SPA-Anwendungen mit Angular, JSON und SAS

Carsten Zaddach
BDE Business Datawarehouse
Engineering GmbH
Landsberger Str. 218
12623 Berlin
cz@bde-gmbh.de

Zusammenfassung

Die ansprechende Darstellung von Informationen in Form von Grafiken oder Tabellen im Internet bzw. Intranet ist heutzutage eine wichtige Forderung in Projekten. Dabei muss die Generierung der HTML-Seiten nicht mehr in SAS erfolgen, was eine saubere Trennung zwischen Daten und deren Darstellung ermöglicht.
Mittels des Dreiergespanns Angular4, JSON und SAS sind mittlerweile Webanwendungen möglich, die sowohl optisch sehr ansprechend sind als auch von der Performance keine Wünsche übrig lassen. Das SAS mit der Version 9.4 M4 über eine eingebaute JSON-Libname Engine verfügt, kann die Kommunikation zwischen der Applikation und SAS im Backend bidirektional mit dem Quasistandard JSON erfolgen.

Schlüsselwörter: JSON, Webservice, Ajax, HTML

1 Einführung

Die Verbreitung von Informationen über HTML-basierte Berichte hat sich mittlerweile zum Quasistandard entwickelt. Dabei verschwinden die Grenzen zwischen Desktop- und Webapplikationen immer mehr. Zunehmend erwarten die Anwender auch von solchen Berichten, dass sie sich wie eine „normale“ Anwendung verhalten und nicht bei jeder Aktualisierung der Daten die Seite neu aufgebaut wird. Eine serverseitige Generierung der HTML-Seiten ist somit nicht mehr nötig bzw. auch nicht gewünscht. Es erfolgt somit eine klare Trennung zwischen der Generierung bzw. Aufbereitung der Daten auf der Serverseite und deren Darstellung auf der Clientseite. Für den Datenaustausch stehen theoretisch eine Vielzahl von möglichen Formaten zur Verfügung, jedoch hat sich die Verwendung von JSON (JavaScript Object Notation) als Standard in diesem Zusammenhang etabliert.

2 Was steckt hinter SPA, Angular, Ajax und JSON?

Da im Folgenden einige Begriffe verwendet werden, deren Bedeutung mitunter nicht jedem bekannt ist, werden diese Begriffe in diesem Abschnitte näher erklärt.

2.1 SPA

Der Begriff SPA hat in diesem Zusammenhang nichts mit der bekannten Wellnesseinrichtung zu tun, sondern ist die Abkürzung für **S**ingle-**P**age-**A**pplication. Die Idee dahinter ist, dass der Benutzer lediglich eine Seite lädt, was natürlich ziemlich schnell passiert, und sämtlicher anderer Inhalt dynamisch durch Ajax hinzugeladen wird. Dies hat nicht nur den Vorteil, dass das wiederholte Generieren und Laden entfällt, sondern auch, dass sämtliche in der Seite enthaltenen Elemente wie Bilder und Javascript-Code nur einmal geladen werden, was die gefühlte Geschwindigkeit erhöht.
Durch die clientseitige Ausführung der Anwendung wird somit die Serverlast erheblich reduziert, was sich wiederum in einer schnelleren Abarbeitung der (Daten-)Anfragen bemerkbar macht. Auch wird die Menge der zu übertragenen Daten erheblich reduziert, da das Gerüst der Seite bereits vorhanden ist. Klassische bekannte Vertreter dieser Art von Anwendung sind z.B. Facebook, Twitter sowie die meisten Webmail-Seiten.
Für die Erstellung von Single-Page-Applikationen steht eine Reihe von Open-Source-Frameworks zur Verfügung. Zu nennen sind hier die wichtigsten Vertreter wie Angular (Google), React (Facebook) und Knockout.js (Community).

2.2 Angular

Das Framework Angular wurde ursprünglich von Google entwickelt und wird auch weiter von Google aktiv betreut. Mit dem Sprung auf die Versionsnummer 2 wurde das Konzept des Frameworks im Vergleich zur Version 1 komplett geändert. Im Rahmen dieses Beitrages wird auf die neue Architektur des Frameworks Bezug genommen.
Da in diesem Zusammenhang nicht das komplette Framework erklärt werden kann, da dies den Umfang sprengen würde, wird nur auf die für diesen Beitrag wichtigsten Elemente eingegangen. Bei weiterführendem Bedarf sei auf die offizielle Webseite https://angular.io/ verwiesen.
Die wichtigsten Elemente sind Module, Komponenten, Templates, Metadaten, Datenbindung, Services und Direktiven.

