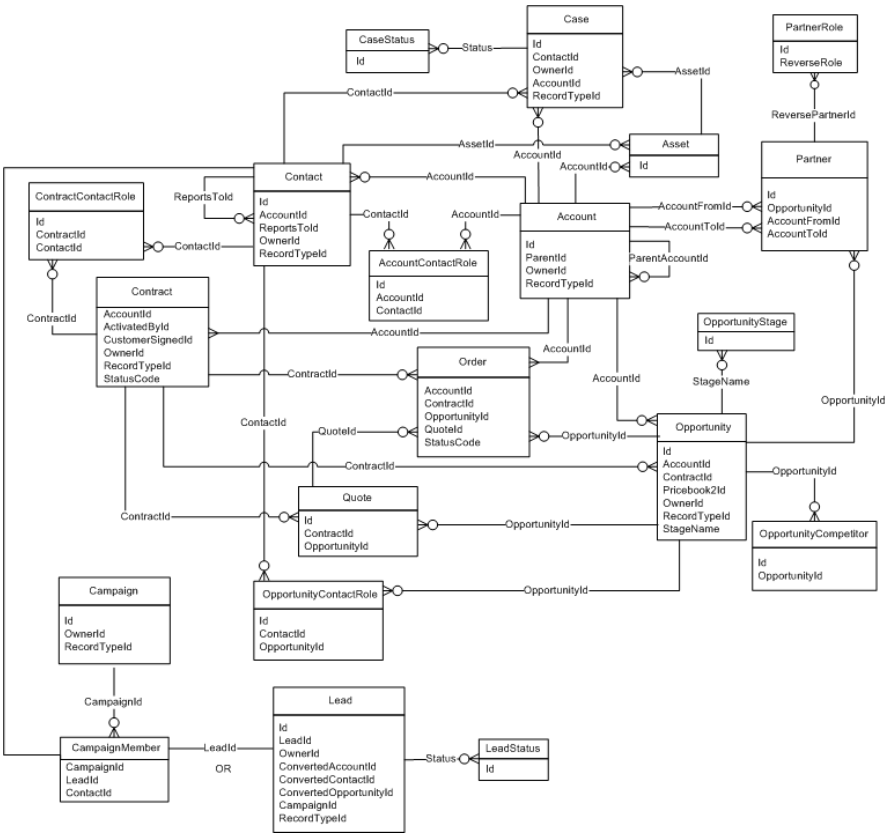
Hier ein Beispiel für ein DML Statement, welches eine E-Mail in das Salesforce System loggt:

```
 Task storeEmail = new Task(Description = 'From: ' + UserInfo.getUserEmail() +
'\nTo: ' + receiverEmail + '\n' + mailTextWithoutSignature,

                            Priority = 'Normal',

                            Status = 'Completed',

                            Subject = subjectLine,

                            IsReminderSet = false,

                            WhatId = getAccountID(),

                            TaskSubtype = 'Email',

                            ActivityDate = System.today()+0);

insert storeEmail;
```

Es wird zunächst ein normales Objekt vom Typ Task angelegt und dieses erst im nächsten Schritt mit insert geschrieben und in einen persistenten Zustand überführt.

4. Datenmodell

Im Vergleich zu anderen Umgebungen nimmt das Model in Salesforce sicherlich einen größeren Stellenwert ein. Die Abbildung [20] zeigt nur einen recht kleinen Ausschnitt der Standard-Objekte in Salesforce.



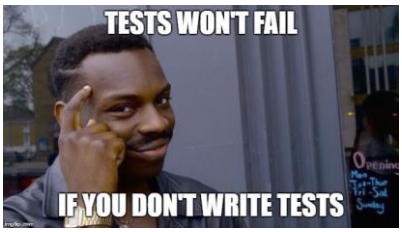
Das vorgebenene Modell wird in fast allen Fällen durch eigene Objekte erweitert werden. Es lohnt sich hier am Anfang etwas gründlicher zu planen – spätere Änderungen nach Initial Datenload sind schwierig und anfällig für Inkosistenzen.

