Online Multimedia

Winter Semester 2019/20

Tutorial 09 – REST APIs

REST APIs

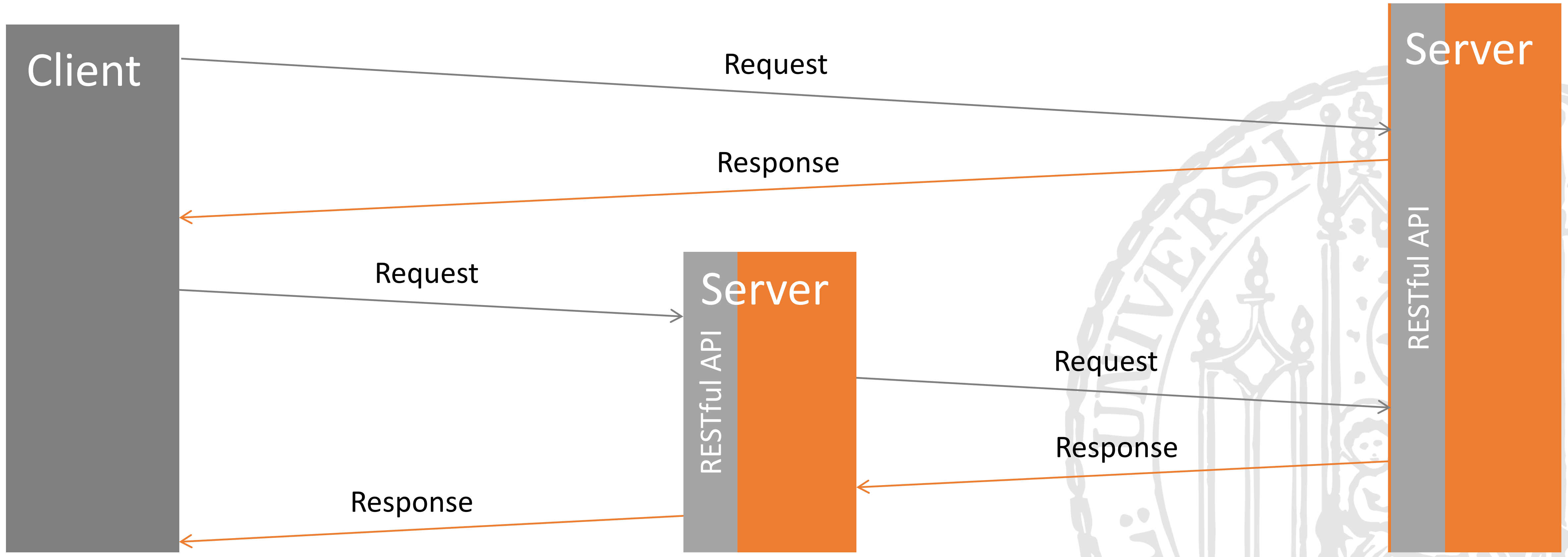
REpresentational **S**tate **T**ransfer – REST

- REST is a general architecture for distributed systems, prominently (mis)used on the Web
- Defined by a number of architectural constraints:
    - *Client-Server* – Separation of concerns; given on the Web
    - *Stateless* – Each request must contain all the context necessary to respond
    - *Cache* – Responses must be cacheable and indicate whether that is a good idea
    - *Uniform Interface* –
    - *Layered System* – Servers can be clients for other services → Machine-to-machine communication
    - *Code-On-Demand* – Clients can request code to extend their functionality. Rarely used to prevent XSS

- A REST API consists of resources and actions on those resources

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST – Architecture



REST – Key Concepts

"The key abstraction of information in REST is a *resource*."