2.2.1 Module

Eine Angular-Anwendung besteht aus n Modulen, wobei das Root-Modul immer vorhanden sein muss. Die folgende Abbildung stellt den klassischen Aufbau einer solchen Anwendung dar. Ein Modul bildet in der Regel eine separate Funktionalität ab z.B. Anmeldemodul oder Dashboardmodul. Dies hat den Vorteil, dass das gleiche Modul in unterschiedlichen Angular-Anwendungen wiederverwendet werden kann. Wie in Abbildung 1 zu sehen ist, besteht ein Modul wiederum aus n Komponenten.

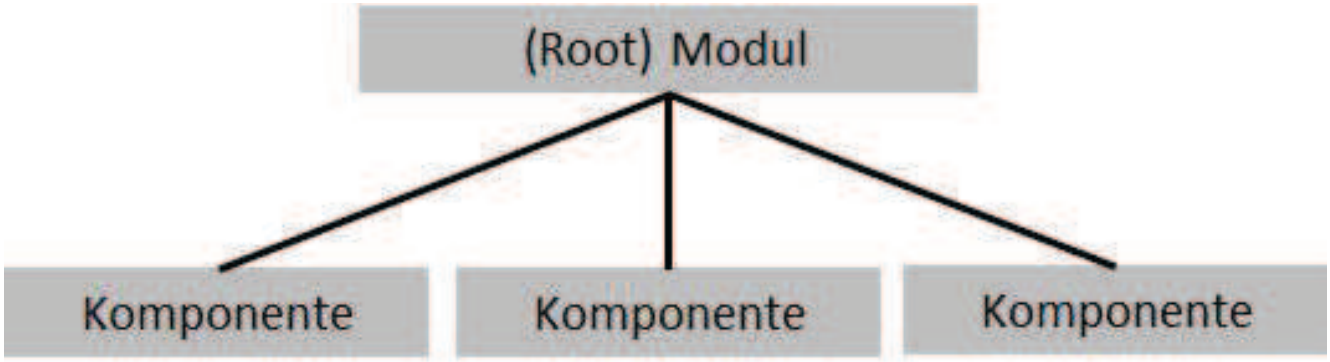

Abbildung 1: Schematischer Aufbau einer Angular-Anwendung

2.2.2 Komponenten

Die Komponente ist der darstellende Teil der Anwendung und kontrolliert einen Teil des Bildschirms, den View. Eine Komponente kann z.B. ein Anmeldedialog sein, eine Tabelle, eine Grafik oder auch eine eMail-Liste. Jede Komponente beinhaltet wiederum drei „Teile“: das Template, die Klasse und Metadaten.
Das **Template** definiert das Aussehen einer Komponente, denn es beinhaltet den darzustellenden HTML-Code mit ggf. enthaltenen Direktiven.
Die zur **Komponente** zugehörige Klasse enthält die deren bereitgestellten Funktionen. So kann in der Klasse z.B. definiert sein was passieren soll, wenn der Button „Neu“ gedrückt wird oder die Auswahl einer Dropdown-Box geändert wird.
Die **Metadaten** geben Angular zusätzliche Informationen darüber, wie Klasse abgearbeitet und interpretiert werden soll. Das Template ist beispielsweise ein Teil der Metadaten einer Komponente.

