Online Multimedia

Winter Semester 2019/20

Tutorial 02 – AJAX

Today's Agenda

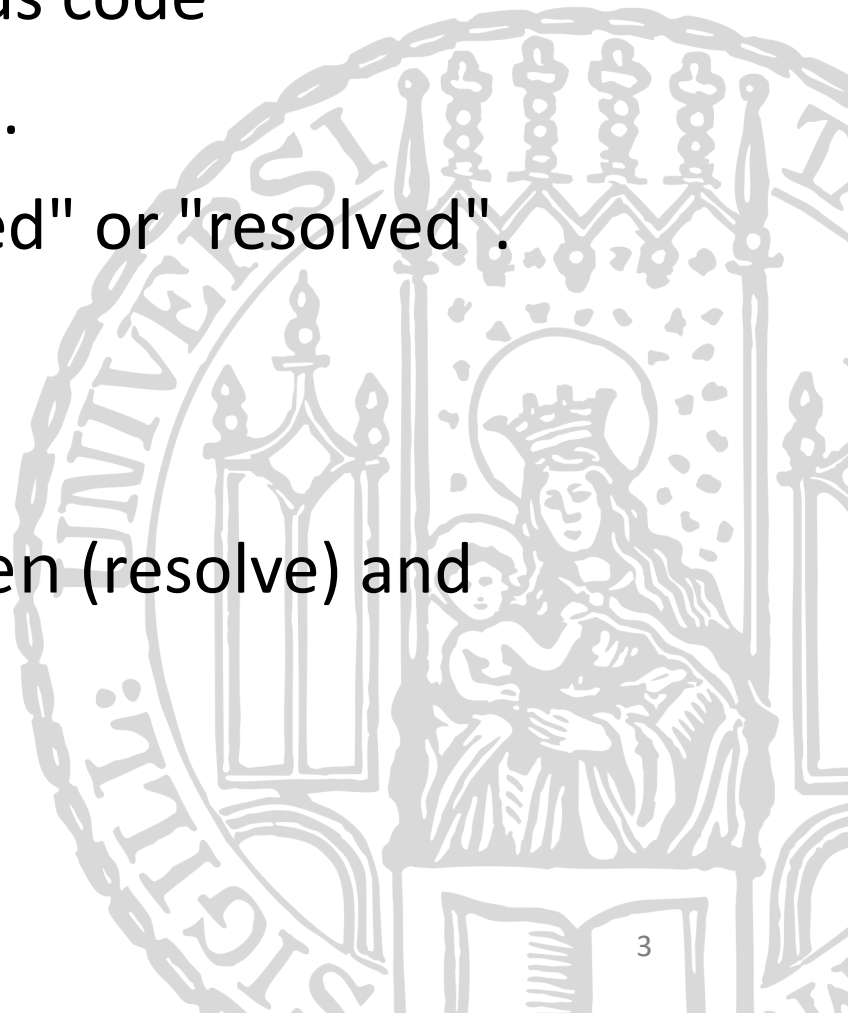
- AJAX
- Web APIs
- Same Origin Policy
- Roundup Quiz

Recap

Promises

- Promises are a concise way to write asynchronous code
- Inside a promise, asynchronous code is executed.
- When the code completes the promise is "fulfilled" or "resolved".
- When the code fails the promise is "rejected".

- The result is used by callbacks attached via `.then` (resolve) and `.catch` (reject)
- Actions on promises can be chained