- Roy T. Fielding

Name	Examples	Where to find it in HTTP
Resource	Application-specific, e.g. *user*, *artist*, *album*	*Abstract concept*
Resource Identifier	URI/URL	URL
Resource Metadata	Resource-Media-Type	Header
Representation	JSON, XML, HTML, Images	Body
Representation Metadata	Representation-Media-type, last-modified, etc.	Header
Resource Methods	GET, POST, PUT, …	HTTP Method

REST – Resource Identifiers

- Resources are identified by URI/URLs
- Since resources are *"things"*, resource identifiers primarily use *nouns*
  Good: `/users`
  Bad: `/select-users`
- URIs usually follow a hierarchical scheme:
  `/projects/536/members/4`
- By convention (and Googles recommendation) use hyphens (-) in URIs:
  Good: `/date-of-birth`
  Bad: `/date_of_birth` or `/dateofbirth`
- Use Query-Parameters to filter/sort a collection of resources
  `/users?born-before=1990&sort-by=name`

Resource Representation & Metadata

- Resource representation must be hypermedia/hypertext
  - "Hypertext (means) the **simultaneous presentation of information and controls"** – Roy T. Fielding
  - E.g. in HTML the <a>-tag both presents data and can be used to GET another resource
- A *true* RESTful API should be self-descriptive. Hypertext-references therefore should be described with their media-type or relationship
- This hypertext & metadata should be the primary definition of what is possible with a resource (*HATEOAS* – **H**ypertext **A**s **T**he **E**ngine **O**f **A**pplication **S**tate)
- This can mean new media-types in conjunction with common ones
  `application/hal+json` *(HAL = Hypermedia API Language)*

http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

Resource representation – JSON

```
GET /projects/10

{
    "departmentId": 10,
    "departmentName": "Administration",
    "locationId": 1700,
    "managerId": 200,
    "links": [
        {
            "href": "/projects/10/employees",
            "rel": "employees",
            "type": "application/myapp.list+json"
        }
    ]
}
```

REST – Resource Methods

- Resource Methods describe *what* should be done with resources
- The table on the right describes the common convention on the web

HTTP Method	CRUD
POST	Create
GET	Read
PUT	(Full) Update
DELETE	Delete
PATCH	(Partial) Update

Idempotent and Safe Methods

- Certain HTTP methods are expected to be idempotent and/or safe
  - Safe methods do not change data
  - Idempotence means you can perform an operation multiple times and the result will always be the same.
  - i.e. You can call the same URL multiple times in a row and the result is always the same – as long no other URLs are called in between

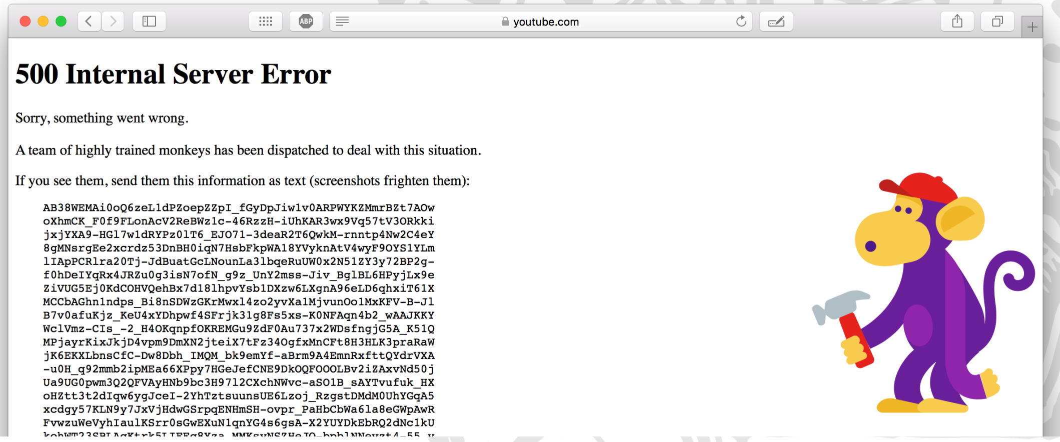
Method	Idempotent	Safe
`GET`	Yes	Yes
`POST`	No	No
`PUT`	Yes	No
`DELETE`	Yes	No
`PATCH`	No	No
`HEAD`	Yes	Yes
`OPTIONS`	Yes	Yes

Status Codes

- To ensure self-descriptiveness, servers should respond with the appropriate HTTP status code

1xx – Informational
2xx – Success
3xx – Redirects
4xx – Client Error
5xx – Server Error

https://cdn.shopify.com/s/files/1/0322/6897/files/404-permalink_1170x1170.png?432866230176278629



500 Internal Server Error

Sorry, something went wrong.

