Das World Wide Web wächst – und wir wachsen mit

Fachartikel über die REST API und ihre Implementierung, Autor: Nicolas Schwarz

"Wie entsteht innovatives Denken? Es ist eine Geisteshaltung, für die man sich entscheiden muss."

Dieses Zitat stammt von Tesla-Chef und Space X-Gründer Elon Musk, welcher mit seinen Firmen im Bereich Technologie fortlaufend neue Wege öffnet und Erfolge feiert. Doch auch wenn sein Name momentan in aller Munde ist, bedeutet das nicht, dass Musk mit seinen Leistungen alleine dasteht. Man muss nur zurück auf die Menschheitsgeschichte schauen, beispielsweise auf die Entwicklung des Internets und auf die Leistungen von Roy Fielding, dem Entwickler des REST-API.

In diesem Artikel soll dem Leser anschaulich vermittelt werden, was ein REST-API ist, woher es kommt und wie man es gebraucht. Hierbei werden anschauliche Beispiele verwendet, damit der Leser ein Verständnis für das Thema entwickelt. Es werden die Programmiersprachen HTML, JavaScript und Java verwendet.

Woher stammt der Name?

Die Abkürzung REST steht für „Representational State Transfer", API für „Application Programming Interface". Diesen Namen verdankt die Architektur ihrer Funktion als Kommunikationsschnittstelle.

Wer hat es entwickelt und wo wird es gebraucht?

Das REST-API wurde von **Roy Fielding**[1] entwickelt und erstmals im Jahre 2000 vorgestellt. Es entstand durch das **„http Object Model"**[2], welches ebenfalls von Roy Fielding entworfen wurde. REST wird oft nicht direkt eingesetzt, sondern in abgewandelter Form, das sind sogenannte RESTful-Webservices. REST wird/wurde überall da gebraucht/genutzt, wo eine Kommunikationsschnittstelle zwischen Client und Server aufgebaut wird, beispielsweise Online-Dienste mit Single-Page-Anwendungen.

Das sind Webseiten, die beim Aktivieren von Funktionen nicht neu laden, sondern sich sozusagen anpassen. In diesem Artikel geht es um eine RESTful-Website zum Bewerten von Restaurants. Um diesen Server so portabel wie möglich zu halten, werden MicroProfiles eingesetzt.

Ein MicroProfile definiert eine Sammlung kleiner APIs, die zusammen einen zentralen Microservice ergeben, mit dem Ziel, über mehrere Laufzeiten hinweg, eine angenehme Portabilität zu erschaffen.

Aufbau der Architektur

Es wird bei REST kein fester Aufbau vorgeschrieben, aber es werden sechs Prinzipien definiert, die als Voraussetzung dienen.

Das Prinzip des **„Client-Server-Modells"** beschreibt eine Trennung von Nutzeroberfläche und Datenhaltung, sprich Datenbanken und Server laufen getrennt von dem Nutzerinterface.

Außerdem wird die **„Zustandslosigkeit"** gefordert, dies bedeutet, dass Client und Server zustandslos miteinander kommunizieren. Jede Anfrage beinhaltet alle Informationen die der Server braucht, der Server selbst greift auf keine gespeicherten Zusammenhänge zurück.

Das Prinzip des **„Caching"** ist allgemein bekannt und wird hier ebenfalls gefordert. Vom Server gesendete Informationen können vom Client gespeichert und wiederverwendet werden, so wird nicht unnötigerweise eine erneute Anfrage an den Server geschickt und die Netzwerkauslastung verringert.

REST-Dienste nutzen **„einheitliche Schnittstellen",** welche vom Datendienst entkoppelt sind. Hiermit erreicht man eine vereinfachte Architektur, auch wenn dies die Effizienz beeinträchtigen kann, da Daten, manchmal hingegen ihres Ursprungsformates, in das einheitliche Format gebracht werden müssen, bevor diese über die Schnittstellen weitergegeben werden können.

Die ganze Architektur basiert auf einem hierarchischen System, ein sogenanntes **„Layered System"**. Eine Schicht kann nur mit der darüber oder darunterliegenden direkt kommunizieren, so müssen Informationen zwar durchgereicht werden, erhält aber einen einfacheren Aufbau.

