**REST - Representional State Transfer**

REST Ist kurz gesagt ein Architektur-Stil und eine Abstraktion für verteilte Hypermedia-Systeme. Als solches ist REST nicht Service orientiert sondern Resourcen orientiert. REST ist hingegen keinen Technologie, kein Protokoll und kein zweites SOAP. Der Begriff REST trat erstmals im Jahre 2000 auf und wurde durch Roy Fielding in seiner [Dissertation](http://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm) eingeführt.

**5 Prinzipien**

**Ressourcen eindeutig identifizierbar.**

Eine Resource ist ein Objekt, welches ein Entwickler als "eindeutig identifizierbar" in einen Service implementieren will. Das heißt konkret, dass eine Resource sowohl ein statisches HTML Dokument, als auch ein dynamischer Webservice, ein Datensatz, eine Funktion/Methode oder gar ein ganzer Service sein kann. Eine Resource kann durch diverse Requests abgerufen, angelegt, verändert oder gelöscht werden. Die eindeutige Identifikation der Resource wird durch die [URI ( Uniform Resource Identifier )](http://www.ietf.org/rfc/rfc3305.txt) sichergestellt.

**Verwendung von Hypermedia**

Hypermedia bezeichnet die Verlinkung von Dokumenten untereinander. Im Sinne von REST würde man eher die Verlinkung von Resourcen verstehen. Eine Resource kann wiederrum Unterresourcen haben und verweist auf diese. Somit entsteht ein Resourcen-Baum durch den sich ein User oder auch ein Computer itterieren kann.

**Verwendung standardisierter Methoden**

Nach der Dissertation von Roy Fielding sind diese Methoden vom Protokoll HTTP abgeleitet / übernommen. Diese sind hauptsächlich GET, POST, PUT, DELETE und HEAD und werden von jedem Browser verstanden.

**Mehrere Repräsentationen**

Eine Repräsentation stellt das Datenformat einer Resource dar, welches vom Server im Response zurückgeliefert wird. Dabei kann eine einzelne Resource mehrere Repräsentationen haben. Zum Beispiel kann die Beschreibung eines Autos ( Marke = Tatra, Farbe = Grün, etc. ) sowohl als vollständiges HTML Dokument, als auch im XML, im JSON oder in sonstigen Formaten zurück gegeben werden. Um eine bestimmte Form der Repräsentation abzurufen wird entweder ein Accept: <Format> HTTP Header Field beim Request mit übertragen. Danach kann der Server die passende Representation zurückliefern. Eine weitere Möglichkeit besteht darin, die Representation als Dateiendung in den Request zu verpacken. Bsp: localhost/auto.html oder localhost/auto.xml.

**Statuslose Kommunikation**

Statuslose Kommunikation bedeutet, dass der Server keine Angaben über den Clientstatus speichert. Der Server "trennt" die Verbindung sofort nach erfolgreichem Abarbeiten eines Requests respektive nach erfolgtem senden des Response. Er kann den Client allerdings dennoch identifizieren. Z.B. durch Cookies. Diese Technik hat jedoch nichts mit dem Begriff statuslos zu tun. Statuslos ist zum Beispiel auch das HTTP Protokoll.

**Was ist eine Resource?**

Resourcen sind im weitesten Sinne Daten, Methoden und Dateien, die eine eindeutige Identität besitzen. Hierbei ist es unerheblich,  
ob es sich Serverseitig um eine statische Datei, ein Skript oder sonstige Techniken handelt.  
Wichtig ist nur, dass sich die URI nicht verändern sollte. Beim Aufruf des gleichen [URI](http://de.wikipedia.org/wiki/Uniform_Resource_Identifier) sollte stets das gleiche Ergebnis zurückgeliefert werden.

Beispiele für einfache Resourcen sind:

www.example.com/autos/marken/2010

www.example.com/foo/bar/

http://www.example.com/buecher

Bei diesem Beispiel ist sowohl das Protokoll als auch die Domäne, die Unterverzeichnisse, der Dateiname und die Dateiendung ausschlaggebend für unterschiedliche Resourcen. Zum Beispiel könnte das Protokoll HTTPS auf eine andere Resource zeigen als das Protokoll HTTP, obwohl der Rest identisch ist.