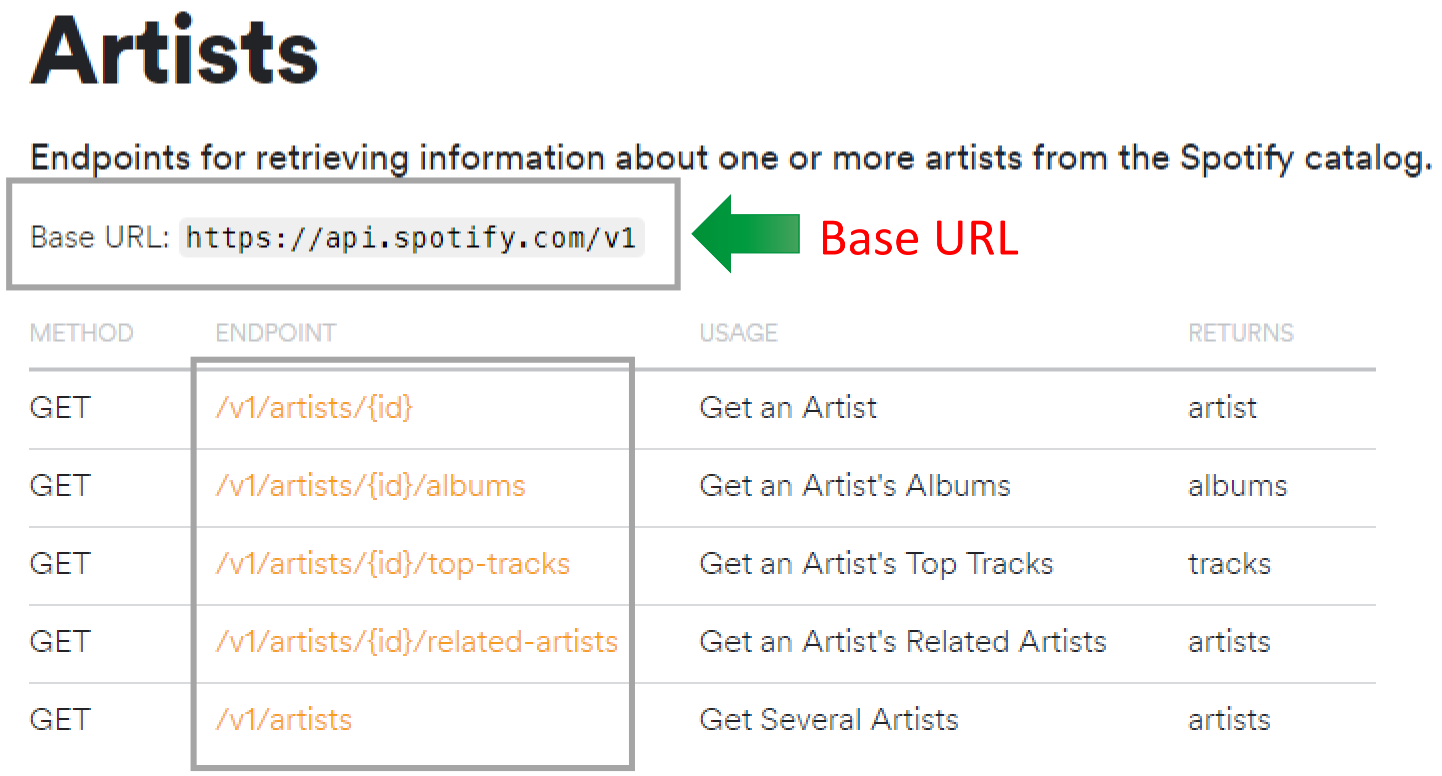
A team of highly trained monkeys has been dispatched to deal with this situation.

