

Tobias Günther

REST in Rails

Teil 1: Theorie einer neuen Web-Architektur in RubyOnRails

Das neue Rails 1.2 bietet wieder einige Detailverbesserungen. Mehr als nur ein Detail dürfte wohl aber ein neues Architektur-Konzept sein: REST (Representational State Transfer). In Rails findet es als einem der ersten Frameworks Einzug in die Welt der Web-Entwicklung.

Es gibt unzählige Variationen, eine Web-Applikation aufzubauen. Genauso zahllos sind deshalb auch die Dokumentationen, die notwendig werden, um immer wieder andere, Entwickler-spezifische Architekturen zu verstehen und zu nutzen. Neben den zahlreichen Variationsmöglichkeiten beim Aufbau einer Website gibt es ein weiteres, hinlänglich bekanntes Problem: die URLs. Hübsch und gut zu bookmarken können sie sein oder aber lang, kryptisch und ein Graus für viele Suchmaschinen-Roboter. REST bietet Lösungen für beide Probleme an. REST ist ein Architektur-Pattern für Web Services. RESTful entwickelte Anwendungen zeichnen sich vor allem durch folgende Vorteile aus:

- klare Architekturen
- saubere URLs

In diesem ersten Artikel zum Thema geht es zunächst um die theoretischen Konzepte hinter REST. Später folgen dann im zweiten Teil Informationen zur praktischen Implementierung in Rails.

Klare Architekturen

Die Architektur einer Anwendung bestimmt unter anderem, wie gut sie zu warten und zu erweitern ist und wie leicht es fremden Entwicklern fällt, den Service von außen zu nutzen oder sich in das System einzuarbeiten. Das ist ein wichtiger Punkt, wenn es um die Zukunfts- und Wachstumsfähigkeit einer Web-Applikation geht. Konventionen und Standards helfen dabei, denn durch sie muss das Rad nicht immer wieder neu erfunden werden. Stattdessen können Best Practices zum Einsatz kommen, die sich in zahlreichen Projekten herausgebildet haben. Alle Entwickler haben eine einheitliche Grundlage, auf der sich einheitlicher Code entwickeln lässt. REST ist solch ein Standard.

Saubere URLs

Eine Web-Applikation (oder allgemeiner: ein Web-Service) wird nach außen hin durch URLs repräsentiert. Dafür gibt es unendlich viele Implementierungs-Möglichkeiten. Diese Vielfalt bringt den Nachteil, dass jeder Service für Außenstehende erst einmal wieder unbekannt und unverständlich ist. Dabei sind Außenstehende nicht nur fremde User und Entwickler, die den Service einbinden oder sich einarbeiten müssen. Auch andere Services wie Robots oder Clients, denen erst einmal beigebracht werden muss, welche Parameter im Service was bewirken, profitieren von Standards.

Hier einige Beispiele für einen fiktiven Adressen-Service mit vier (von unendlich vielen weiteren) denkbaren URLs, die alle das Gleiche bewirken – eine Adresse löschen:

Aktion	Beispiel-URL
GET-Call	http://beispiel.de/addresses.php?id=123&do=delete
POST-Call	http://beispiel.de/addresses.php mit CGI-Stil-Bodyid=123&do=delete
POST-Call	http://beispiel.de/addressAdmin/command=delete mit CGI-Stil-Bodyid=123
POST-Call	http://beispiel.de/addresses/destroy/123
DELETE-Call	http://beispiel.de/addresses/123

Die erste URL, die per GET versendet wird, verletzt eine goldene Regel: GET-Requests sollen keine Ressourcen verändern – sie sind nur für Abfragen zuständig. Die URL, per GET zum Beispiel auf einem handelsüblichen Link, löscht aber etwas, nämlich die Adresse mit der ID 123. Wird der Link nun als Bookmark angelegt und erneut aufgerufen oder durch einen Suchmaschinen-Crawler aktiviert, wird die Aktion abermals aufgerufen, womöglich auf einem anderen Datensatz, der jetzt die ID 123 hat. Das ist sicherlich nicht im Sinne des Erfinders und somit ist diese Methode disqualifiziert. Die zweite, dritte und vierte Variante kommen ohne den Fehler aus. Dennoch gibt es auch hier ein Problem: Ressource und Aktion werden vermischt. Bei einer normalen CRUD-Aktion (Create Read Update Delete) ist es nämlich nicht nötig, die Aktion mit in die URL zu packen. Praktisch bedeutet das nämlich, dass jede Aktion unnötigerweise wieder eine unterschiedliche URL bekommt. Ein Vergleich zwischen klassischen Rails-URLs und RESTful-Rails-URLs zeigt den Unterschied:

Aktion	URL ohne REST	URL mit REST
show	/addresses/show/123	/addresses/123
delete	/addresses/destroy/123	/addresses/123
update	/addresses/update/123	/addresses/123
create	/addresses/create	/adresse

Da bei REST Ressource und Aktion voneinander getrennt bleiben, gilt hier für eine Ressource immer die gleiche URL – egal, welche Aktion mit ihr durchgeführt werden soll. Zwei Aspekte ermöglichen das in REST: die Wiederentdeckung von HTTP und das Denken in Ressourcen.