Recap

async & await

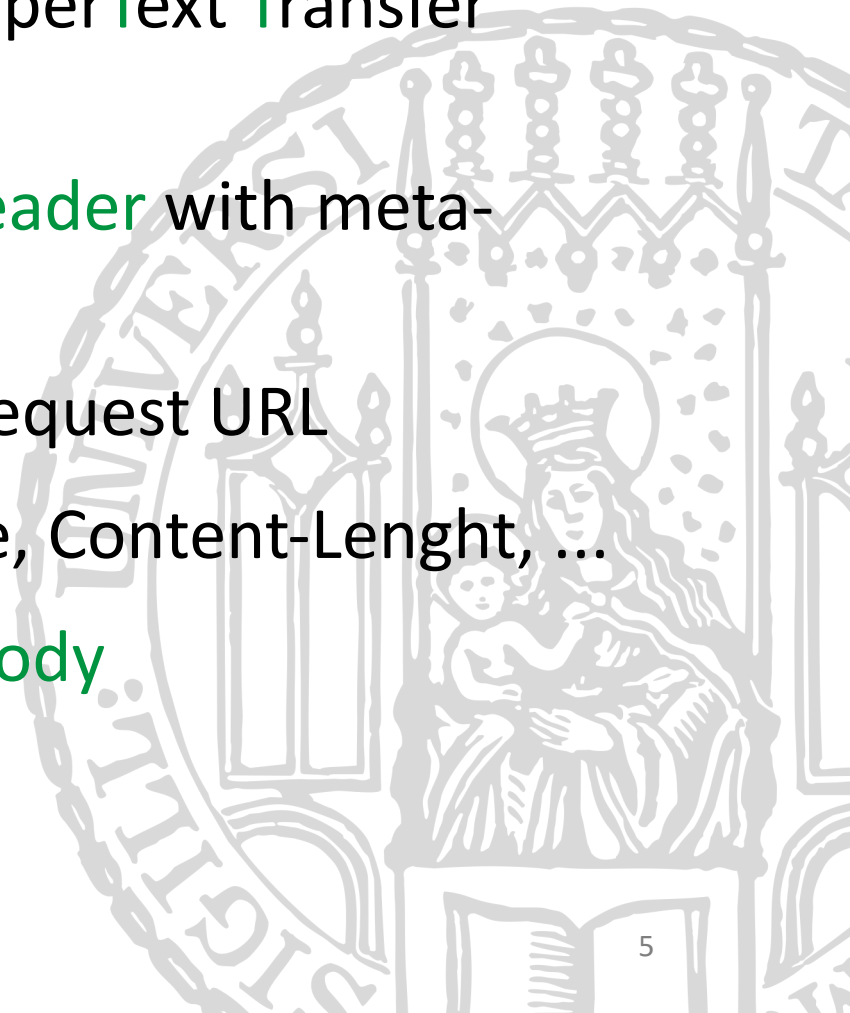
```
function () {
    return new Promise((resolve, reject) => {
        db.getByName(…)
            .then((id) => db.getById(id)
            .then((result) => resolve(result))
        )
    });
}

// The same function using async and await
async function () {
    const id = await db.getByName(…);
    const result = await db.getById(id);
    return result;
}
```

Recap

HTTP

- Data on the Web is mostly transferred via the HyperText Transfer Protocol
- A HTTP Request consists of a request line, the header with meta-information and the body with the playload
- The request line has the HTTP method and the request URL
- The header contains things like the Content-Type, Content-Lenght, ...
- The response has a status code, headers and a body



Recap

HTTP

HTTP Status Codes

- 1xx Informational
- 2xx Success
- 3xx Redirect
- 4xx Client Error
- 5xx Server Error

HTTP Methods

Method	Effect
GET	Read
POST	Create
PUT	Full Update
PATCH	Partial Update
DELETE	Delete
HEAD	Just the header please
OPTIONS	Pre-Flight Requests

AJAX

- **A**synchronous **Ja**vaScript **+ X**ML (Ajax is meant as “shorthand” and not as acronym. But it’s a good mnemonic device...)
- Allows passing around data between client- and server-side applications back and forth – **without refreshing the page**
- AJAX requests (also: **XHR** = XMLHttpRequest):
  - GET: retrieve data – no manipulation on the server
  - POST: retrieve and/or modify data
  - and some more
- “XML” used to be the predominant format, nowadays it is more common to use JSON.

XML? JSON!

- Before JSON was widely adopted, XML was the state of the art
- JavaScript Object Notation
- Easily human-readable format for data exchange
- Based on key-value pairs

```
{
  "firstName": "John",
  "lastName": "Doe",
  "phone": [
    {
      "type": "Home",
      "number": "5648978965"
    },
    {
      "type": "Mobile",
      "number": "6458979878"
    }
  ]
}
```

Quick Break Out

- What are the advantages of using JSON?
- What are the advantages of using XML?

FYI: Testing AJAX with httpbin.org

- Allows testing different settings
- Response format: JSON

httpbin(1): HTTP Request & Response Service

Freely hosted in HTTP, HTTPS & EU flavors by Runscope

ENDPOINTS

/ This page.
/ip Returns Origin IP.
/user-agent Returns user-agent.
/headers Returns header dict.
/get Returns GET data.
/post Returns POST data.
/patch Returns PATCH data.
/put Returns PUT data.
/delete Returns DELETE data
/encoding/utf8 Returns page containing UTF-8 data.
/gzip Returns gzip-encoded data.
/deflate Returns deflate-encoded data.
/status/:code Returns given HTTP Status code.
/response-headers?key=val Returns given response headers.
/redirect/:n 302 Redirects *n* times.