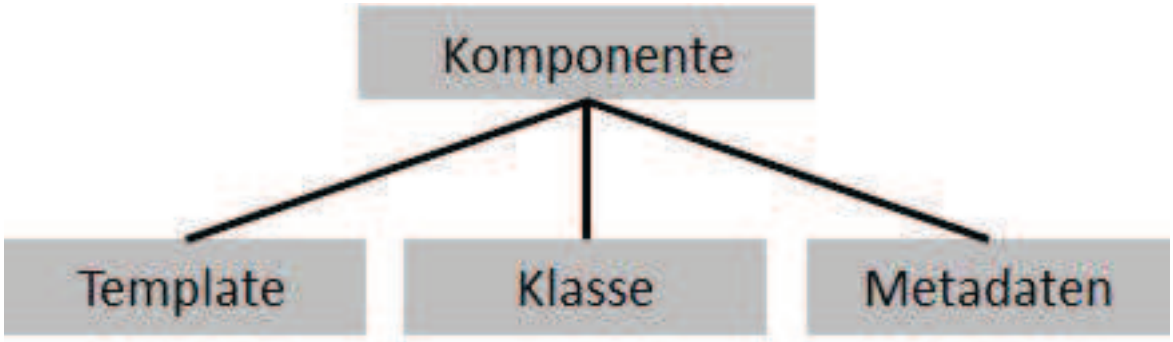

Abbildung 2: Schematischer Aufbau einer Angular-Komponente

Die Definition einer Komponente sieht beispielsweise folgendermaßen aus:

```
@Component({
  selector:    'app-list',
  templateUrl: './list.component.html',
  providers:  [ DataService ]
})
export class ListComponent implements OnInit {
}
```

2.2.3 Datenbindung

In „normalen“ HTML-Seiten ist der Entwickler dafür verantwortlich, neue Daten an die entsprechenden HTML-Objekte zu senden bzw. diese zu aktualisieren, Benutzereingaben in entsprechende Aktionen umzuwandeln und eingegebene Daten entsprechend zu behandeln. Dies ist natürlich nicht nur aufwendig, sondern auch sehr fehleranfällig.
Durch die Datenbindung im Angular-Framework wird dem Entwickler diese Arbeit abgenommen. Wie in der unteren Abbildung dargestellt, unterstützt das Framework vier mögliche Varianten der Datenbindung. Neben dem „einfachen“ Weg wird auch eine bidirektionale Datenbindung unterstützt. Eine bidirektionale Datenbindung kommt meist bei Eingabefeldern, Auswahlboxen oder ähnlichen HTML-Objekten zum Einsatz.

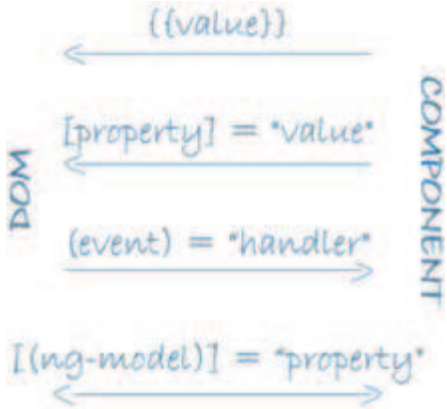

Abbildung 3: Varianten des Databinding
(Quelle:https://angular.io/generated/images/guide/architecture/databinding.png)

2.2.4 Direktiven

Wie bereits gesehen, definieren die Templates das Erscheinungsbild einer Komponente, wobei Angular das Template analysiert und entsprechend den darin enthaltenen Anweisungen (Direktiven) rendert. Diese Direktiven können eine Vielzahl von möglichen Aufgaben abbilden. So können über Direktiven Schleifen abgebildet werden, oder HTML-Elemente können ein-/oder ausgeblendet werden, abhängig vom Wert einer Variablen etc. Auch können über Direktiven die CSS-Klassen von HTML-Elementen automatisch geändert werden, wenn sie der Wert einer Variablen ändern.

2.2.5 Services

Services stellen, wie der Name schon sagt, den Komponenten einen bestimmten Service in Form von Funktionen zur Verfügung. So kann es z.B. einen AnmeldeService mit den Funktionen logon und logoff geben. Oder einen generellen LoggingService, der von allen Komponenten benutzt werden kann. Services haben den Vorteil das sie kapseln, wie eine Funktion ihre Arbeit macht. So kann ein DummyDatenService die Daten während der Entwicklung von der Platte lesen, während der „echte“ DatenService diese über das Netz von einem Server empfängt. Die aufrufende Komponente muss sich um das „wie“ nicht kümmern, sondern erhält nur die Daten.

Im Rahmen dieses Beitrages werden wir einen Daten Service erstellen, der die Kommunikation mit dem SAS-Server übernimmt.

2.3 Ajax

Damit das Konzept der Single-Page-Applikation überhaupt funktioniert, muss jede Form von Kommunikation im Hintergrund und asynchron erfolgen. Die Asynchronität ist wichtig, damit die Anwendung auch beim Laden von Daten auf Anwendereingaben reagieren kann. Dieses Konzept ist unter dem Begriff Ajax (Asynchronous JavaScript and XML) zusammengefasst, wobei es natürlich nicht nur auf XML beschränkt ist.
In Angular ist das Konzept von Ajax bereits implementiert und kann somit für den entsprechenden Service verwendet werden.