Beispiele für komplexere Resourcen sind:

www.example.com/foo/bar.php?filter=town&order=desc

www.example.com/foo/bar/key/value/key2/value2/

Bei diesen Beispielen ist die Resource auf den ersten Blick oftmals nicht zu erkennen.  
Man kann jedoch anmerken, dass die Parameter einer Query ?filter=town&order=desc normalerweise nicht als eigenständige Resource anzusehen sind, sondern diese nur Steuern oder Filtern sollten. Auch das zweite Beispiel ist ein wenig komplexer. In dieser Darstellung ist nicht zu erkennen, was genau die Resource darstellt. Es kann durchaus vorkommen, dass Teile des vermeintlichen Verzeichnisbaums in Wirklichkeit ein Teil der Query sind. Diese Technik des "Rewriting" wird z.B. von [Zend Framework](http://framework.zend.com/) verwendet. Hierbei werden alle Anfragen Serverseitig auf ein und dieselbe Datei gelenkt, welche die passende Resource zurück liefert. Siehe auch: [htaccess](http://de.selfhtml.org/servercgi/server/htaccess.htm). In diesem Beispiel könnte man sagen, dass /foo/bar/ die Resource und key/value/key2/value2/ die einzelnen Parameter mit ihren dazugehörigen Werten darstellen.

**Was ist eine Representation?**

Eine Repräsentation ist im weitesten Sinne die Formatierung einer Resource. Formatierung im Sinne von REST darf man allerdings nicht mit einer Formatierung im Sinne von CSS verwechseln. Hierbei geht es um die Datenstruktur respektive das Datenformat, sprich: liefert der Server die Daten der Resource als XML,JSON, HTML oder in welchem Format auch immer zurück. Hierbei ist irrelevant, ob eine Überschrift in Rot oder Grün dargestellt wird.

Es gibt nach dem Prinzip von REST zwei Möglichkeiten, die Repräsentation in den dazugehörigen Formaten abzurufen. Die erste Methode ist die sogenannte [Content Negotiation](http://de.wikipedia.org/wiki/Content_Negotiation) Methode. Bei dieser Technik wird das Datenformat der Resource über Parameter im Header gesteuert. Diese Parameter sind u.a. **Accept**, **Accept-Charset**, **Accept-Encoding** und **Accept-Language**. Zum Beispiel könnte über den Accept Header der akzeptierte [MIME Typ](http://tools.ietf.org/html/rfc2045) festgelegt werden. Beispiel:

Accept: application/json

Eine weitere Methode ist, das gewünschte Datenformat über die Dateiendung zu steuern. Z.B. könnte ein index.html die Resource als html und ein index.json die Resource als JSON Object zurück liefern.

**REST Methoden**

Die REST Methoden sind von HTTP übernommen.

**GET**

Die GET Methode dient dazu, eine **Resource abzurufen**. Sie hat in der Regel keine Seiteneffekte. Das heißt, dass bei erneutem Aufruf das gleiche Ergebnis zu erwarten ist und die angesprochene Resource nicht verändert wird. Eine mit GET durchgeführte Abfrage kann in der Regel gecached werden, wenn die passenden Parameter in den Header Fields gesetzt worden sind. Diese Methode wird durch jeden Browser unterstützt. Dieses in der Regel durch einfügen der URI in der Addressleiste. Ein Nachteil dieser Methode ist die Sichtbarkeit der URI in der Addressleiste. Dadurch können sich verschiedene Angriffsvektoren auf die Webanwendung ergeben. Zum Beispiel ist ein weitverbreitetes Sicherheits-Risiko von Webanwendungen das Übergeben der SessionID im GET-Request respektive in der Addressleiste. Hierbei könnte durch einfaches abschreiben der Addressleiste eine komplette Session durch eine fremde Personen übernommen werden.

Weiterhin ist die maximale Länge des Requests ebenfalls ein nicht zu unterschätzendes Problem. Zwar gibt die HTTP Spezifikation keine Auskunft über die maximale Länge der einzelnen Header-Fields, allerdings sagt sie aus, dass ein Server, der längere URIs seiner Resourcen zur Verfügung stellt, mit diesen auch umgehen können muss. Ein weiteres Problem der maximalen Länge liegt im Bereich der einzelnen Clients respektive Browsern. So kann es durchaus sein dass z.B. der IE 7 eine maximale Länge von 255 Zeichen in der Addressleiste zulässt. Andererseits akzeptieren einige Webserver auch nur eine bestimmte Länge eines GET Requests. Diese Daten müssen vorher bei den einzelnen Dokumentationen der Clients und Browser nachgeschlagen werden. Siehe dazu auch folgenden Text:

The HTTP protocol does not place any a priori limit on the length of

a URI. Servers MUST be able to handle the URI of any resource they

serve, and SHOULD be able to handle URIs of unbounded length if they

provide GET-based forms that could generate such URIs. A server

SHOULD return 414 (Request-URI Too Long) status if a URI is longer

than the server can handle (see section 10.4.15).

