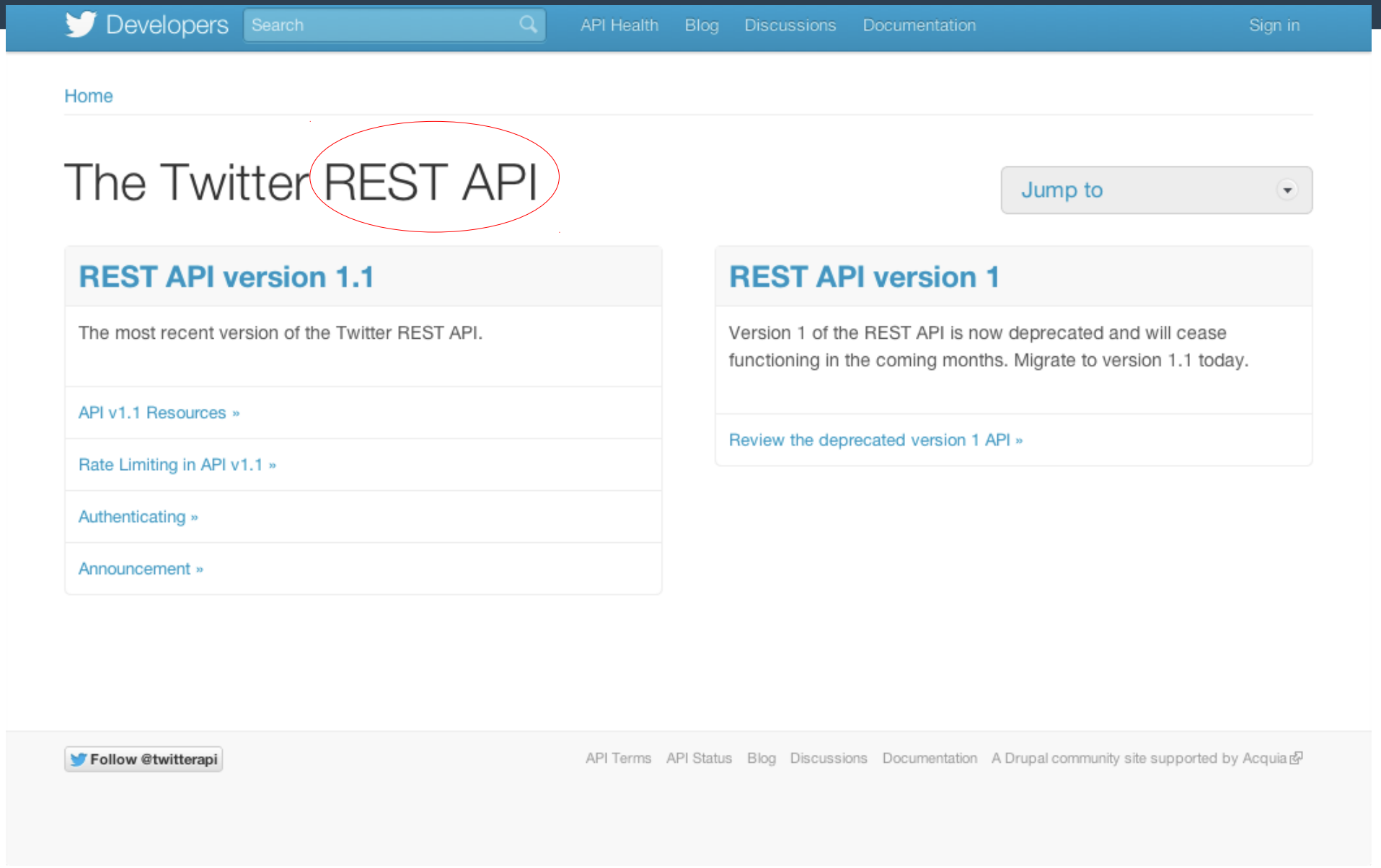
RESTful Webservice APIs


Developers
Search
API Health
Blog
Discussions
Documentation
Sign in
Home
The Twitter REST API
Jump to
REST API version 1.1
The most recent version of the Twitter REST API.
API v1.1 Resources »
Rate Limiting in API v1.1 »
Authenticating »
Announcement »
REST API version 1
Version 1 of the REST API is now deprecated and will cease functioning in the coming months. Migrate to version 1.1 today.
Review the deprecated version 1 API »
Follow @twitterapi
API Terms
API Status
Blog
Discussions
Documentation
A Drupal community site supported by Acquia

LinkedIn Developers

Bernhard Haslhofer

Home | Why Develop With Us | Case Studies | Documentation | Support | Blog

API Overview

- People
- Share and Social Stream
- Groups
- Communications
- Companies
- Jobs

People

Leverage LinkedIn as an identity authority for application registration and signin with the benefits of simplifying the need for users to enter additional data.

REST | JavaScript

```
http://api.linkedin.com/v1/people/~:(first-name,last-name,headline,picture-url)
http://api.linkedin.com/v1/people/~/connections
http://api.linkedin.com/v1/people-search?keywords=Hacker
http://api.linkedin.com/v1/people-search:(people,facets)?facet=location,us:84
```

Choose implementation type REST | JavaScript

Share and Social Stream

Use the share API for seamless integrations for content creators to distribute content into the LinkedIn network updates stream. Allow users to consume insights and content from their professional network.

REST | JavaScript

```
http://api.linkedin.com/v1/people/~/shares
http://api.linkedin.com/v1/people/~/network/updates
http://api.linkedin.com/v1/people/~/network/updates?scope=self
```

Dropbox

Get free space! Bernhard Haslhofer

- API home
- Developer blog
- My apps
- Getting started
  - Core concepts
  - Setup
  - Authentication
  - Files and folders
- Reference
  - Development kits
  - REST API
  - Best practices
  - Branding guide
  - Terms and conditions
- Developer forum
- Dropbox API support

Authentication
- /request_token
- /authorize
- /access_token

REST API

The REST API is the underlying interface for all of our official Dropbox mobile apps and our SDKs. It's the most direct way to access the API. This reference document is designed for those interested in developing for platforms not supported by the SDKs or for those interested in exploring API features in detail.

General notes

SSL only

We require that all requests are done over SSL.

App folder access type

The default root level access type, **app folder** (as described in core concepts), is referenced in API URLs by its codename `sandbox`. This is the only place where such a distinction is made.

UTF-8 encoding

Every string passed to and from the Dropbox API needs to be UTF-8 encoded. For maximum compatibility, normalize to Unicode Normalization Form C (NFC) before UTF-8 encoding.

Version numbers

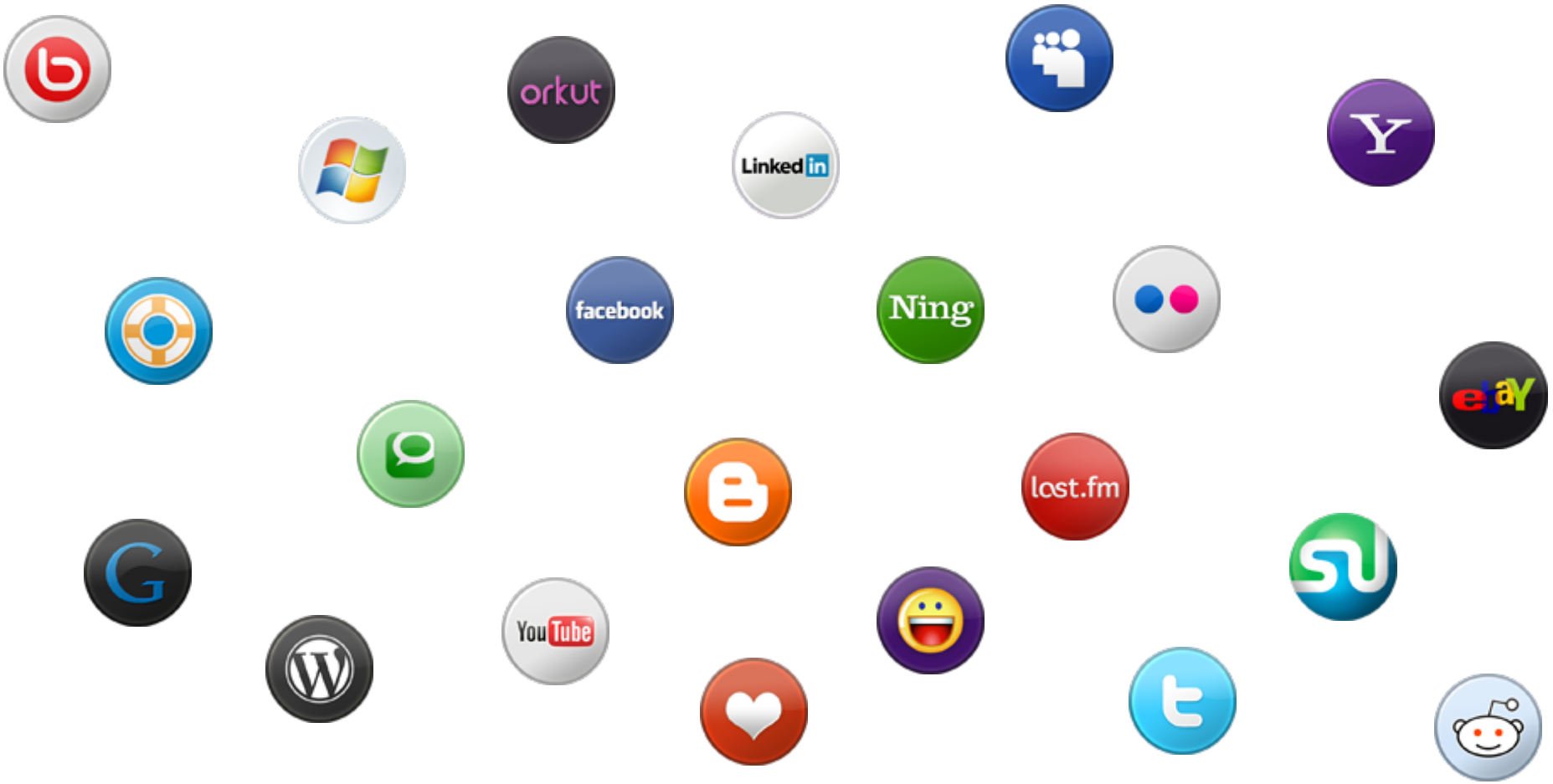
The current version of our API is version 1. Most version 0 methods will work for the time being, but some of its methods risk being removed (most notably, the version 0 API methods `/token` and `/account`).