XMLHttpRequest

- API to transfer data between client and server without a full page refresh
- Originally designed by Microsoft, adopted by all other browsers
- Ready States 0, 1, 2, 3, 4
  - Mostly relevant: **4** (done)
  - See lecture slides
- HTTP Status Codes are accessible in XMLHttpRequest.status
  - 200: Success
  - 401: unauthorized
  - 404: not found
  - 500: internal server error

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

XHR

```
const xhr = new XMLHttpRequest();
xhr.open('POST', url);
xhr.onreadystatechange = () => {
  if (xhr.readyState === 4) {
    if (xhr.status === 200) {
      const results =
        JSON.parse(xhr.responseText);
      handleResults(results);
    }
  }
}
xhr.send(JSON.stringify(data));
```

fetch

- Since XHR is arduous, there is a „new“ API: fetch
- It greatly simplifies AJAX by using Promises

```
fetch('https://httpbin.org/post‘,
      { method: 'POST', ... })
  .then((res) => res.json())
  .then((res) => console.info(res.url))
  .catch((err) => console.error(err));
```

XHR vs. fetch

```
const xhr = new XMLHttpRequest();
xhr.open('POST', url);
xhr.onreadystatechange = () => {
  if (xhr.readyState === 4) {
    if (xhr.status === 200) {
      const results =
       JSON.parse(xhr.responseText);
      handleResults(results);
    }
  }
}
xhr.send(JSON.stringify(data));
```

```
fetch(url, {
  method: 'POST',
  body: data,
})
  .then((res) => res.json())
  .then(handleResults);
```

XHR vs. fetch

XHR	fetch
Event-callback based	Promise based
1. Create XHR	N/A
2. Open with URL and method	Same Request options can be passed easier
3. Attach readystatechange	Done via `.then`/`.catch` Multiple actions can be chained
4. Send (with payload)	N/A

The fetch-API simplifies AJAX and provides a modern interface with Promises.

Breakout #01

- Use the file breakout-skeleton/breakout-01.html
- We want a re-usable function `postJSON(...)` to perform AJAX POST requests.
- Complete the function, take a look at the previous slide.
- Timeframe: **8 Minutes**

Web APIs

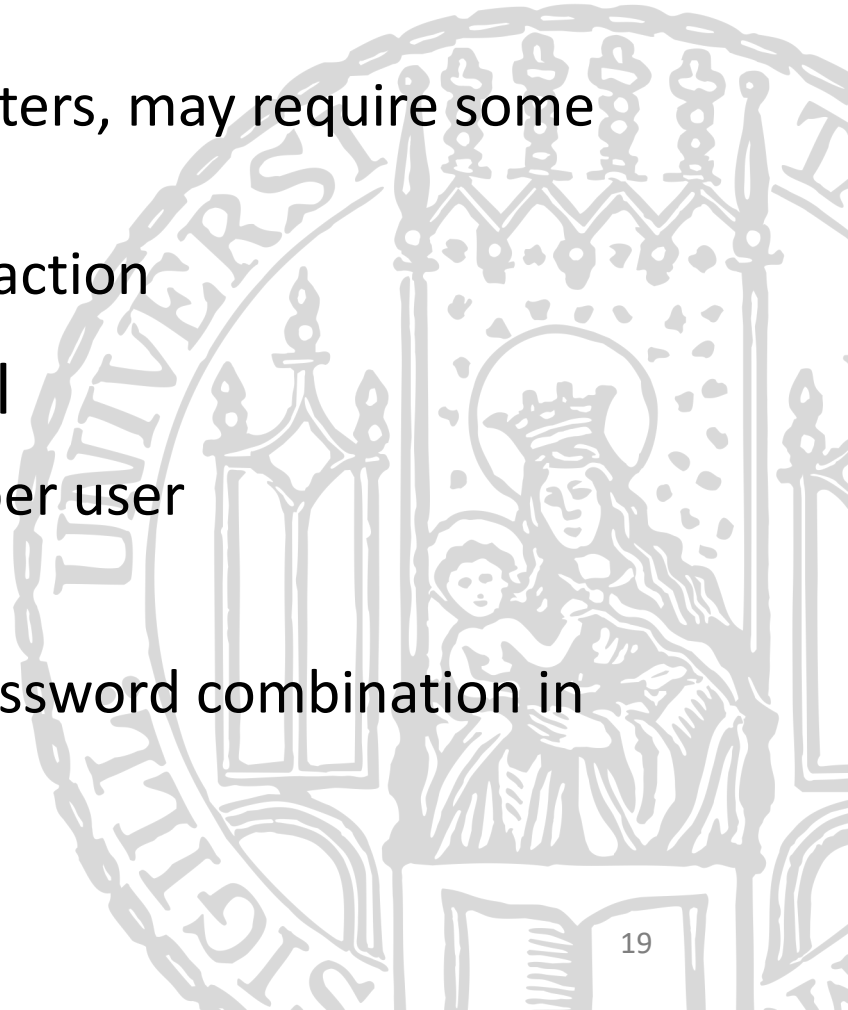
- **A**pplication **P**rogramming **I**nterface