If you see them, send them this information as text (screenshots frighten them):

```
AB38WEMAi0oQ6zeL1dPZoepZZpI_fGyDpJiw1v0ARPWYKZMmrBZt7AOw
oXhmCK_F0f9FLonAcV2ReBWz1c-46RzzH-iUhKAR3wx9Vq57tV3ORkki
jxjYXA9-HGl7w1dRYPz0lT6_EJO71-3deaR2T6QwkM-rnntp4Nw2C4eY
8gMNsrgEe2xcrdz53DnBH0iqN7HsbFkpWA18YVyknAtV4wyF9OYS1YLm
lIApPCRlra20Tj-JdBuatGcLNounLa3lbqeRuUW0x2N51ZY3y72BP2g-
f0hDeIYqRx4JRZu0g3isN7ofN_g9z_UnY2mss-Jiv_BglBL6HPyjLx9e
ZiVUG5Ej0KdCOHVQehBx7d18lhpvYsb1DXzw6LXgnA96eLD6qhxiT61X
MCCbAGhn1ndps_Bi8nSDWzGKrMwxl4zo2yvXa1MjvunOo1MxKFV-B-Jl
B7v0afuKjz_KeU4xYDhpwf4SFrjk31g8Fs5xs-K0NFAqn4b2_wAAJKKY
WclVmz-CIs_-2_H4OKqnpfOKREMGu9ZdF0Au737x2WDsfngjG5A_K51Q
MPjayrKixJkjD4vpm9DmXN2jteiX7tFz34OgfxMnCFt8H3HLK3praRaW
jK6EKXLbnsCfC-Dw8Dbh_IMQM_bk9emYf-aBrm9A4EmnRxfttQYdrVXA
-u0H_q92mmb2ipMEa66XPpy7HGeJefCNE9DkOQFOOOLBv2iZAxvNd50j
Ua9UG0pwm3Q2QFVAyHNb9bc3H97l2CXchNWvc-aSO1B_sAYTvufuk_HX
oHZtt3t2dIqw6ygJceI-2YhTztsuunsUE6Lzoj_RzgstDMdM0UhYGqA5
xcdgy57KLN9y7JxVjHdwGSrpqENHmSH-ovpr_PaHbCbWa6la8eGWpAwR
FvwzuWeVyhIaulKSrr0sGwEXuN1qnYG4s6gsA-X2YUYDkEbRQ2dNc1kU
```

https://proxy.duckduckgo.com/iu/?u=https%3A%2F%2Fi.redd.it%2Fu2uity1r99hy.png&f=1

Example: Spotify API



Artists

Endpoints for retrieving information about one or more artists from the Spotify catalog.

Base URL: `https://api.spotify.com/v1`

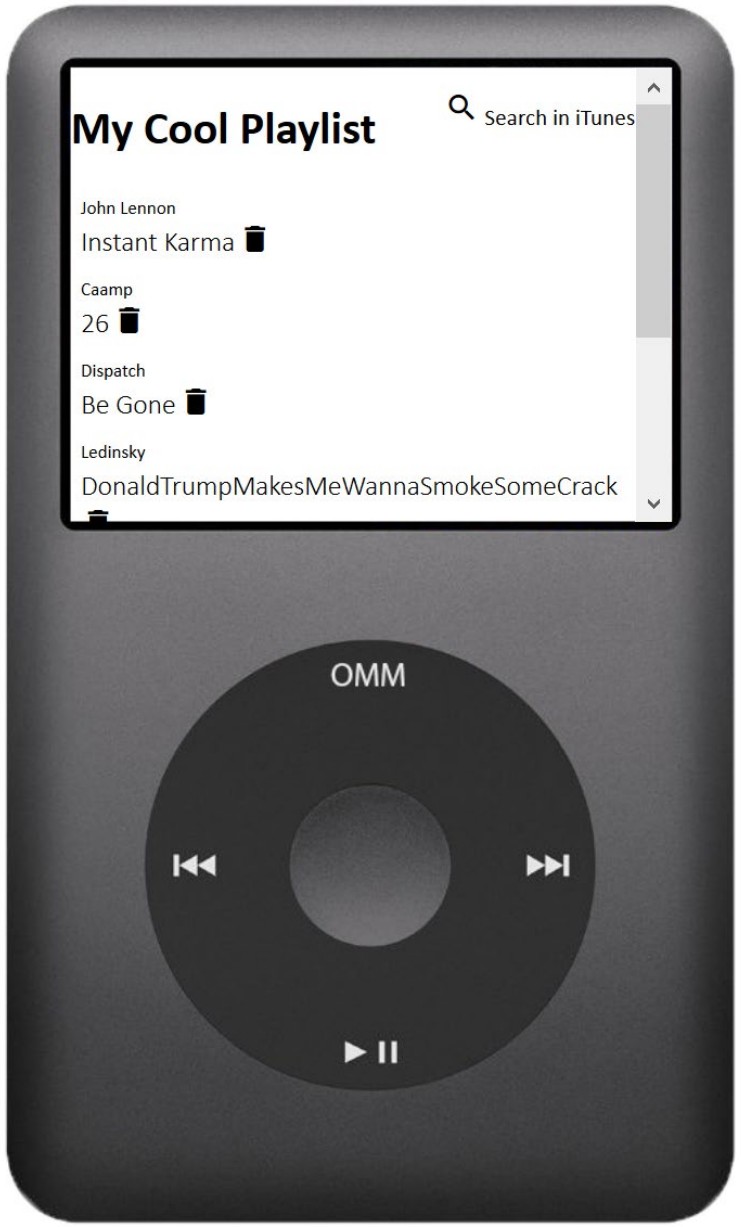
METHOD	ENDPOINT	USAGE	RETURNS
GET	/v1/artists/{id}	Get an Artist	artist
GET	/v1/artists/{id}/albums	Get an Artist's Albums	albums
GET	/v1/artists/{id}/top-tracks	Get an Artist's Top Tracks	tracks
GET	/v1/artists/{id}/related-artists	Get an Artist's Related Artists	artists
GET	/v1/artists	Get Several Artists	artists