Date format

All dates in the API are strings in the following format:

```
"Sat, 21 Aug 2010 22:31:20 +0000"
```

In code format, which can be used in all programming languages that support `strftime` or `strptime`:


orkut
LinkedIn
facebook
Ning
lost.fm
YouTube

Основи на Web

Internet ≠ World Wide Web

Основи на Web

- **Ключови архитектурни компоненти**
  - Идентификация: ???
  - Взаимодействие: ???
  - Стандартизирани формати на данните: ???, ???, ???

Основи на Web

- **Ключови архитектурни компоненти**
  - Идентификация: URI
  - Взаимодействие: HTTP
  - Стандартизирани формати на данните: HTML, XML, JSON, etc

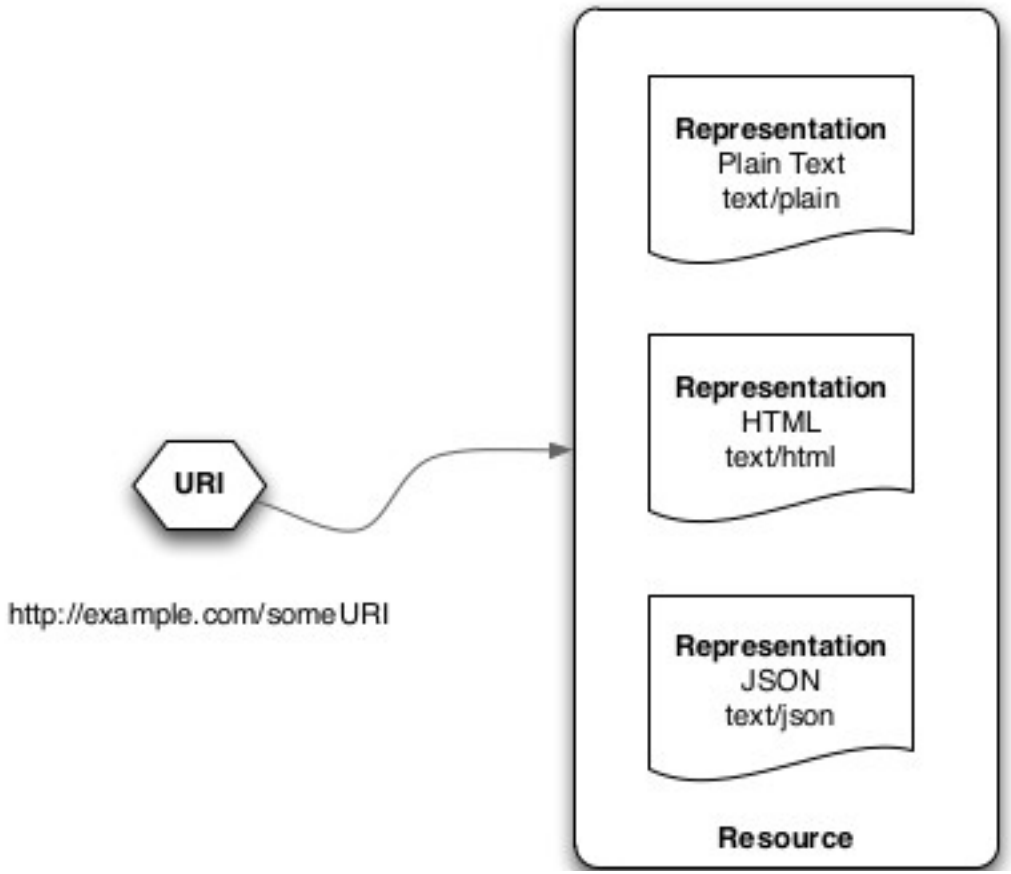
URIs / Ресурси

- **URIs идентифицират интересни ”неща”**
  - Документи в Web
  - Аспекти свързани с множеството от данни
- **HTTP URIs наименоват и адресират ресурси в Web-базираните системи**
  - един URI идентификатор наименова и адресира един ресурс
  - един ресурс може да има повече от едно име
    - http://foo.com/sogware/latest
    - http://foo.com/sogware/v1.4

Представяне на ресурс

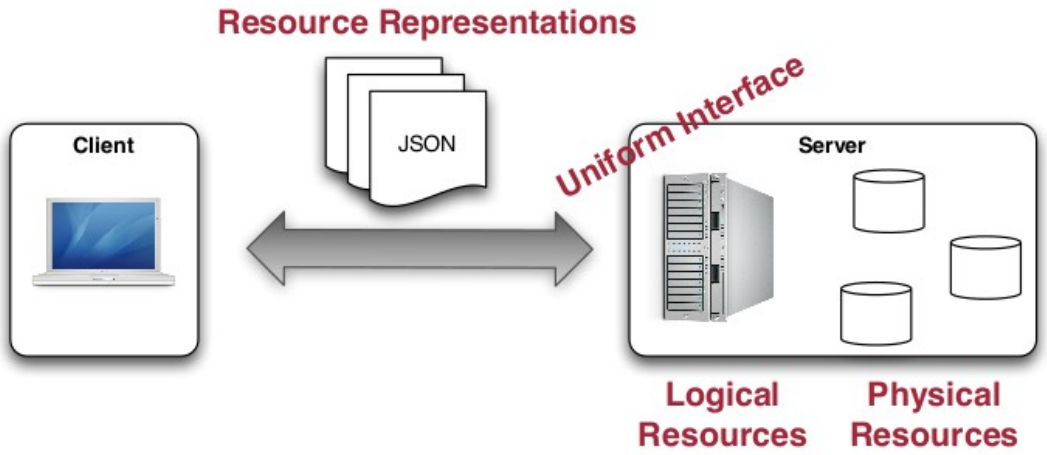
- **Даден ресурс може да има повече от едно представяне**

- **Представянето може да бъде в различен формат**
  - HTML
  - XML
  - JSON
  - ...