2.4 JSON

JSON, die Kurzform für JavaScript Object Notation, ist ein plattformunabhängiges Format, mit dem Daten in kompakter Art und Weise zwischen Anwendungen ausgetauscht werden kann. Im Gegensatz zu XML ist JSON für den Menschen gut lesbar und leichter schreibbar.
Das Ziel von JSON ist es, JavaScript Objekte sehr schnell und einfach in eine Textform zu bringen und aus diesem Text wieder ein JavaScript-Objekt zu erstellen.
JSON unterstützt fasst alle JavaScript Datentypen. Dies sind im Einzelnen:

- true und false als boolsche Datentypen
- Zahl
- Zeichenkette, wobei diese mit einem doppelten Anführungszeichen beginnen und enden muss (keine einfachen Anführungszeichen!!)
- Arrays, die mit [beginnen und mit] enden
- Objekte, die durch { eingeleitet werden, und mit } abgeschlossen werden

Eine Kombination der einzelnen Datentypen ist möglich z.B. Array mit Objekten, oder ein Array als Feld eines Objektes.
Bei der Verwendung der Ajax-Funktionalität des Angular-Frameworks, wird eine empfangene Zeichenkette im JSON Format automatisch in ein JavaScript-Objekt konvertiert und ein JavaScript-Objekt beim Senden in das JSON-Format geschrieben.

2.5 Webservice

Ein Webservice ist ein zur Verfügung gestellter, anwendungsunabhängiger Dienst. Dieser Dienst kann entweder über das Internet bereitgestellt werden, oder intern im Intranet. Ein Webservice bildet nur eine einzige Funktion ab, z.B. Benutzerauthentifizierung oder Abfrage von Wetterdaten und wird durch eine eindeutige Adresse spezifiziert z.B. http://server/benutzer/logon für das Anmelden eines Benutzers.
Über eine Stored-Process und das dahinterliegende SAS-Programm, kann auch ein Webservice im SAS-Server definiert werden.

3 Trennung von Daten und deren Darstellung

Die Trennung von Daten und deren Darstellung hat mehrere Vorteile. So können Änderungen und Anpassung an beiden Teilen vorgenommen werden, ohne die andere Seite zu beeinträchtigen. Auch ist der Austausch der darstellenden Komponente ohne Anpassung an der Datenkomponente möglich. Außerdem wird der Programmcode nicht mit „zweckfremden“ Codeteilen belastet, die eine Wartung des eigentlichen Programmcodes erschweren.
Des Weiteren ist es so möglich, die beiden Teile getrennt voneinander zu entwickeln, denn es ist lediglich eine Verständigung auf das Austauschformat nötig. So kann das Know-How entsprechend optimal eingesetzt werden.
Updates in der Datenschicht können durchgeführt werden, ohne das die Darstellung beeinträchtigt wird oder angepasst werden muss. Ebenso kann der Datenservice für andere zur Nutzung zur Verfügung gestellt werden.

4 JSON in SAS

4.1 Lesen von JSON

Mit der Einführung von SAS9.4M4 verfügt SAS jetzt über eine Libname-Engine, mit der aus JSON-Dateien wieder SAS-Tabellen erstellt werden können.
Die Verwendung der Engine entspricht der gängigen Syntax für die Libname-Anweisung:

```
LIBNAME libref JSON < 'Pfad zu Datei.json' > <options>;

z.B.

LIBNAME libref JSON 'c:\temp\test.json';
```

Eine Ausgabe von SAS-Tabellen in das JSON-Format ist über die Engine leider nicht möglich. Sollte die angegebene Datei nicht existieren, so wird die Anweisung mit einem Fehler beendet.