- Simply speaking:
  APIs allow your software to use parts of other software.

- Web APIs often come as REST APIs
  - Representational State Transfer
  - Allow maintaining remote object states through an API

A Web API…

- Offers a set of **endpoints**
  - Can be accessed via **HTTP request**: url, some parameters, may require some headers, …
  - Either return data in the response, or perform some action
- Usually is secured by some kind of access control
  - **Rate limit** to cap the maximum amount of requests per user
  - Define the access **scope** of each user
  - Is implemented by an **access token** or username / password combination in the request‘s header or parameters



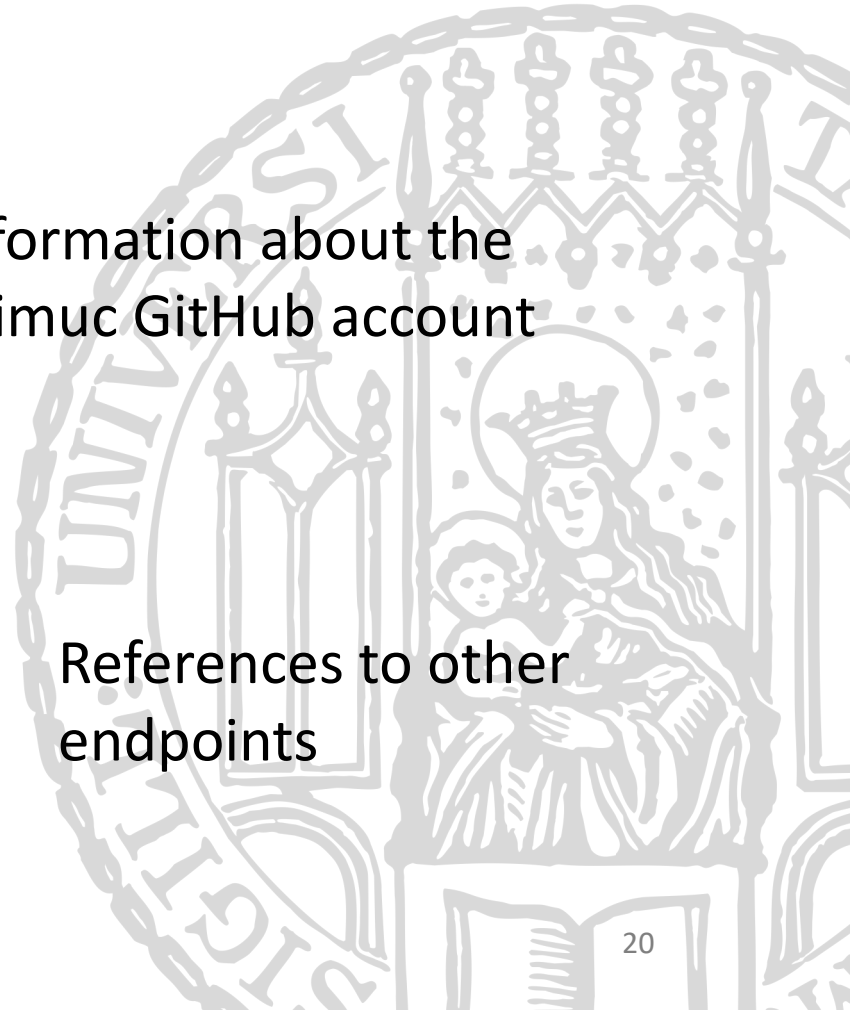
The GitHub API v3 – Retrieve Information