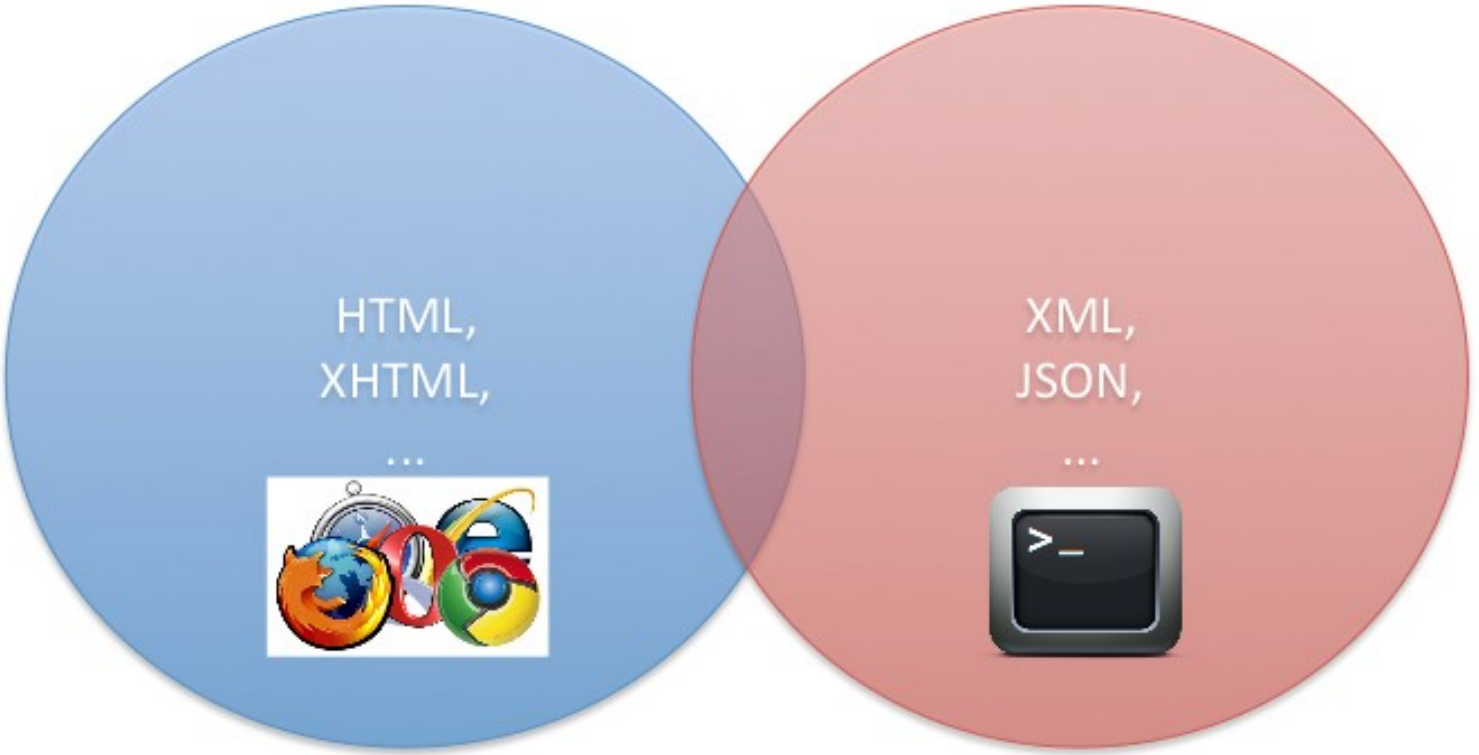


Взаимодействие с ресурсите

- **Взаимодействие посредством представянето на ресурса**
  - Не със самия ресурс
  - Представянето може да е в различне формат (определен от media-type)
- **Всеки ресурс реализира стандартен унифициран интерфейс (HTTP)**
  - Малко множество от действия (verbs) приложено върху голямо множество от съществителни (nouns)
  - Действията са универсални, а не определяни за всяко приложение



Документ / Формати на данните



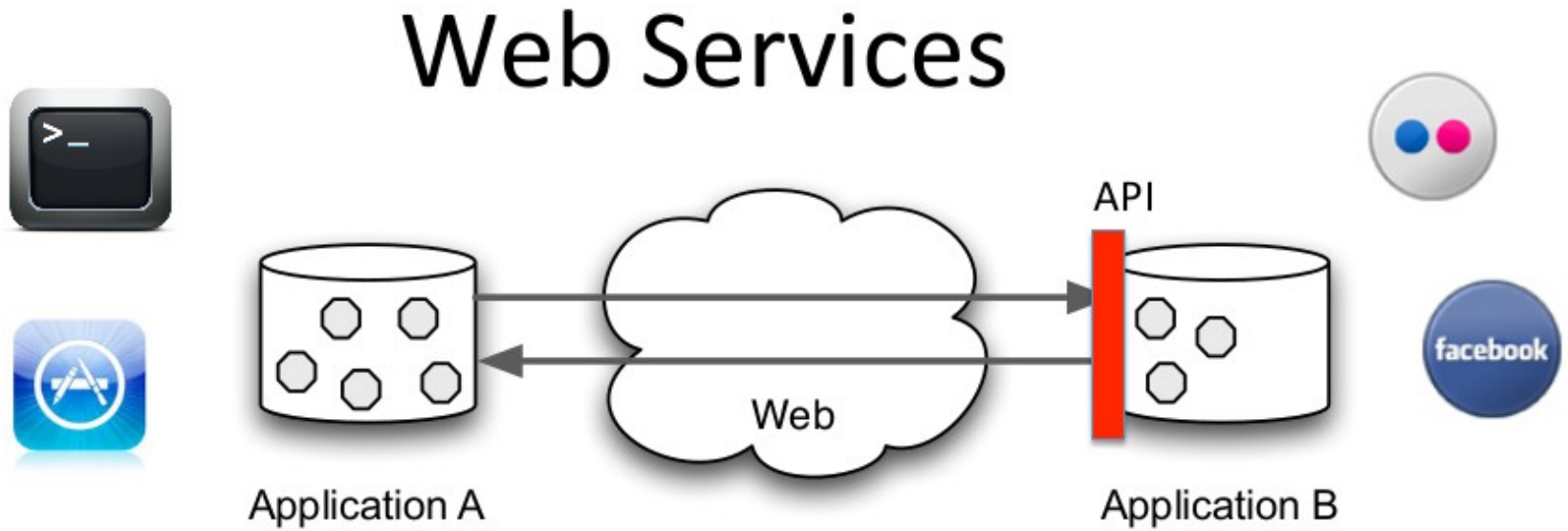
Display data

Transport and store data

(Web) APIs

- **Application Programming Interface**
- **Определя как софтуерните компоненти взаимодействат един с друг**
  - Пример: Java API, 3rd party library APIs
  - Обикновено върви с документация (howtos)
- **Web API: определя как приложения комуникират едно с друго посредством Web (HTTP, URI, XML, etc.)**

Web услуги



- **Примерни операции:**
  - Публикуване на снимка във Flickr
  - Поръчване на книга от Amazon
  - Публикуване на съобщение на Facebook стената на приятел
  - Обновяване на потребителската снимка във Foursquare

Web услуги

- **“Web услуги” ≈ “Web APIs”**
- **Изградени върху принципите за проектиране и архитектурните компоненти на Web**
- **Предоставя определени операции**
- **Обменя структурирани данни в стандартни формати (JSON, XML и други)**

RESTful APIS

АРХИТЕКТУРНИ ПРИНЦИПИ

RESTful Web услуги

- **REST = Representational State Transfer**
  - Базиран върху дисертацията на Roy Fielding’s от 2000г. (глава 5)
- **Архитектурен стил за изграждане на свободно свързани системи (loosely coupled)**
- **Сам по себе си Web е пример за този стил**
- **Web услугите се базират и изграждат върху този стил**

Resource-Oriented Architecture

- **Набор от принципи за проектиране и изграждане на RESTful Web услуги**
  - Адресируемост (Addressability)
  - Унифициран интерфейс (Uniform interface)
  - Свързаност (Connectedness)
  - Без запазване на състоянието (Statelessness)

Адресируемост

- **Адресируемо приложение**
  - Предоставя интересни аспекти от базата данни посредством ресурси
  - Предоставя URI за всяка част от информацията която предоставя
  - Представлява набор от URI адреси

Адресируемост

- **Ресурс**
  - Представлява всяко нещо което е достатъчно важно за да бъде съотнесено като част от системата
  - Обикновено неща за които
    - Желаете да предоставите информация относно
    - Могат да бъдат представени като поток от данни - примерно:
      - актьори
      - филми
  - Всеки ресурс трябва да притежава поне едно име (URI)

Адресируемост

- **Имена на ресурсите (URIs)**
  - URI представлява името и адреса на ресурса
  - URI идентификатора трябва да е описателен

примерно

`http://example.com/movies`

вместо

`http://example.com/overview.php?list=all&type=movie`

Resource-Oriented Architecture

- **Набор от принципи за проектиране и изграждане на RESTful Web услуги**
  - Адресируемост (Addressability)
  - Унифициран интерфейс (Uniform interface)
  - Свързаност (Connectedness)
  - Без запазване на състоянието (Statelessness)

Унифициран интерфейс

- **Едно и също множество от операции приложено върху всичко (всеки ресурс)**
- **Малко множество от действия (methods) приложено върху голямо множество от съществителни (ресурси)**
  - Действията са универсални, а не определяни за всяко приложение