Zuletzt gibt es hier noch die Anforderung, dass die Clientfunktionen mit Hilfe von beispielsweise Skripten einfach erweitert werden können, dies wird **„Code-On-Demand"** genannt.

[1] US-amerikanischer Informatiker
[2] Spezifikation einer Programmierschnittstelle

Umsetzung einer Webanwendung anhand eines Beispiels

In diesem Abschnitt wird an Hand von Beispielen gezeigt, wie die Kommunikation zwischen einem getrennten Client und Server funktioniert und wie diese mit Hilfe von JavaScript und Java umgesetzt werden kann.

Die Post-Methode

Mit Hilfe von „Post" können Daten vom Client an den Server übermittelt werden. Ein Webentwickler hat beispielsweise ein HTML-Formular zum Registrieren entworfen, welches mit Hilfe von „onklick=" eine JavaScript-Methode aufruft.

```
<form id="anmeldeform" method="post">
    <table>
        <tr>
            <td>Vorname</td>
            <td><input id="Vorname" type="text" name="firstname" size="40"
            required /></td>
        ...
            <button id="Registrieren" type="button" style="width: 100px;
            height: 30px;" onclick="Register()">Registrieren</button>
            </td>
        </tr>
    </table>
    </form>
```

Abbildung 1 HTML-Formular

In diesem Fall wird die JavaScript-Methode „Register()" aufgerufen, die im Anschluss die Daten für eine Schnittstelle vorbereitet.

```
function Register(){
    let data = {
            username :document.querySelector("#Benutzername").value,
            ...
            password : document.querySelector("#Password").value
        };
fetch('rateme/register',   {
   method: 'post',
   headers: {
    'Content-type': 'application/json'
   },
   body: JSON.stringify(data)
 })
 .then( response => {
        if( !response.ok )
        {
            ...
        }
        else
        {
            ...
        }
    } )
 .catch( error => console.error('Error:', error));
}
```

Abbildung 2 JavaScript-Methode

In der Methode „Register()" werden die Daten aus dem Formular abgerufen und als Variablen gespeichert. Danach werden mit Hilfe eines „Ajax-Requests" die Daten an einen serverseitigen Controller geschickt. Hierbei wird auch festgelegt, welcher Controller und welche Methode adressiert werden soll, in unserem Fall die Post-Methode. Je nachdem welche „Response", also Antwort, der Server dann zurückgibt, werden weitere Befehle ausgeführt. Der Controller wird hierbei mit Hilfe der „fetch-Zeile" adressiert, genauer noch die, mit „post" gekennzeichnete, Methode.

Die Get-Methode

Mit Hilfe einer Get-Methode können Daten vom Server empfangen werden, hier ist der Aufbau ähnlich. Man schickt einen Ajax-Request an den Server, dieser liest Daten beispielsweise aus einer Datenbank aus oder führt Berechnungen durch. Nach der Datenverarbeitung schickt er diese dann zurück und die JavaScript-Methode handelt dann je nach Resultat.

Die Delete-Methode

Eine andere wichtige Funktion ist die Delete-Methode. Sie dient beispielsweise dazu, einen Nutzer aus einer Datenbank wieder zu löschen oder ein Cookie[3] zu entfernen. Hierbei werden auch Daten von einer JavaScript-Methode an den Server übergeben und mit Hilfe dieser Daten arbeitet der Controller dann weiter.

[3] Datei zum Informationsspeicher

Die Put-Methode

Im Prinzip ist die Put-Methode nichts anderes als die Post-Methode, allerdings mit dem Unterschied, dass bei der Put-Methode der Client auswählt was adressiert wird, beispielsweise zum Updaten eines schon vorhandenen Eintrages. Bei der Post-Methode wird ein neues Element erstellt und der Server erledigt die Adressierung des Elements.