GET https://api.github.com/users/mimuc

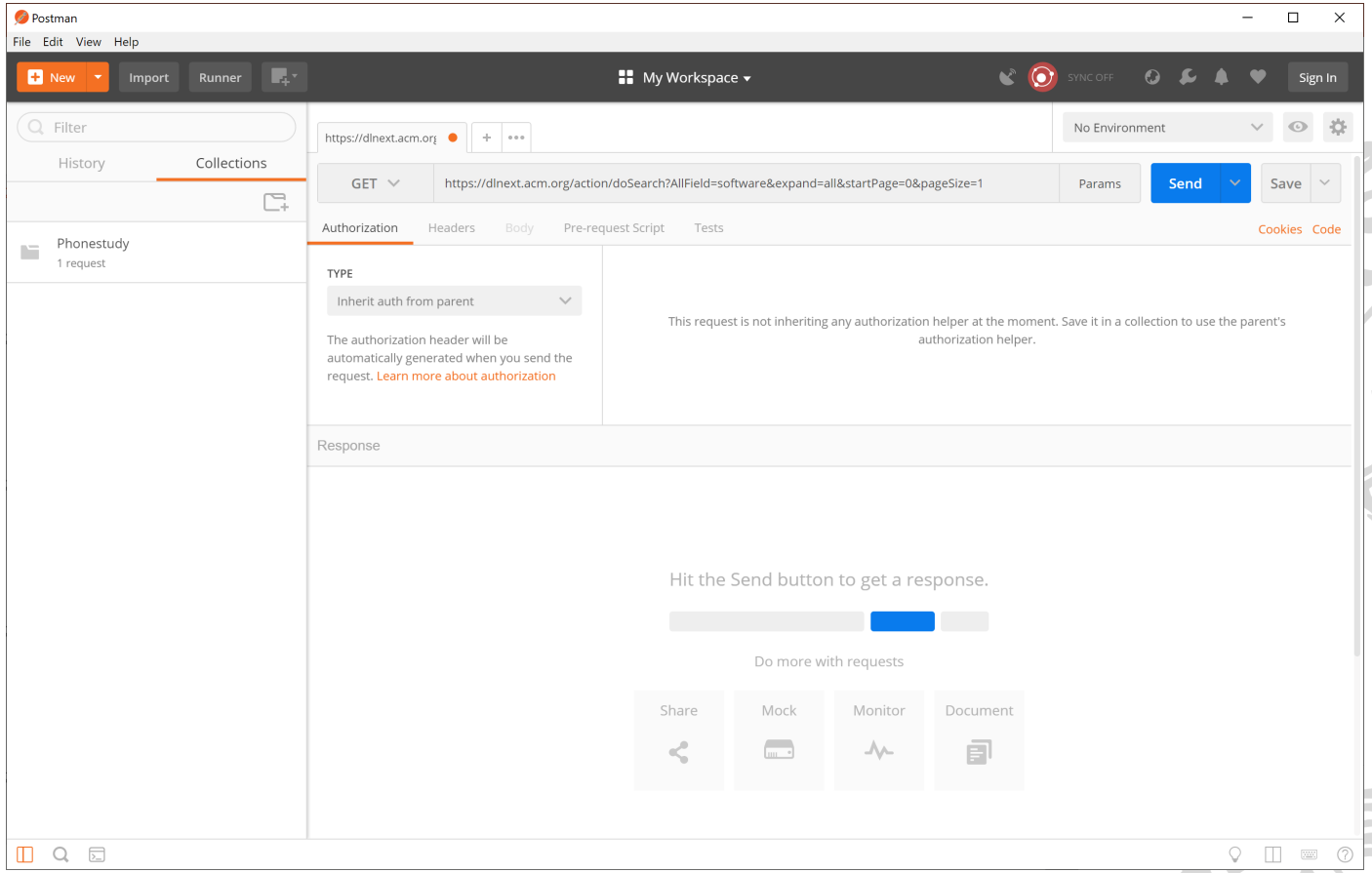

```
{
"id": 22574607,
"avatar_url": "https://avatars3.githubusercontent.com/u/22574607?v=4",
"type": "Organization",
"name": "MIMUC",
"blog": "https://www.medien.ifi.lmu.de",
"bio": "Media Informatics Group @ LMU Munich",
"public_repos": 12,
"followers": 0,
"following": 0,
"url": "https://api.github.com/users/mimuc",
"followers_url": "https://api.github.com/users/mimuc/followers",
"following_url": "https://api.github.com/users/mimuc/following{/other_user}",
"starred_url": "https://api.github.com/users/mimuc/starred{/owner}{/repo}",
"subscriptions_url": "https://api.github.com/users/mimuc/subscriptions",
"organizations_url": "https://api.github.com/users/mimuc/orgs",
"repos_url": "https://api.github.com/users/mimuc/repos",
}
```