Унифициран интерфейс

- **HTTP протоколът дефинира множеството от действия (methods) прилагани върху ресурси идентифицирани с URI**

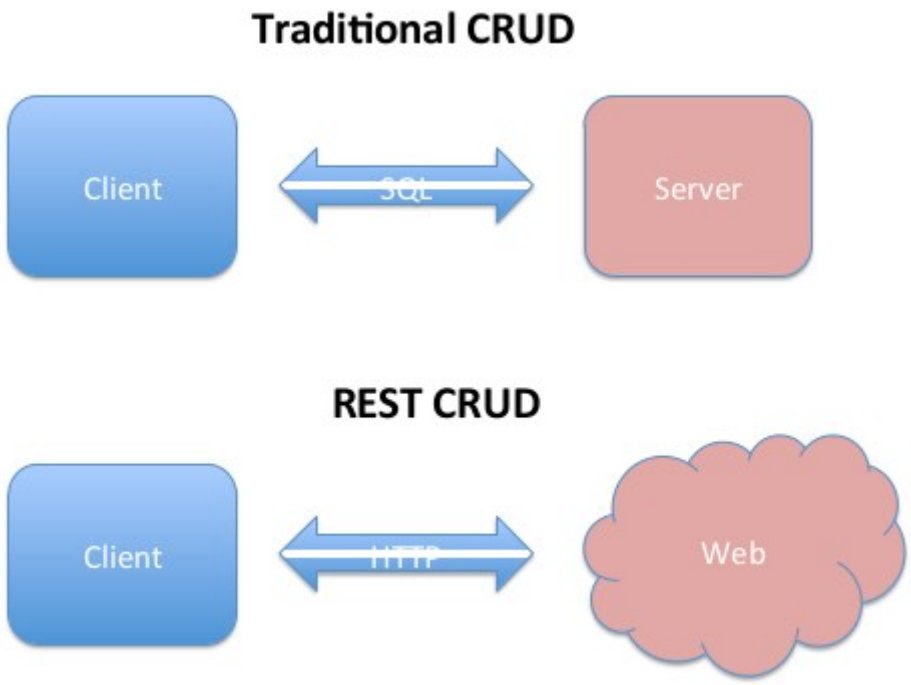
- **Кои HTTP методи познавате?**

Унифициран интерфейс

- **RESTful Web услугите използват пълните възможности на HTTP**
  - HTTP Методи: GET, POST, PUT, DELETE, (...)
  - HTTP Хедъри: Authorization, Content-Type, Last-Modified
  - HTTP Кодове: 200 OK, 304 Not Modified, 401 Unauthorized, 500 Internal Server Error
  - HTTP съдържание: “пликът” съдържащ данните за транспортиране от А до Б

Унифициран интерфейс

- **HTTP предоставя всички необходими методи за манипулиране на Web ресурси (CRUD interface)**
  - Create = POST (or PUT)
  - Read = GET
  - Update = PUT (or PATCH)
  - Delete = DELETE



Безопасно и идемпотентно поведение

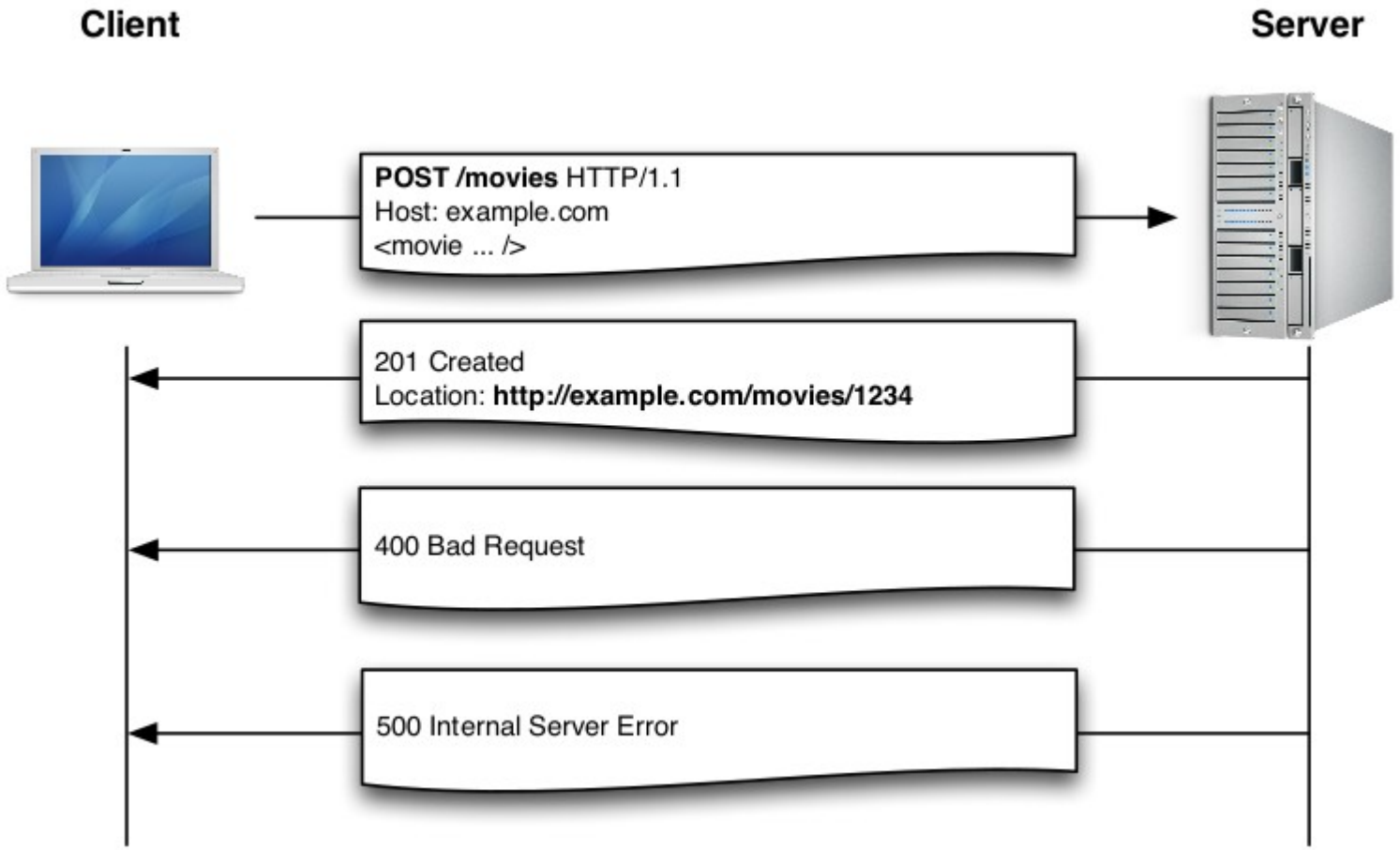
- **Безопасните** методи могат да бъдат игнорирани или повторени без странични ефекти: GET и HEAD

- **Идемпотентните** методи могат да бъдат повторени без странични ефекти: PUT и DELETE