Schwächen im Datenmodell müssen in der Praxis oft später in der Logik ausgeglichen werden. Typische Probleme sind das nicht einheitliche Ablegen von Informationen. Im eingangs erwähnten Beispiel des Versicherungsunternehmens könnte es z.B. ein Objekt Versicherungstyp__c und ein Objekt Versicherungsvertrag__c geben und die angebotenen Versicherungen verteilen sich auf diese beiden Objekte. Die Logik muss dann stets beide Objekte abfragen.

Ein weiteres häufiges Problem sind Relationen, die in beide Wege gerichtet sein können oder auch mal in beide Richtungen bestehen. Hier erhöht sich schnell der Aufwand, wenn neue Anforderungen dann nicht mehr mit den „Bordmitteln“ von Salesforce umgesetzt werden können und aufwändig ein Workarround entwickelt werden muss.

5. Unit Tests

Eine Besonderheit bei der Entwicklung auf der Force.com Plattform ist, dass eine Anwendung mindestens 75% Line-Coverage erreichen muss, um auf die Produktivumgebung übertragen werden zu können. Falls die Coverage darunterliegt oder ein Test fehlschlägt, scheitert der Deploy. [22]



Keine Option für Force.com [21]

Die Tests sind ähnlich aufgebaut wie bei Java/jUnit:

```
@isTest
public class rejectComponentTest {
    public static testMethod void addAccount() {
        PageReference pageRef = Page.anyComponent;
        Test.setCurrentPage(pageRef);
        anyController myController = new anyController();
        Account myAccount = new Account();
        myAccount.Name = 'anyTestAccountName';
        myAccount.State__c = 'Closed';
        insert myAccount;
        System.assertEquals(false, myController.getShowForm());
    }
}
```

Eine Testklasse wird mit @isTest eingeleitet und besteht sonst aus Methoden, die nicht mehr separat gekennzeichnet werden. Der Testaufbau ist analog zu Java, die Überprüfung der Testbedingung erfolgt mit `System.assertEquals(expected, actual).` Es kann auch

`System.assertNotEquals(!expected, actual)` oder `System.assert(condition)` verwendet werden. [23]

6. Deployment

Das Deployment von der Entwicklungsumgebung auf Produktiv wird durch ein Changeset vorgenommen. Dadurch werden die veränderten Dateien der Codebase sowie Änderungen in der Konfiguration (wie z.B. ein neues Custom-Field) über die Weboberfläche einem „Deploy Package“ hinzugefügt. Dieses wird anschließend an die Produktions-Org geschickt. Dort muss es entgegengenommen werden und alle Tests der Produktivumgebung erfolgreich durchlaufen. Anschließend sind die Änderungen live. Der gesamte Deploy Prozess wird über die Weboberfläche gesteuert. [25]

Der normale Entwickler hat oft nur Zugriff auf eine Sandbox mit ca. 10% der gesamten Daten. [25] Salesforce gelingt es aber ganz gut, hier aber keine „halben“ Datenpunkte zu erzeugen, wie z.B. fehlende Relationen, so dass der Wechsel zur Produktumgebung eigentlich keine Probleme bereiten kann. In den allermeisten Fällen funktioniert das auch gut. Für Produktiv freigeschaltet wird eine Anwendung meist nicht von der Person die sie entwickelt hat.

Aufgrund der weitreichenden Berechtigungen und der Exportmöglichkeit für alle Kundendaten gibt es in der Praxis oft nur zwei Personen in einem Unternehmen mit vollständigem Administratorzugriff auf das CRM.

Sehr häufig werden die Änderungen in der Produktumgebung für die ersten Tage nur für die Administratoren freigeschaltet um live noch etwas zu „testen“ bevor eine neue Funktion für alle Benutzer freigeschaltet wird.

IV. Persönliches Fazit

Ein Fazit zu Apex zu ziehen ist nicht ganz leicht, da die Sprache für ihren Anwendungsfall alternativlos ist. Es ist ja nicht so, dass man seine Salesforce-Komponenten in Python entwickeln und auf Apex einfach verzichten könnte. Mitnichten - wenn Salesforce erst einmal eingeführt ist, ist es zu spät und man wird sich dann auch mit Apex beschäftigen (müssen).

Daher sollte die Frage aus Sicht des Unternehmens lauten: brauchen wir Salesforce? Oder wäre PipeDrive, Bitrix24, vTiger oder close.io für unsere Zwecke besser geeignet?