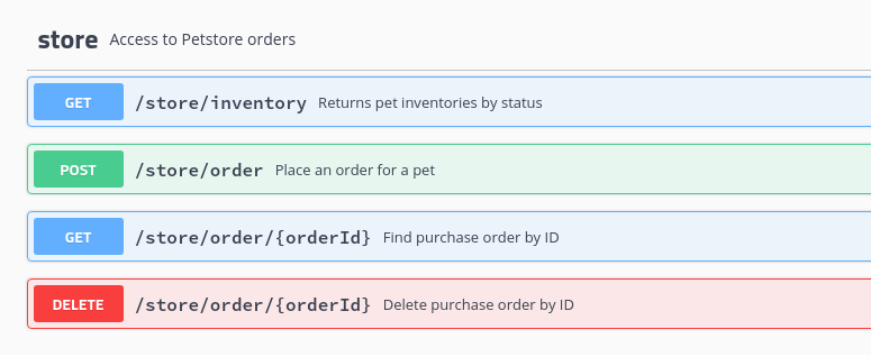

Abbildung 3 Beispiele der verschiedenen Anwendungen, Quelle: assertible.com

Beispiel eines Controllers

Bisher ist in diesem Artikel das Wort Controller sehr oft gefallen, weshalb nun aufgezeigt wird wie ein solcher aussieht und aufgebaut wird.

```
import java.util.UUID;
...
import de.hskl.swtp.security.AccessManager;

@Path("/register")
@Singleton
public class RegisterController
{
    @Inject
    UserDB userDB;

   @Inject
   AccessManager accessController;

   @POST
   @Consumes(MediaType.APPLICATION_JSON)
   public Response register(User user)
   {
     boolean successful = userDB.existUser(user);
     if (!successful) {
         System.out.println("ERROR Username already in use!");
         return Response.status(404).build();
     }
     UUID uuid = this.accessController.login(user.getUsername());
     NewCookie loginCookie = new NewCookie("LoginID", uuid.toString());
     return Response.status(200).cookie(loginCookie).build();
   }
}
```

Abbildung 4 Beispiel eines Controllers

Zunächst werden alle notwendigen Imports durchgeführt. Die Klasse „RegisterController" wird mit dem Pfad gekennzeichnet, unter der man sie später finden kann. In der Klasse selbst müssen alle anderen Serverabhänigkeiten importiert werden, das passiert hier mit „@Inject...". Nun kommt die Methode „register(...)" zum Tragen. Diese wird oberhalb des Methodenkopfes mit „@POST" gekennzeichnet, damit später bekannt ist, welche Methode mit der vorher genannten Post-Methode aufgerufen wird. In der Methode selbst wird dann mit der Datenbank kommuniziert. Die Methode gibt eine „Response" zurück, auf der die JavaScript-Methode dann später aufbauen kann. In diesem Fall gibt das Programm, wenn die „existUser(user)-Methode" der Klasse „userDB" ein „false-Wert" zurückgibt, eine negative Statusmeldung weiter, falls der Nutzer nicht existiert, gibt die Methode einen „true-Wert" zurück, wird der User in die Datenbank aufgenommen und es wird direkt eine Cookie-Session erstellt. Mit Hilfe des „accessControllers" wird der Nutzer danach eingeloggt.

Fazit

Als abschließendes Statement kann festgehalten werden, dass die Idee von REST sehr einfach zu verstehen und zu implementieren ist. Im heutigen digitalen Zeitalter ist REST unverzichtbar und wird deshalb auch sehr häufig eingesetzt, zwar nicht immer eins zu eins wie es gedacht ist, aber zumindest darauf basierend.

Zum Autor

Nicolas Schwarz ist Studierender an der Hochschule Kaiserslautern im Fachbereich Informatik. Dieser Fachartikel entstand auf Grund einer Projektarbeit in der Veranstaltung „Softwaretechnik Praktikum" unter der Leitung von Prof. Dr. Jörg Hettel. Sie erreichen Ihn unter info@nicolas-schwarz.de.

Links & Literatur

[1] Representational State Transfer

https://de.wikipedia.org/wiki/Representational_State_Transfer

[2] Roy Fielding

https://de.wikipedia.org/wiki/Roy_Fielding

[3] Was ist eine REST API?

https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/

[4] Konzept, Aufbau und Funktionsweise von REST

https://www.dev-insider.de/konzept-aufbau-und-funktionsweise-von-rest-a-603152/

[5] 7 HTTP methods every web developer should know and how to test them

https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them

[6] Titelbild

NicoElNino/Shutterstock.com