- Останалите методи трябва да бъдат обработвани изключително внимателно: POST

Унифициран интерфейс

- **CREATE** създаване на нов ресурс с HTTP POST



Примерна POST заявка

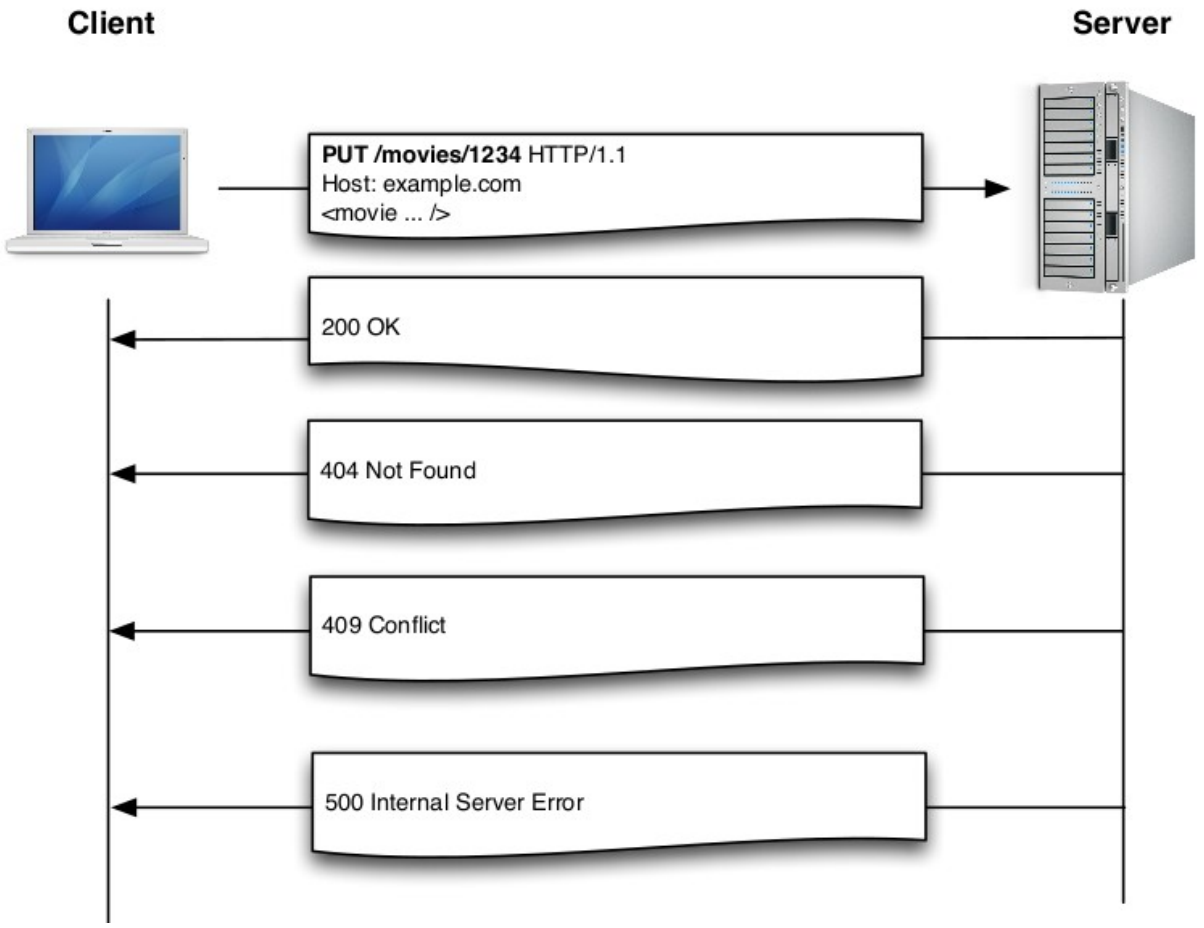
```
POST /movies HTTP/1.1
Host: example.com
...
<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

Семантика на POST заявката

- **POST създава нов ресурс**
- **Сървърът определя URI на ресурса**
- **POST не е идемпотентен или безопасен:**
  - Две или повече еднавки POST заявки водят до странични ефекти
  - Human Web:
    - “Do you really want to resend this form again?”
  - Programmatic Web
    - Всяка следваща заявка създава нов ресурс

Унифициран интерфейс

- **CREATE** създаване на нов ресурс с HTTP PUT



Примерна PUT заявка

```
PUT /movies/1234 HTTP/1.1
Host: example.com
...
<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

Семантика на PUT заявката

- **PUT създава нов ресурс**
- **Клиентът определя URI на ресурса**
- **PUT е идемпотентен**
  - Няколко PUT заявки не водят до странични ефекти
  - Но могат да доведат до промяна на ресурса

Създаване с PUT или POST ?!

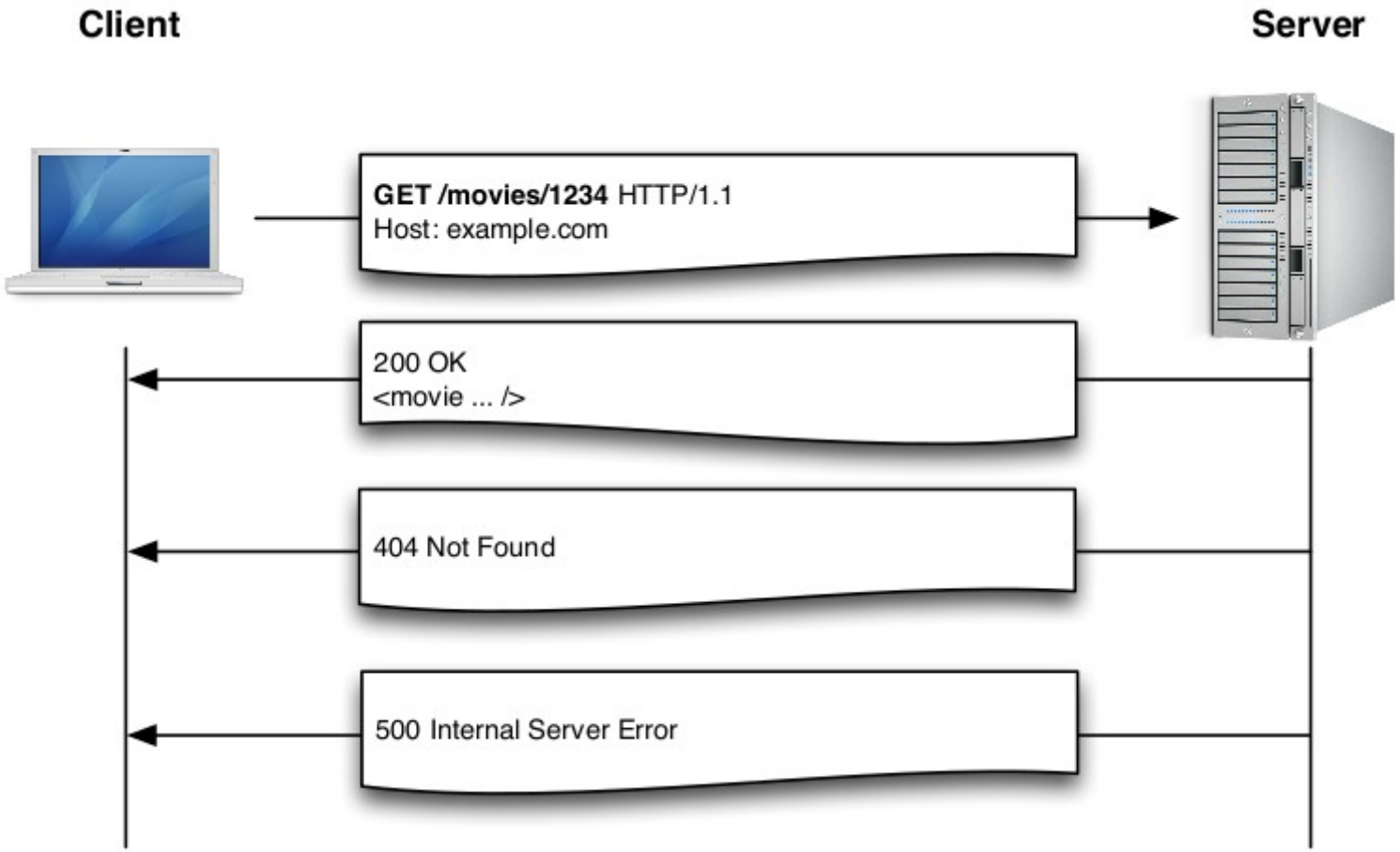
- **Правилния отговор: зависи**
- **Съображения**
  - PUT ако клиента
    - Може да определя URI
    - Изпраща пълното представяне на ресурса
  - POST ако сървъра определя URI (следвайки алгоритъм)
  - Някои защитни стени пропускат само GET и POST
  - POST е най-предпочитания начин

Пример за създаване на ресурс посредством PUT

```
# Create Amazon S3 bucket

PUT / HTTP/1.1
Host: colorpictures.s3.amazonaws.com
Content-Length: 0
Date: Wed, 01 Mar 2016 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

# Add Object to a bucket

PUT /my-image.jpg HTTP/1.1
Host: colorpictures.amazonaws.com
Date: Wed, 12 Oct 2016 17:50:00 GMT
```

Унифициран интерфейс

- **READ** прочитане на ресурс с HTTP GET



Пример за GET Заявка / Отговор

Request:

```
GET /movies/1234 HTTP/1.1
Host: example.com
Accept: application/xml
...
```

Response:

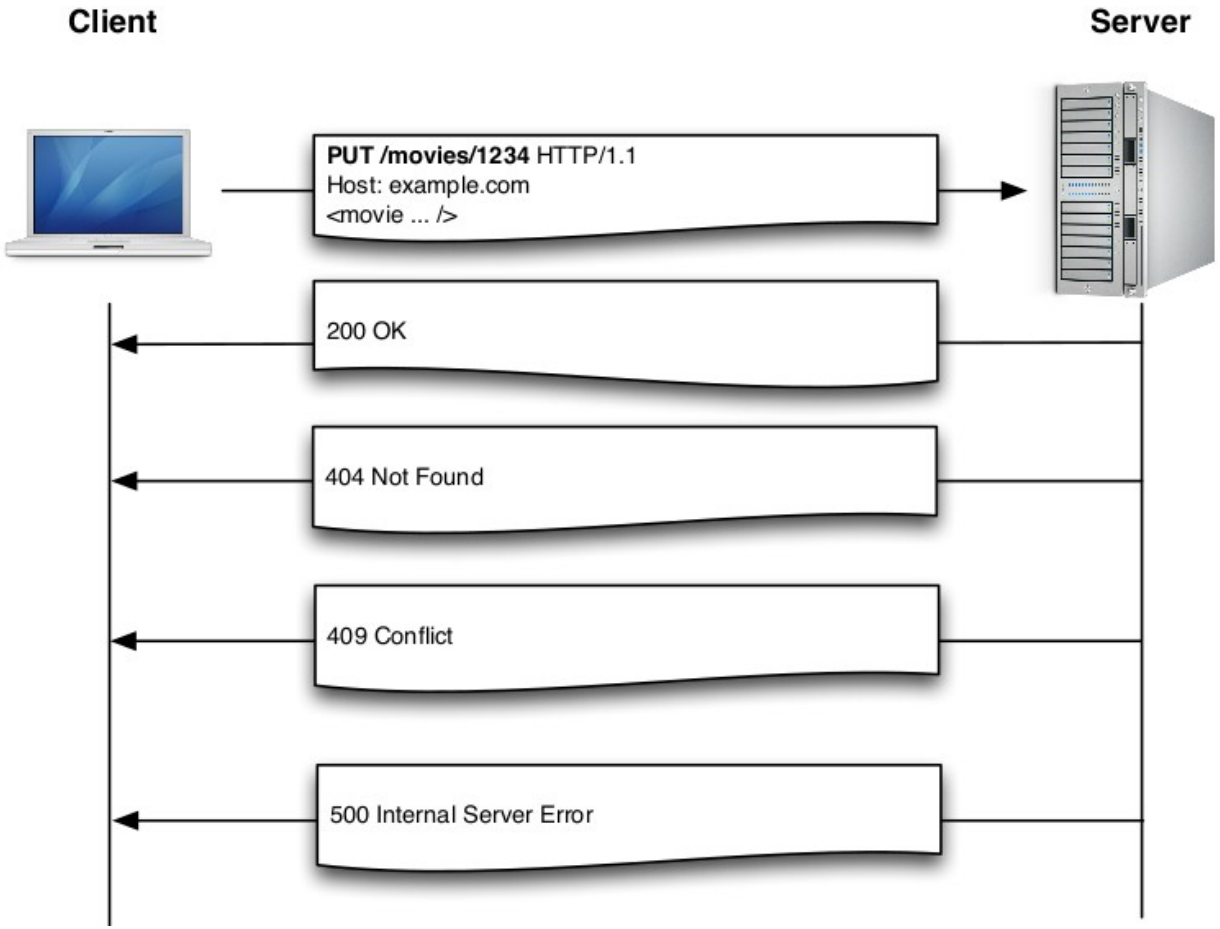
```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
</movie>
```

Семантика на GET заявката

- **GET извлича представянето (текущото състояние) на ресурса**
- **GET е безопасен метод**
  - Не променя състоянието на ресурса
  - Няма странични ефекти
- **Ако GET заявката доведе до грешка**
  - Изпрати нова GET заявка!
  - Няма проблем защото метода е безопасен

Унифициран интерфейс

- **UPDATE** актуализация на съществуващ ресурс с HTTP PUT

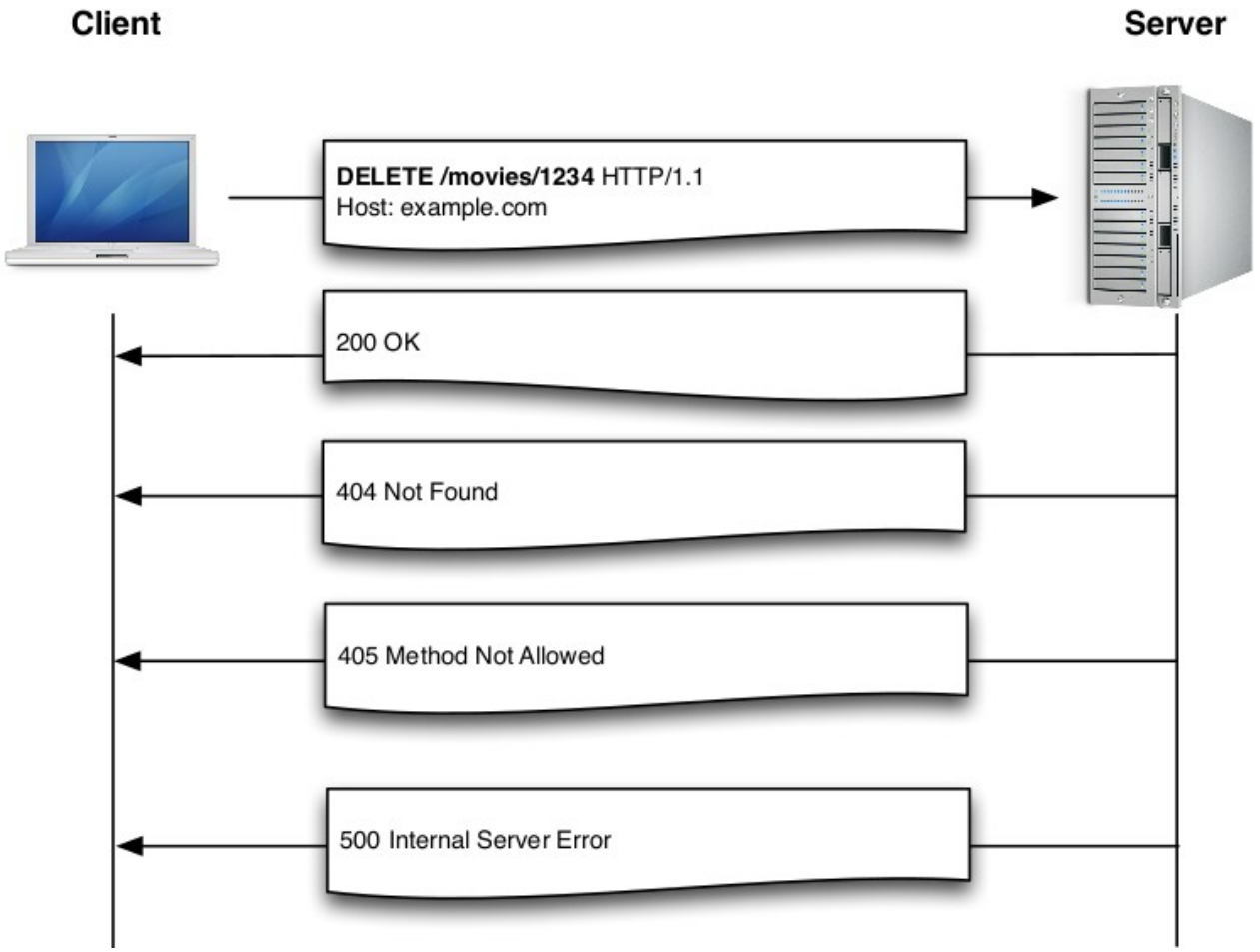


Когато PUT не успее

- **Ако се получи грешка 5xx error или 4xx errors**
  - Просто повтаряме PUT заявката!
  - Няма проблем, защото PUT е идемпотентен
- **Ако се получи грешка от състоянието на ресурса – компенсиращи действия и повтаряне на PUT заявката**
  - 409 Conflict (пр. при опит да се смени потребителското име с ново, което вече е заето)
  - 417 Expectation Failed (сървъра не приема представянето на ресурса – оправяне на XML/JSON грешки)

Унифициран интерфейс

- **DELETE** изтриване на съществуващ ресурс с HTTP DELETE



Семантика на DELETE заявката

- **Спира достъпа до съответния ресурс**
  - Логическо изтриване
  - Не задължително физическо изтриване
- **Ако DELETE заявката не успее**
  - Опитайте отново!
  - DELETE е идемпотентна заявка

Resource-Oriented Architecture

- **Набор от принципи за проектиране и изграждане на RESTful Web услуги**
  - Адресируемост (Addressability)
  - Унифициран интерфейс (Uniform interface)
  - Свързаност (Connectedness)
  - Без запазване на състоянието (Statelessness)