4.2 Schreiben von JSON

Für die Ausgabe von JSON stellt SAS die gleichnamige Prozedur JSON zur Verfügung. Mittels dieser Prozedur können selbst komplexe JSON-Strukturen erstellt werden, da die Prozedur über Ausgabebefehle verfügt, mit der die beschriebenen JSON-Datentypen ausgegeben werden können.
Das folgende Statement schreibt beispielsweise alle Männer aus der SASHELP.CLASS in ein Array mit dem Namen „male“ und alle Frauen in ein Array mit dem Namen „female“.

```
proc json out="c:\temp\test.json" nosastags pretty;
   write open object;
   write values "male";
   write open array;
   export sashelp.class(where=(sex="M"));
   write close;
   write values "female";
   write open array;
   export sashelp.class(where=(sex="F"));
   write close;
   write close;
run;
```

Wenn man nun eine Libname mit der JSON-Engine auf die generierte JSON-Datei legt, erhält man folgendes Bild im SAS-Explorer.

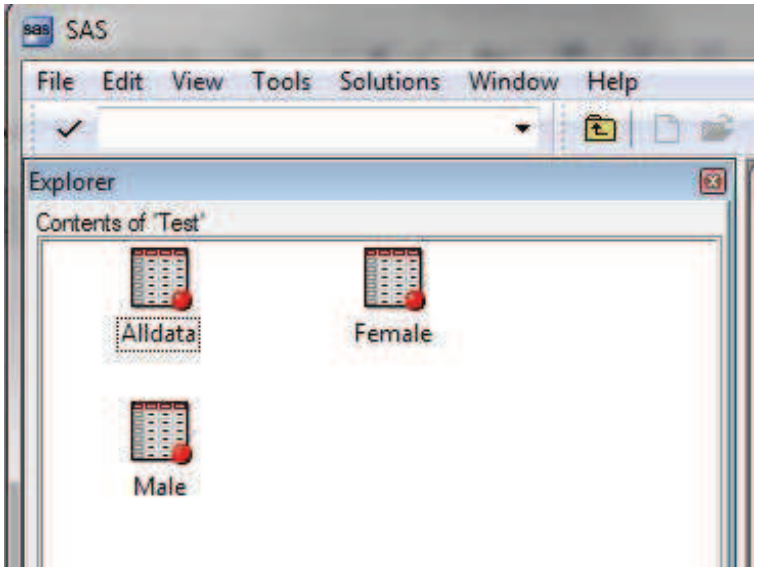

Abbildung 4: Darstellung der JSON-Daten im SAS-Explorer

5 Erstellung eines Webservices in SAS

Die Kommunikation zwischen der Webapplikation und dem SAS-Server erfolgt über einen einfachen Webservice. Dieser soll lediglich den Inhalt einer SAS-Tabelle also JSON-Format ausgeben. Der Webservices besteht lediglich aus dem kleinen folgenden Programm.

```
Proc json out=_webout nosastags pretty;
   write open object;
   write open array;
   export ksfe2018.testdaten
   write close;
   write close;
run;
```

Die Erstellung des Webservices erfolgt über das Kontext-Menü im Enterprise Guide.