Information about the Mimuc GitHub account

References to other endpoints



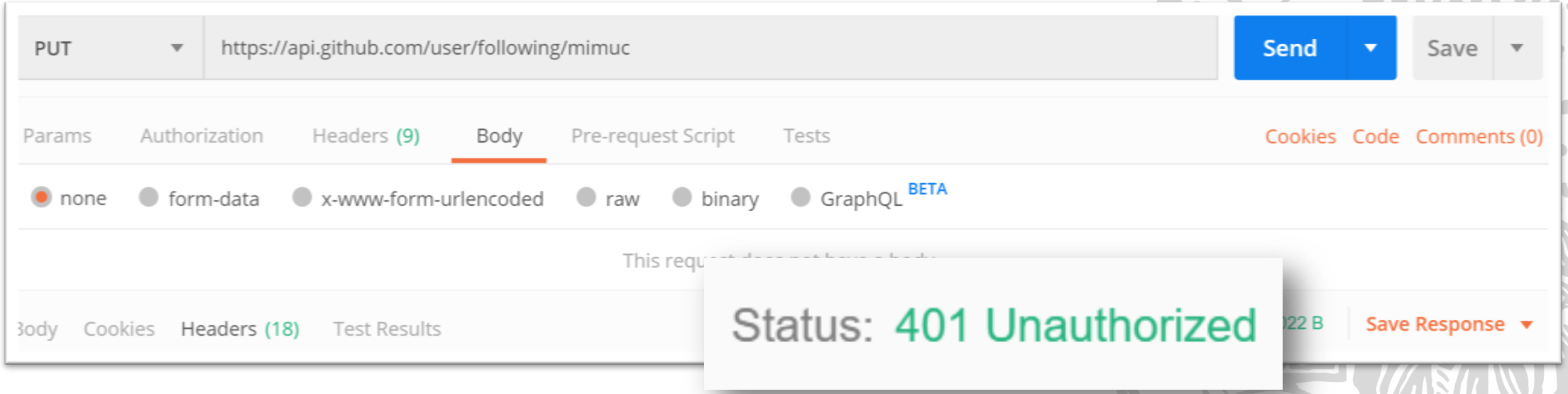
FYI: Testing APIs with Postman



The GitHub API v3 – Perform an action

PUT https://api.github.com/user/following/mimuc

„Add *mimuc* to the accounts I am following“



Authentication vs. Authorization

- For many actions an API must know who you are and whether you are allowed to do what you want to do.
- Note the difference:
  Authentication: „Who am I?“
  Authorization: „Am I allowed to do what I want to do?“

Authentication

The Quick-and-Dirty-way: **Basic Authentication**

- Pass username and password with the request.

