

# REST

Resources and  
caching

# Caching

Explanation

# Caching

- Es gibt Server und Clients
- Jedes mal, wenn ein Client etwas benötigt, setzt er einen Request an den Server ab
- Da kann schon was zusammen kommen

# Browser

- HTTP1.1 sagt: Jeder Browser darf nur maximal 2 Connections pro Webpage offen haben
- Chrome hat ein Limit von 6 pro remote-site und 10 über den gesamten Browser
- Wenn man da auf viele kleine Antworten wartet, dann kann das zu Verstopfungen führen

# Lösung

- Ich (der Client) hole mir nicht immer das allerneueste Objekt vom Backend, sondern gebe mich mit einem etwas älteren zufrieden. (-> weniger Requests)
- Der Server schickt nur ein neues Objekt, falls sich die Antwort seit der letzten Anfrage geändert hat (sonst nicht)
- Geht natürlich nur bei GET Requests

# Expires

# HTTP-Header

- Ich biete als Server eine Resource an
- Wenn ich eine Response schicke, hänge ich ein Header-Feld an

```
1 Expires: Sat, 13 May 2017 07:00:00 GMT
```

- Der Client weiß jetzt, dass er vor diesem Timestamp nicht nochmal fragen braucht
- Der Client gibt in diesem Fall dem Programmierer die 'alte' Antwort (passiert in Clients automatisch)
- Ansonsten holt sich der Client eine neue Resource ab (mit neuem expires-feld)

# Expires

- Eingestellt am Server
  - Muss sich mal wer überlegen
  - Kann sich auch ändern
- Passiert dann eigentlich im Client
- Ist für den Entwickler 'transparent'
- Der Entwickler macht einen Request, der Client merkt sich die Antworten per Resource (URL)



# Beispiel

## 1. Request:

### General

Request URL: <http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js>

Request Method: GET

Status Code:  200 OK

Remote Address: 216.58.211.202:80

### Response Headers

[view source](#)

Access-Control-Allow-Origin: \*

Age: 526383

Cache-Control: public, max-age=31536000, stale-while-revalidate=2592000

Content-Encoding: gzip

Content-Length: 33140

Content-Type: text/javascript; charset=UTF-8

Date: Mon, 08 Aug 2016 15:31:34 GMT

Expires: Tue, 08 Aug 2017 15:31:34 GMT


Last-Modified: Fri, 16 Oct 2015 18:27:31 GMT

## 2. Request:

### ▼ General

Request URL: <http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js>

Request Method: GET

Status Code:  200 OK (from cache)

Remote Address: 216.58.211.202:80

# E-Tag Hashing

# E-Tag

- Ich biete am Server eine Resource an
- Wenn ich eine Resource schicke, hänge ich einen Hash-Wert an, der eindeutig ist für diese 'Version' der Antwort
- Wenn ein Client die Resource anfragt, schickt er den Hash-Wert mit
- Am Server: Wenn der Hash-Wert der Antwort sich mit dem mitgeschickten Hash-Wert deckt, dann schicke ich einfach den HTTP-Status-Code:  
**304 - Not Modified**

# E-Tag

- Ich muss am Server trotzdem zuerst die Antwort zusammenbauen (Datenbankabfragen, etc...)
- Ich spare mir eigentlich nur das Schicken der Daten der Antwort (im Idealfall)
- Ich brauche keine Zusätzliche Einstellungen am Server (wie das Einschätzen der Expires-Time)

# Beispiel

## ▼ General

**Request URL:** `https://apis.google.com/js/api.js`

**Request Method:** GET

**Status Code:** 🟢 200

**Remote Address:** 216.58.211.206:443

**Referrer Policy:** origin

## ▼ Response Headers

**alt-svc:** quic=":443"; ma=2592000; v="39,38,37,35"

**cache-control:** private, max-age=1800, stale-while-reva

**content-encoding:** gzip

**content-security-policy:** script-src 'unsafe-inline' 'uns  
s://pagead2.googleadservices.com https://pagead2.goo  
s://www.youtube.com;report-uri /\_/cspreport/es\_oz\_20

**content-type:** application/javascript; charset=utf-8

**date:** Wed, 16 Aug 2017 15:14:06 GMT

**etag:** "7ad70e5091a6050e7b9ea82a20ed08f5"

# HTTP2

Multiplexing

# Multiplexing

- 'bündelt' viele kleine Requests in einem großen
- Die Antworten können in beliebiger Reihenfolge zurück kommen
- Erlaubt es dadurch eine Unmenge von Requests abzusetzen