HTTP ist mehr als GET & POST

HTTP hätte allen Grund, beleidigt zu sein, denn jahrelang wurde missachtet, dass unser aller Lieblings-Protokoll mehr kann als GET und POST. REST erinnert sich nun an dieses „Mehr“ und holt zusätzlich die Methoden PUT und DELETE mit ins Boot. In REST dreht sich alles um Ressourcen. Und im Ressourcen-orientierten Ansatz liegt das Geheimnis von REST: Es betrachtet den Server als eine große Ansammlung von Ressourcen, die fast immer als Substantive auftreten, zum Beispiel Autos, Adressen oder Bestellungen. Diese Ressourcen werden in REST als URLs repräsentiert. Merken darf man sich in diesem Zusammenhang zwei Dinge:

- Die URLs sind immer eindeutig.
- Die URLs bezeichnen eine Sache (die Ressource) anstatt einer Aktion.

Nun drängt sich allerdings die Frage auf, wo die Aktionen verbleiben, denn die bloße Ressource reicht in den wenigsten Fällen aus. Wir wollen sie auch manipulieren können.

Aktionen sind HTTP-Methoden

Wenn in der REST-Welt die Ressourcen als Substantive vorkommen, dann sind die Aktionen die Verben. Und hier wird die Brücke zurück zu den (teilweise in Vergessenheit geratenen) HTTP-Me-

thoden geschlagen. Denn in REST besteht jeder Request aus einer URL und einer HTTP-Methode. Zwar ist es richtig, dass auch in traditionellen Architekturen ein Request immer aus URL und HTTP-Methode besteht, allerdings werden die HTTP-Methoden in REST sehr bewusst und gezielt eingesetzt. Das ist beim herkömmlichen Standard-Repertoire „GET und POST“ so nicht der Fall.

Jede HTTP-Methode repräsentiert eine ganz spezielle Aktion (zur Verdeutlichung mit der SQL-Entsprechung in Klammern):

HTTP-Methode	Aktion (vergl. SQL-Kommando)
GET	anzeigen (select)
POST	neu erstellen (insert)
PUT	aktualisieren (update)
DELETE	löschen (delete)

Das Interessante an einer REST-Implementierung ist: Um eine der Aktionen auf eine Ressource anzuwenden, wird, wie oben bereits gezeigt, immer die gleiche URL (mit kleinen Einschränkungen) verwendet. Denn die URL ist unabhängig von der durchgeführten Aktion. Sie beschreibt nur die Ressource. Nehmen wir wieder die Adressen als beispielhafte Ressourcen: Die URL, um mit der Adresse (=Ressource) mit der ID 1 zu arbeiten, lautet immer `http://www.beispiel.de/addresses/1`. Wenn sie angezeigt werden soll, wird die URL als GET abgesetzt. Wollen wir sie ändern, verwenden wir PUT als HTTP-Methode. Und ein Löschen der Adresse (Ressource) funktioniert über ein DELETE mit derselben URL. REST muss in Projekten allerdings nicht immer in Reinform vorkommen. Für viele Anwendungsfälle ist eine Mixtur zwischen klassischen Architekturen und REST absolut in Ordnung und praktikabel.

REST in Rails

Die Version 1.2 ist die erste Rails-Version, die REST umsetzt. Zum Redaktionsschluss lag noch keine finale Version 1.2 vor. In Edge-Rails, dem aktuellsten Rails-Extrakt, sind aber alle Konzepte bereits unterstützt. Wie RubyOnRails nun ganz genau REST implementiert, wird im zweiten Teil dieser Artikel-Serie gezeigt. Die Vorteile, die RESTful entwickelte Anwendungen in Rails mit sich bringen, sollen aber nicht so lange verschwiegen werden. Denn außer der klaren Architektur und den sauberen URLs bietet REST in Rails noch weitere Verbesserungen:

- Unterschiedliche Ausgabeformate.
- Mit REST ist es nun ein Kinderspiel, unterschiedliche Ausgabeformate in Rails anzubieten: Die Wahl von HTML, RSS, Atom, JS, YAML oder XML macht im Idealfall nur eine Extrazeile im Controller notwendig. Das bedeutet weniger Code.
- Die MultiClient-fähigen Anwendungen, die beim Anbieten unterschiedlicher Ausgabeformate entstehen, können mit deutlich weniger Code entwickelt werden.
- Bessere CRUD-Unterstützung: Controller, die RESTful entwickelt werden, sind gleichzeitig auch nach dem CRUD-Prinzip entwickelt. Das verbessert die Struktur der Anwendung.

Wie schon erwähnt, wird der zweite Teil des Artikels einen praxisnahen Einblick geben. Wer es bis dahin nicht aushält, der kann schon in EdgeRails mit REST experimentieren.

DER AUTOR



Tobias Günther leitet seit drei Jahren die Web-Agentur puremedia in Stuttgart (www.puremedia-online.de). Schwerpunkte seiner Arbeit sind die Themen AJAX, JavaScript und Ruby on Rails.