Base URL

Endpoints

https://api.spotify.com/v1/artists/1
vCWHaC5f2uS3yhpwWbIA6/albums
?market=ES
&include_groups=album
&limit=2

Code Along: The iPod API



Code Along: The iPod API

Imagine a „super-thin-client“ iPod

- Nearly no local memory
- Requests data for each screen from a server
- Transfer only the data necessary at a point of time

The iPod API: Requirements

The API should be able to…

- Handle multiple playlists
- Search the iTunes API for tracks
- Add songs to a playlist
- Remove songs from a playlist

Brainstorming Session

What would be a suitable API design?

METHOD	ENDPOINT	USAGE	RETURNS

The iPod API: Our Suggestion

METHOD	ENDPOINT	USAGE	RETURNS
GET	/playlists	Get all playlists	List of ids
GET	/playlists/{id}	Get a playlist	Playlist metadata (name, id)
GET	/playlists/{id}/songs	Get all playlist‘s song	List of track ids
GET	/playlists/{id}/songs/{trackId}	Get a song	Redirect to /songs/{trackId}
POST		Add a song to the playlist	-
PUT		Add a song to the playlist	-
DELETE		Remove a song from the playlist	-
GET	/songs/{trackId}	Get a song	Song metadata (title, artist, id)
GET	/search?term={term}	Search for a song	Metadata for all found songs

Breakout #1

Use the skeleton *ipod-server* from GitHub

- „virtual iPod“ as webpage
- NodeJS Express server, that manages playlists data

Open http://localhost:3000 to view the iPod in your browser

Implement the *playlists* and *songs* endpoints of the API! No changes have to be done on the clientside. Do not implement *search* yet.

- The data is managed by the *DataStorage* class. One endpoint is already implemented for you, to see how *DataStorage* can be accessed