Свързаност

- **При RESTful услугите, представянето на ресурса включва hypermedia**
- **Документите могат да съдържат не само данни, но и хипервръзки към други ресурси**

```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
   <title>The Godfather</title>
   <synopsis>...</synopsis>
   <actor>http://example.com/actors/567</actor>
</movie>
```

Resource-Oriented Architecture

- **Набор от принципи за проектиране и изграждане на RESTful Web услуги**
  - Адресируемост (Addressability)
  - Унифициран интерфейс (Uniform interface)
  - Свързаност (Connectedness)
  - Без запазване на състоянието (Statelessness)

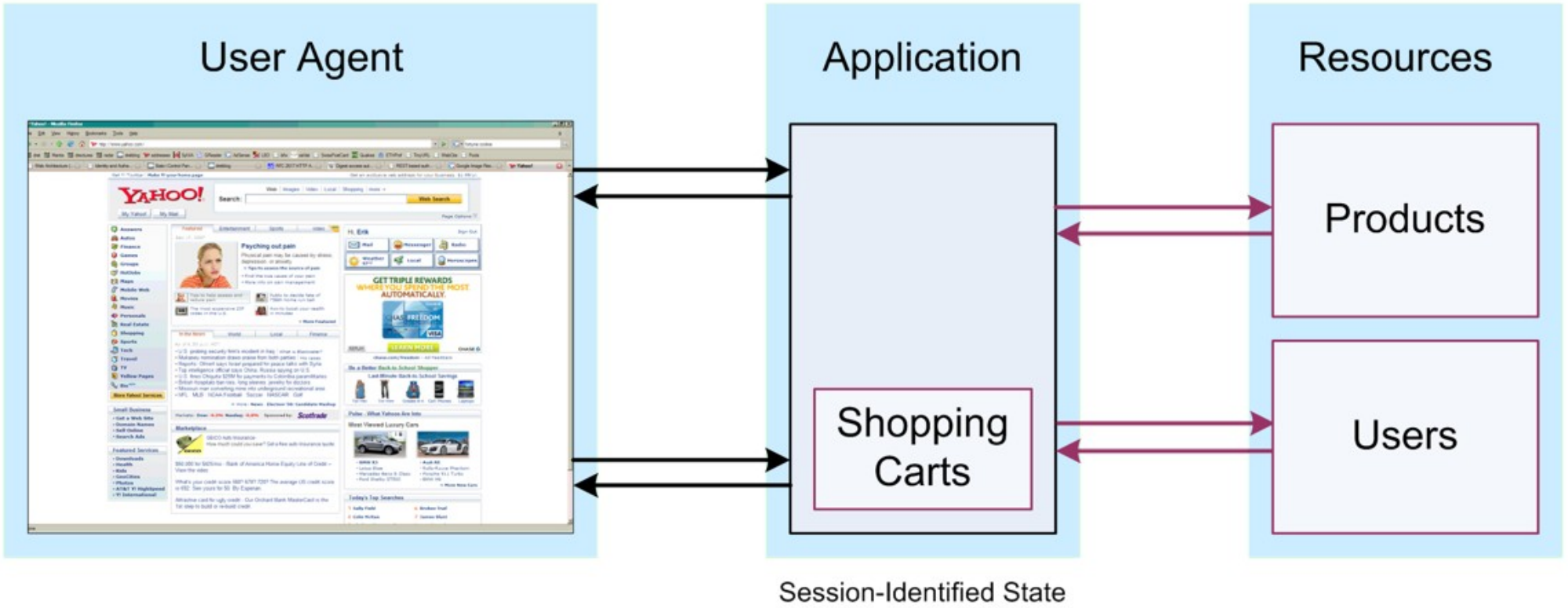
Statelessness

- **Всяка HTTP заявка се изпълнява в пълна изолация (без зависимост от предишни)**
- **За целта заявката включва цялата необходима информация необходима на сървъра да я изпълни**
- **Сървъра не разчита на информация от предишни заявки**
  - Автентикация и оторизация - клиента включва необходимата информация във всяка заявка

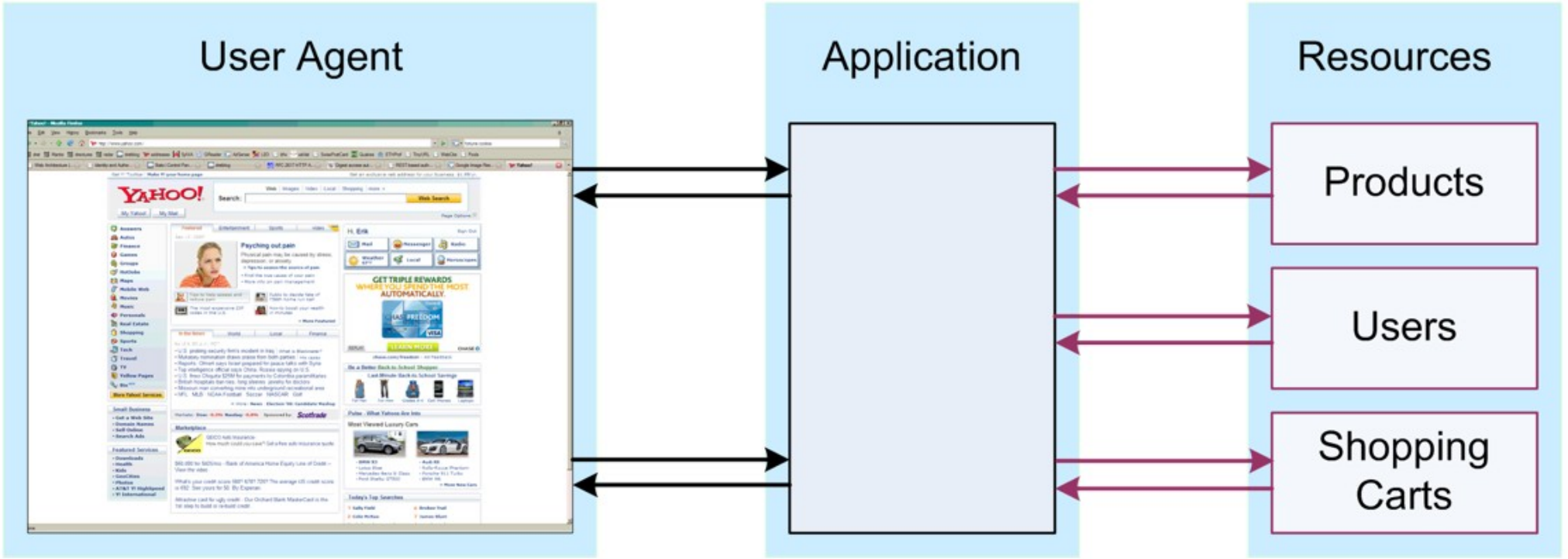
Statelessness

- **Това ограничение не означава “приложения без запазване на състоянието”!**
  - За много RESTful приложения, състоянието е важно
  - Пример: пазарска кошница
- **Означава да се премести запазването на състоянието в клиента или ресурса**
- **Запазване на състоянието в ресурса**
  - Еднакво за всички клиенти използващи услугата (ресурса)
  - Когато клиент промени състоянието на ресурс, останалите клиенти виждат промяната
- **Запазване на състоянието в клиента(пр., cookies)**
  - Специфично за клиента, трябва да се поддържа отделно от всеки
  - Примернп запазване на състоянието на сесията (login / logout)

