

# REST

---

eine neue Architektur für Webanwendungen

- Internet-Agentur
  - AJAX / JavaScript
  - Ruby on Rails
  - Typo3 / Extension-Programmierung
  
- Kunden:



DAIMLERCHRYSLER

# Warum REST?

---

# Wieso? Weshalb? Warum?

---

- Unzählige Möglichkeiten und Herangehensweisen, um eine Web-Applikation aufzubauen

==> jede Architektur ist anders, (viel) Dokumentation ist nötig

# Wieso? Weshalb? Warum?

---

- Unzählige Möglichkeiten und Herangehensweisen, um eine Web-Applikation aufzubauen

==> jede Architektur ist anders, (viel) Dokumentation ist nötig

REST schafft klare Architekturen --> Wartbarkeit / Erweiterbarkeit

Standardisierung/Konvention schafft wiederverwendbares Wissen

# Wieso? Weshalb? Warum?

---

- diese Vielzahl wird auch in URLs sichtbar  
=> unsaubere, kryptische URLs

## Löschen / updaten ohne REST

<http://beispiel.de/articles/index.php?action=delete&id=123>

<http://beispiel.de/articles/index.php?action=update&id=123&name=geaendert>

# Wieso? Weshalb? Warum?

---

- diese Vielzahl wird auch in URLs sichtbar  
==> unsaubere, kryptische URLs

## Löschen / updaten ohne REST

<http://beispiel.de/articles/index.php?action=delete&id=123>

<http://beispiel.de/articles/index.php?action=update&id=123&name=geaendert>

REST sorgt für saubere, sinnvolle und suchmaschinenfreundliche URLs

## Löschen / updaten mit REST

<http://www.beispiel.de/articles/123>

<http://www.beispiel.de/articles/123>

## Wieso? Weshalb? Warum?

---

- um unterschiedliche Formate auszugeben (HTML, XML, JSON...) ist oft viel (ähnlicher!) Code notwendig
  - ==> unnötiger, duplizierter Code für versch. Ausgabeformate

# Wieso? Weshalb? Warum?

---

- um unterschiedliche Formate auszugeben (HTML, XML, JSON...) ist oft viel (ähnlicher!) Code notwendig

==> unnötiger, duplizierter Code für versch. Ausgabeformate

**REST reduziert Code, v.a. für die Ausgabe versch. Formate**

# Wie funktioniert REST?

---

# HTTP ist mehr als GET und POST

---

HTTP umfasst mehr Methoden als nur

- GET (z.B. Anfrage über Adress-Zeile des Browsers) und
- POST (z.B. Absenden eines Formulars)

...nämlich u.a.

- PUT
- DELETE

Reichen GET und POST nicht aus???

- PUT und DELETE existieren nicht umsonst
- GET und POST werden “missbraucht”

# Ressourcen und Aktionen

---

- Alles dreht sich um **Ressourcen** und **Aktionen**
  - Ressourcen (Substantive) – worum geht es?  
Adressverwaltung --> Adresse, Online-Shop --> Produkt...
  - Aktionen (Verben) – was soll mit der Ressource passieren?

**Klassisch:** Vermischung zwischen Ressource und Aktion in der URL  
<http://beispiel.de/articles/index.php?action=delete&id=123>

**REST:** saubere Trennung von Ressource und Aktion – nur R. in der URL  
<http://beispiel.de/articles/123>

- Ressourcen (Substantive) = URLs
  - URLs sagen lediglich, **welche** Ressource manipuliert werden soll, nicht mehr die Art der Manipulation
- Aktionen (Verben) = HTTP-Methoden / HTTP-Verben
  - HTTP-Verben (PUT, POST, GET, DELETE) sagen, was mit der Ressource geschehen soll

## Saubere URLs

---

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	/addresses

Frage: Woher weiß die Applikation, was sie beim Aufruf von  
"/addresses/123"  
tun soll???

Antwort: HTTP-Methode!

## Die Realität

- Ruby on Rails bietet als eines der ersten Frameworks hervorragende Unterstützung für REST

## Die Einschränkungen

- REST ist kein Allheilmittel
- REST passt nicht für jedes Problem
- REST muss nicht in Reinform verwendet werden – Mischung klassisch/REST

## Die Vorteile

- klare, standardisierte Architekturen
- weniger Code
- saubere URLs
- einfachere Unterstützung verschiedener Ausgabeformate (v.a. in Rails)