Note: Servers ought to be cautious about depending on URI lengths

above 255 bytes, because some older client or proxy

implementations might not properly support these lengths.

<http://tools.ietf.org/html/rfc2616#section-3.2.1> <http://tools.ietf.org/html/rfc2616#section-9.3>

**POST**

Diese Methode dient dazu, eine neue Sub-Resource auf einem Server anzulegen. Sie kann zum Beispiel folgende Szenarien abdecken:

* Zum Hinzufügen von Zusatzinformationen zu einer bestehenden Resource
* Hinzufügen eines Posting zu einem Forum oder schwarzen Brett
* Senden eines Datenblocks wie das Ergebnis der Abgabe eines Form-Fields
* Erweitern eines Datenbestandes in einer Datenbank

Die eigentliche Funktion der POST Methode wird vom Server anhand der Request-URI identifiziert. Das heißt, dass die POST Methode keine Funktionalität enthält sonder diese durch den Server bereitgestellt werden muss.  
Die neu anzulegende Resource ist einer bestehenden Resource untergeordnet. Diese kann man mit einem Verzeichnis und einer ( neuen ) Datei in einem Dateisystem vergleichen. Die Datei ist dem Verzeichnis untergeordnet.  
Ein POST Request resultiert nicht zwangsläufig in eine neue, durch eine URI identifizierbare Resource. In diesem Fall sollte der Server einen response-status 200 (Ok) oder 204 (No Content) zurückliefern. Wenn allerdings eine neue Resource angelegt worden ist,sollte der Server einen Statuscode 201 ( created ) und den neuen Identifier der neu angelegten Resource zurückliefern.Hierbei ist zu erwähnen, dass der Server die URI automatisch vergibt und der Client keinerlei Einfluss darauf hat.  
POST Requests sind nicht cacheable und können Seiteneffekte haben.

<http://tools.ietf.org/html/rfc2616#section-9.5>

**PUT**

Die PUT Methode dient dazu, eine Resource neu anzulegen oder eine bestehende zu ändern. Hierbei wird die neue URI an die PUT Methode angehangen und die neue Resource unter dieser angelegt. Sollte die Resource schon bestehen, wird diese überschrieben.Sollte eine neue Resource angelegt werden, gesendet der Server einen Statuscode 201 ( created ). Wenn andernfalls eine bestehende Resource geändert wird, sollte der Server einen Stautuscode 200 oder 204 zurück geben. Wenn jedoch ein Fehler aufgetreten ist, sollte der Server einen Fehler Status oder ein 501 ( nicht implementiert ) zurücksenden.  
Die PUT Methode ist nicht cacheable und es können Seiteneffekte entstehen.

Der fundamentale Unterschied zur POST Methode ist, dass die URI in einer POST Methode die Resource identifiziert, die die an ihr übergebenen Daten verarbeitet. Hingegen identifiziert die URI bei einer PUT Methode die mit ihr übergebenen Daten. Der Client kennt bereits die neue URI beim absenden der Daten und der Server sollte nichts an der URI verändern. Wenn doch, gibt er einen 301 ( moved permanently ) zurück und der Client entscheidet dann, wie er weiter verfahren will.

<http://tools.ietf.org/html/rfc2616#section-9.6>

**DELETE**

Diese Methode dient dazu, eine bestehende Resource zu löschen. Diese Methode wird allerdings durch die meisten Server umgeschrieben respektive geblockt. Somit kann der Client selbst bei einem 200 (OK) nicht sicherstellen, dass der Request Serverseitig auch wirklich ausgeführt wurde.  
Diese Methode ist nicht cacheable.

<http://tools.ietf.org/html/rfc2616#section-9.7>

**HEAD**

Die HEAD Methode ist identisch zur GET Methode, außer dass der Server keinen Body im Response mitschicken muss. Diese Methode kann genutzt werden, um Meta Informationen vom Server zu erhalten, ohne dass der ganze Content übertragen wird.

<http://tools.ietf.org/html/rfc2616#section-9.4>