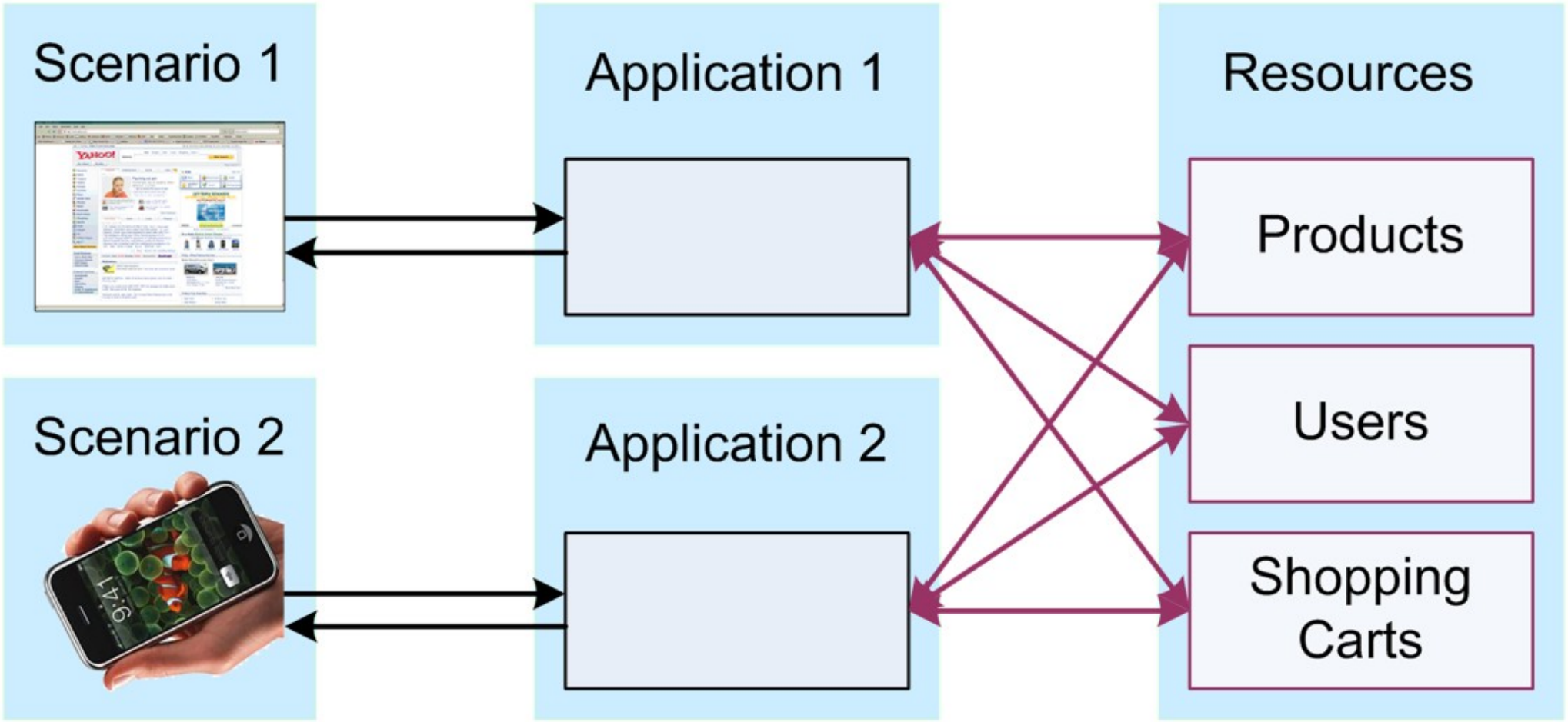
State in application



Statelessness – state in resources



Statelessness – state in clients



© Erik Wilde: http://dret.net/netdret/docs/rest-icwe2010/

RESTFUL SERVICE DESIGN

Методология на проектирането

- **Идентифициране и наименоване на ресурсите, които ще бъдат предоставяни от услугата**
  - пр. актьори и филми
- **Моделиране на връзките между ресурсите**
  - Един актьор може да играе в няколко филма
  - Няколко актьора играят в един и същ филм
- **Дефиниране на “приятни” URIs за адресиране на ресурсите**

Методология на проектирането

- **Съпоставяне на HTTP методите към ресурсите**
  - пр., GET movie, POST movie, и т.н.
- **Проектиране и документиране на представянето на ресурсите**
  - Искаме да предоставяме JSON (и XML)
  - JSON mime типа е application/json
- **Имплементиране и стартиране на Web услугата**
- **Тестване: с cURL или Browser developer tools**

Принципи при проектирането на REST API

- **Съществителните са ОК**
  - Прости и интуитивни базови URL
    - /actors
    - /peopleplayingin80iesmovies
  - 2 базови URL към ресурс
    - /actors (колекция)
    - /actors/1234 (елемент от колекцията)
  - Избягвайте глаголите
    - /getAllActors

HTTP методи

Resource	POST (create)	GET (read)	PUT (update)	DELETE (delete)
/actors	Create a new actor	List actors	Bulk update actors	Delete all actors
/actors/1234	Error	Show actor 1234	If exists update actor 1234 Else: error	Delete actor 1234

Множествено число и конкретни имена

- **Използването на множествено число е по интуитивно**
  - /movies
  - /actors
- **Но не е задължително, избягвайте обаче смесването им**
  - /movie /actor
  - /movies /actor
- **Препоръчително е използването на малък брой (12-24) конкретни имена вместо по-абстрактни**
  - /movie /actor /producer /cinema ...
  - /item

Опростяване на асоциациите

- **Взаимовръзките могат да бъдат сложни**
  - movie --> actor --> pets --> ...
  - URL адресите могат да станат дълги (дълбоки)
- **В повечето случаи URL нивата не трябва да са по дълги от : ресурс/идентификатор/ресурс**
  - /actor/1234/movies
  - /movies/1234/actors

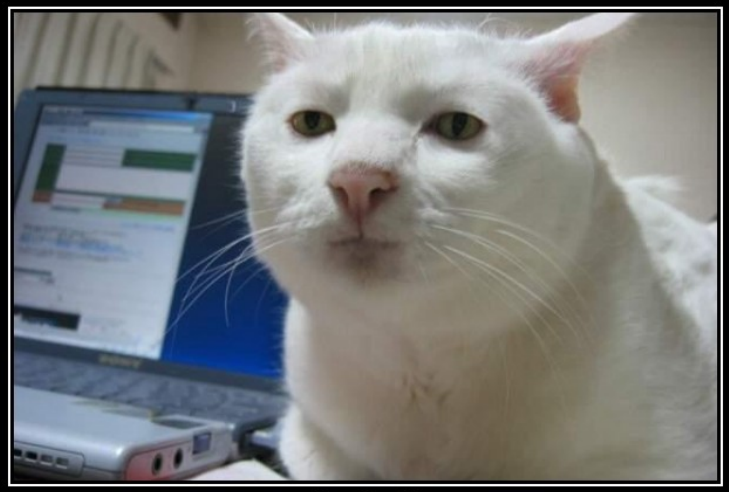
Филтриране

… скрийте сложността зад ?

/actors?gender=male&age=50

Работа с грешки

- **Използвайте HTTP кодовете**
  - Дефинирани са над 70; Повечето API интерфейси използват едва 8 - 10
- **Започнете с**
  - 200 OK (… всичко е наред)
  - 400 Bad Request (… грешка в клиента)
  - 500 Internal Server Error (… грешка в сървъра (т.е. API))
- **Ако имате нужда от повече, добавете ги**
  - 201 Created, 304 Not Modified, 401 Unauthorized, 403 Forbidden, и т.н.



200

OK

400

Bad Request

500

Internal Server Error

Learn to use HTTP Standard Status Codes

```
100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
```

4xx Client's fault

```
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
```

```
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
```

5xx Server's fault

Работа с грешки

- **Нека съобщенията върнати в тялото на HTTP да са колкото е възможно по многословни**

```
{ "developerMessage" : "Verbose, plain language
description of the problem for the app developer
with hints about how to fix it.”,

"userMessage":"Pass this message on to the app
user if needed.”,

"errorCode" : 12345,
"more info": http://example.com/errors/12345” }
```

Поддържане на версии

- **Никога не пускайте API без версия**
- **Препоръчителен синтаксис**
  - Включете номера на версията като първи елемент в URI
  - Префикс "v"
  - + цяло число указващо версията
  - /v1/actors
- **Поддържайте поне една версия назад**

Непълни отговори

- **Често не е необходимо да получите пълното представяне на ресурса**
- **Пестете пропускателната способност (throughput)**
- **Указвайте нужните полета**
  - пр. /movies?fields=title

Странициране

- **Почти винаги е лоша идея да върнете всички налични ресурси**
- **Използвайте странициране. Пример:**
  - /movies?limit=20&offset=0
- **Включете в отговора метаданни относно пълния брой на ресурсите**

Действия които не са свързани с ресурсите

- **Някои заявки към API интерфейса не изпращат и не получават ресурс**
  - calculate
  - translate
  - convert
- **Тук може да използвате глаголи, но ги опишете в документацията**
- **/convert?from=EUR&to=USD&amount=100**

Поддържане на повече формати

- **Препоръчва се вашата услуга да поддържа повече от един формат**
  - JSON обикновено по подразбиране; XML като втори
  - Кой от тях да използвате може да се автоматизира
- **"Чист" RESTful подход**
  - Accept: application/{xml | json} в HTTP хедърите
- **Чрез URI**
  - /actors.json, /actors.xml
  - /actors/1234.json, /actors/1234.xml
- **Смесен подход**
  - /actors --> content negotiated depending on Accept header
  - /actors.json --> direct format--specific access

Търсене

- **Глобално търсене (измежду ресурсите)**
  - /search?q=godfather
- **Търсене с обхват**
  - /actors/1234/movies?q=godfather
- **Форматирани резултати**
  - /search.xml?q=godfather