Wenn es darum geht, komplexe Geschäftsprozesse im CRM abzubilden und das CRM sehr stark auf den spezifischen Unternehmenszweck auszurichten, ist Salesforce unschlagbar. Oft unterschätzen Unternehmen jedoch den Aufwand für die Anpassung. Apex löst dieses Problem nicht. Zudem gefällt mir die Implementierung von SOQL nicht, die viele der SQL-Funktionen vermissen lässt.

Wer sich jedoch gegen Apex entscheidet, wird erst eine Schnittstelle vom CRM zu seiner Anwendung entwickeln müssen. Anfragen über so entwickelte Anwendungen werden immer zusätzlich Latency für das Abrufen der Daten aus dem CRM haben. Auch bei Verarbeitung der Daten in einer externen Applikation wird man auf das spezifische Datenmodell des CRMs Rücksicht nehmen müssen.

Apex stellt daher meiner Meinung nach eine mächtige Option dar, sich komplett eigene Anwendungen auf dem bestehenden Datenmodell zu bauen. Seine Stärken spielt Apex besonders dann aus, wenn der Agent im CRM mit der Komponente auch in Echtzeit interagieren soll.

Insgesamt sehe ich Force.com und Apex vorsichtig positiv – wer etwas Aufwand nicht scheut, bekommt hier sichere und stabile Lösungen, die das eigene Unternehmen voranbringen.

Quellenverzeichnis

[1] https://appexchange.salesforce.com abgerufen 4.5.2018

[2] https://trailhead.salesforce.com/en/modules/business_process_automation/units/process_builder abgerufen 4.5.2018

[3] https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm abgerufen 4.5.2018

[4] https://www.revolutiongroup.com/blog/what-does-a-salesforce-developer-do/ abgerufen 4.5.2018

[5] https://www.cio.com/article/3168901/it-skills-training/salesforce-skills-in-high-demand-in-2017.html abgerufen 4.5.2018

[6] https://neuvoo.de/gehalt/Salesforce-Developer-Gehalt abgerufen 4.5.2018

[7] https://www.indeed.com/salaries/Salesforce-Developer-Salaries,-Boston-MA abgerufen 4.5.2018

[8] https://medium.com/understanding-as-a-service-uaas/demystifying-the-numbers-behind-salesforce-com-s-appexchange-60b3cedbc01f abgerufen 4.5.2018

[9] https://www.lifehacker.com.au/2016/03/how-much-do-mobile-developers-make-per-app/ abgerufen 4.5.2018

[10] https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm abgerufen 4.5.2018

[11] https://developer.salesforce.com/platform/force.com abgerufen 4.5.2018

[12] https://developer.salesforce.com/docs/atlas.en-us.fundamentals.meta/fundamentals/adg_intro_benefits.htm abgerufen 4.5.2018

[13] https://force201.wordpress.com/from-java-to-apex/ abgerufen 4.5.2018

[14] http://www.salesforcegeneral.com/salesforce-mvc-explained/ abgerufen 4.5.2018

[15] https://developer.salesforce.com/docs/atlas.en-us.212.0.api.meta/api/sforce_api_objects_custom_objects.htm abgerufen 4.5.2018

[16] https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_open_source.htm abgerufen 4.5.2018

[17] https://developer.salesforce.com/page/An_Introduction_to_Apex abgerufen 4.5.2018

[18] https://developer.salesforce.com/page/From_SQL_to_SOQL abgerufen 4.5.2018

[19] https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_dml.htm abgerufen 4.5.2018

[20] https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_erd_majors.htm abgerufen 4.5.2018

[21] http://devhumor.com/media/tests-won-t-fail-if-you-don-t-write-tests abgerufen 4.5.2018

[22] https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_code_coverage_intro.htm abgerufen 4.5.2018

[23] https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_qs_test.htm abgerufen 4.5.2018

[24] https://help.salesforce.com/articleView?id=changesets_inbound_deploy.htm&type=5 abgerufen 4.5.2018

[25] https://help.salesforce.com/articleView?id=create_test_instance.htm&type=5 abgerufen 4.5.2018