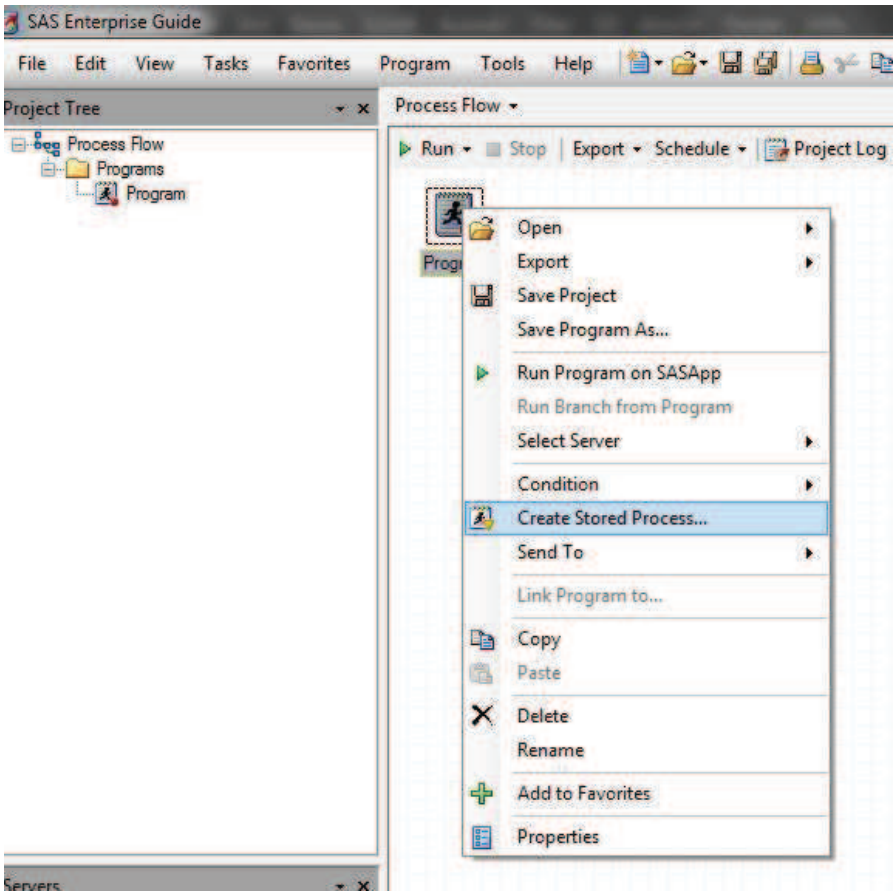

Abbildung 5: Erstellung eines Webservices über das Context-Menü des EG

6 Der Angular-DatenService

Wie bereits oben besprochen, erfolgt die Kommunikation zwischen den darstellenden Komponenten und dem SAS-Server mittels eines Services. Dieser unterscheidet sich nicht groß von normalem Code.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';
import { of } from 'rxjs/observable/of';
import { catchError, map, tap } from 'rxjs/operators';

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};

@Injectable()
export class DatenService {
```

```
  private webserviceUrl = '
/SASBIWS/rest/storedProcesses/Beispiel/ksfe2018/dataTargets/_WEBOUT'
;

  constructor(private http: HttpClient) { }

  ladeDaten (): Observable<TableData[]> {
    return this.http.get<TableData[]>(this.webserviceUrl);
  }
}
```

In diesem Beispiel wurde der Stored Process unter dem Ordner Beispiel und mit dem Namen ksfe2018 abgelegt. Es wird vorausgesetzt, dass die Webapplikation sich auf dem gleichen Rechner befindet wie der Webservice. Andernfalls muss die Variable webserviceUrl entsprechend angepasst werden.

7 Das Resultat

Nachdem alle Eintragungen vorgenommen wurden, kann die Angular-Applikation auf die Daten von SAS-Server ohne Probleme zugreifen und in der Tabelle darstellen.

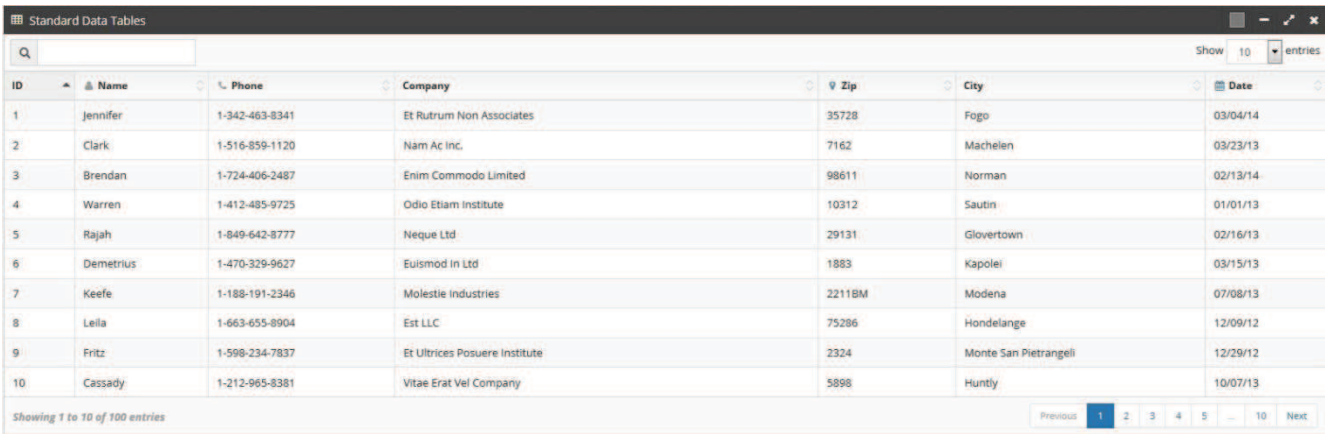

Abbildung 6: Tabelle im Frontend mit Daten aus dem SAS-Backend