# Multiplexing

- SSL (https) muss laufen (sonst kein HTTP2)
- SSL Offloading am Reverse Proxy ist aber auch kein Problem
  - Haben immer Gigabit Ethernet
  - Kaum Latenz
  - Können daher auch gerne über HTTP1.1 kommunizieren



# Resources

Granularity

What frontend developers want

# Frontend: I Want...

- **Möglichst wenig Requests absetzen**  
*Die sind immer ASYNC, also 'mehr Aufwand'. Außerdem möchte ich komplexe Objekte mit einem Aufruf bekommen, damit ich mir das zusammenbauen spare.*
- **Von diesen diesen großen Requests dann sehr viele und sehr schnell**  
*Wenn schon, denn schon. Ansonsten müsste ich in meiner App cachen.*

# Resources

Granularity

What backend developers want

# Backend: I Want...

- **Möglichst kleine Requests bekommen**

*Die kann man besser cachen.*

*Wenn der Frontend-DEV ein komplexes Objekt braucht, dann soll er sich das Stück für Stück abholen über die API.*

- **Von den kleinen Requests dann aber gerne auch viele**

*Dafür müssen wir die Server eh auslegen.*

Das passiert ja auch wenn mehr Clients unsere API nutzen.

# Resources

Granularity

Reality kicks in...

# Reality

- **Immer eher kleine Ressourcen bauen! (feingranular)**  
*Nicht alles zu liefern versuchen, was die GUI gerade in diesem Moment braucht und das auch noch auf einmal.*
- **Jemand anderes braucht wieder was anderes**

Ihr schreibt immer wieder neuen Backend Code wenn ein anderer Programmierer kommt und eure API ein wenig anders verwendet.

Ihr häuft Backend Code an, der teuer ist (Wartung).

Ihr transferiert damit Logik vom Frontend ins Backend.  
(Kann in Ausnahmefällen gut sein -> z.B.: GraphData)

# Reality

- **Immer eher kleine Ressourcen bauen! (feingranular)**  
*Nicht alles zu liefern versuchen, was die GUI gerade in diesem Moment braucht und das auch noch auf einmal.*
- **Caching geht sonst nicht mehr**  
Die Resource kann auch nicht mehr mit expires optimal eingestellt werden

Die Aktualität von Teilen des riesigen Objektes ist unterschiedlich wichtig.

Dadurch muss ich mich quasi am wichtigsten Teil orientieren und expires auf ein Minimum einstellen, was es quasi fast nutzlos machen wird.

# Reality

- **Immer eher kleine Ressourcen bauen! (feingranular)**  
*Nicht alles zu liefern versuchen, was die GUI gerade in diesem Moment braucht und das auch noch auf einmal.*
- **Caching geht sonst nicht mehr**  
Gelinkte Daten werden dann auch immer mitgeschickt, wenn sich was ändert (E-Tag caching ist dann kaputt)

Wenn ich so ein riesiges Objekt habe, dann ändert sich auch immer irgendwas da drinnen.

E-Tag ist damit quasi nutzlos.



# Reality

- **Immer eher kleine Ressourcen bauen! (feingranular)**  
*Nicht alles zu liefern versuchen, was die GUI gerade in diesem Moment braucht und das auch noch auf einmal.*
- **Manipulation werden schwieriger**

Wir schleifen ja alles 1:1 an die Datenbank durch.

Damit haben wir am wenigsten Arbeit...

**Löschen geht dann nicht mehr**

Was soll ich löschen? Alle verbundenen Objekte? Eigene Abfragesprache für Löschen?

**PATCH wird auch immer komplizierter / unbenutzbar**

# Reality

- **Immer eher kleine Ressourcen bauen! (feingranular)**  
*Nicht alles zu liefern versuchen, was die GUI gerade in diesem Moment braucht und das auch noch auf einmal.*

**Lieber 100 mal zum Backend gehen und das komplexe Objekt im Client aufbauen!**

- Caching geht
- Manipulationen funktionieren
- Code ist wartbarer
- HTTP2 macht das möglich



# Referenzen

- <https://www.cloudbees.com/blog/an-overview-of-caching-methods>