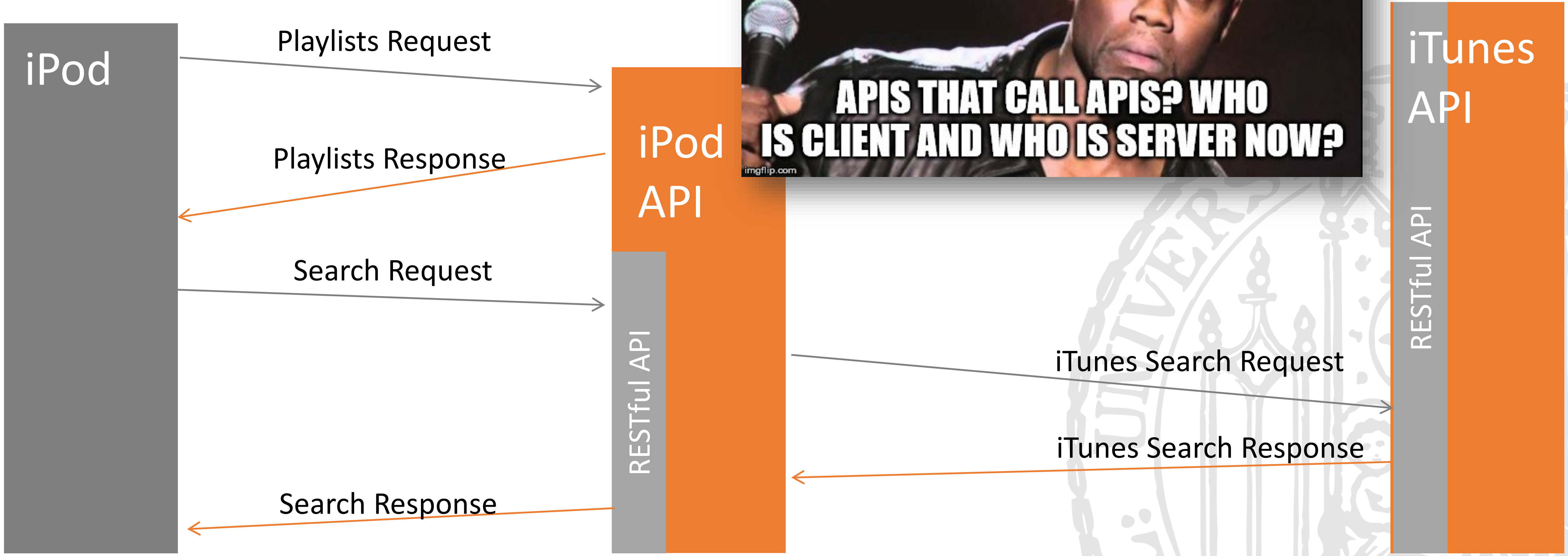
iTunes Search API

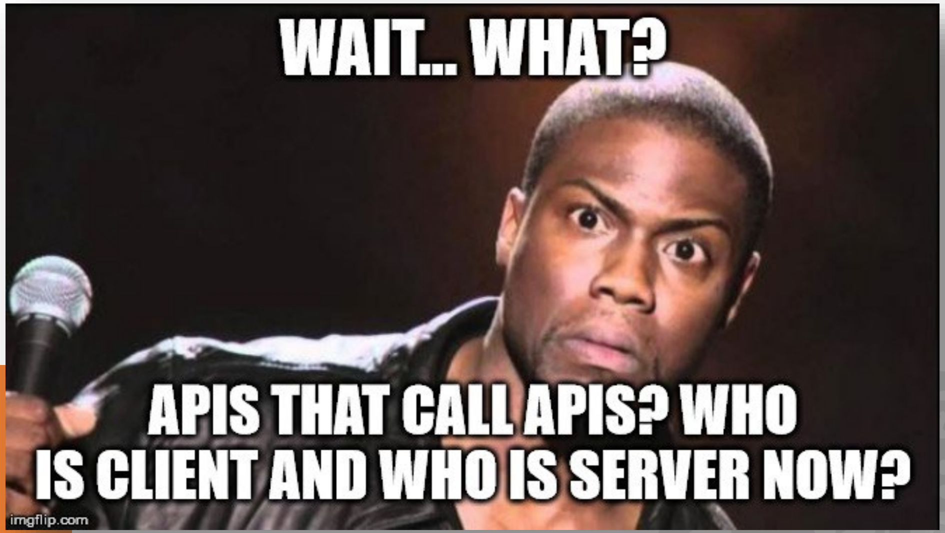
- Search in the iTunes database
- Songs, books, artists, podcasts, videos, …
- Read only – no authentication required

Lets use it to feed our search endpoint

https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/

Search Endpoint





Breakout #2

Implement the search endpoint

http://localhost:3000/search?term=Californication

Under the hood, the search should be performed by the iTunes API

Hint: To avoid complications later on, cache each found song in the *DataStorage*'s *songs* array immediately.

Round-up Quiz

1. What is special about an idempotent endpoint?
2. Which groups of HTTP status codes do you know?
3. What is the key abstraction of REST?
4. What are query parameters appropriate for?

Happy Holidays!
See you next year!