  \+ quick and easy - security issue

Better: **Personal Access Token** 

\+ Security Improvement: token can be revoked in case it gets stolen

Like-a-pro: **OAuth Token** 

\+ Even more security: token has a lifespan and is exchanged regularly

\- More complicated implementation on the client side

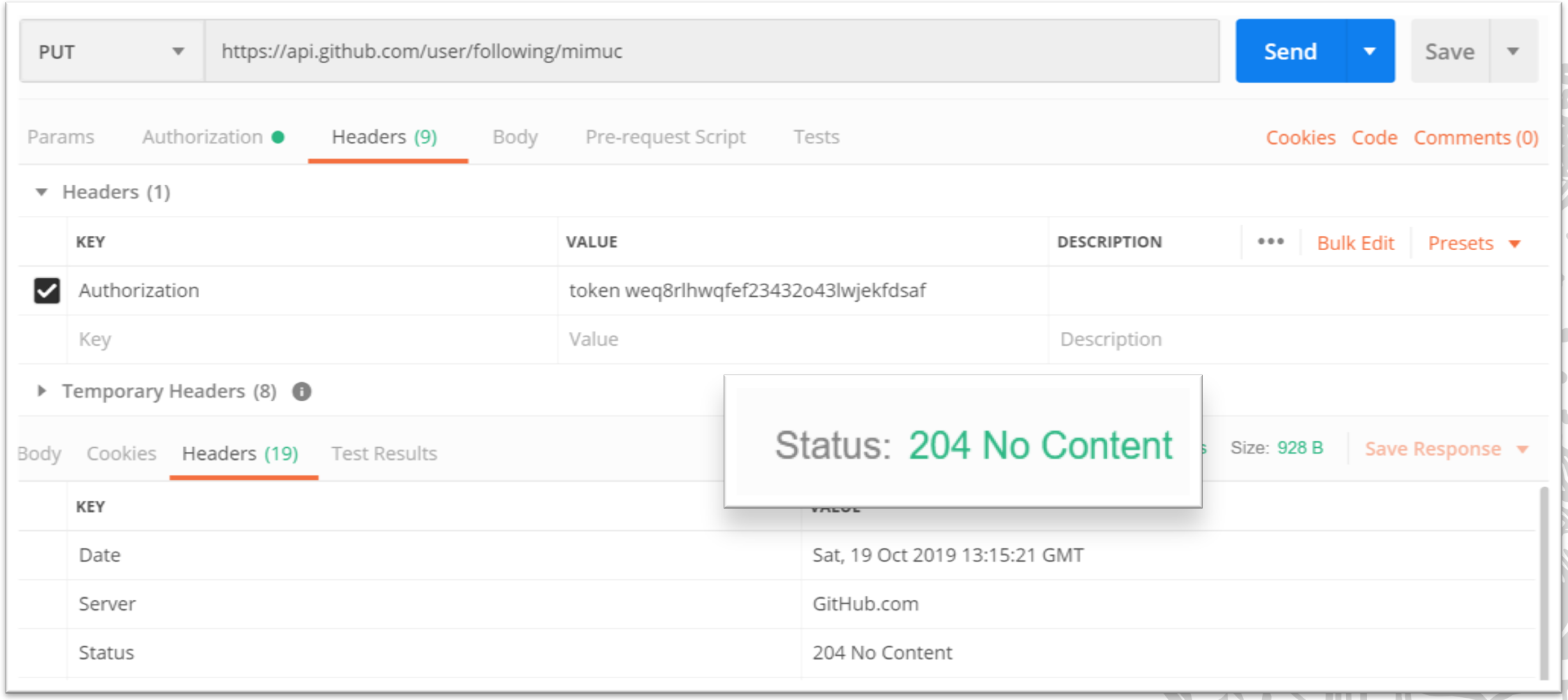
Authentication: Personal GitHub Access Token

- Create a personal access token in the GitHub web interface: https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line

Add access token to request header

```
{ ...
  headers: {
    'Authorization': `Token dasd8f9re92h90v25`,
  }
... }
```

Second Try – Now with Authentication



Breakout #02

- First: Create a GitHub account + access token
- Use the file breakout-skeleton/breakout-02.html. It currently displays information about the specified GitHub user

- Task 1: Change the request, to display all list of who the entered user is following
- Task 2: Create a text field where one can enter a user that should be unfollowed

- Timeframe: **15 Minutes**

A Frequent AJAX Problem

Same Origin Policy (SOP) – Part 1

- Example: your script runs is executed at http://example.com

- Same origin:
  - http://example.com
  - http://example.com**/**
  - http://example.com**/another/page**

- Different origin:
  - http://**www.**example.com
  - http://example.com**:3830**/
  - http**s**://example.com
  - http://example**.org**

Same Origin Policy (SOP) – Part 2

- SOP tries to hamper **cross site scripting**

- SOP would block AJAX request to foreign domains entirely ☹
- You can allow cross-origin requests **on the server side**
- Solutions:
  - Cross-Origin Resource Sharing (CORS)
  - JSON with Padding (JSONP)
  - Apache Reverse Proxies

Roundup Quiz

1. What does AJAX stand for?
2. What does “asynchronous” actually mean?
3. What is the difference between GET and POST?
4. Name the steps to perform an AJAX request using the XMLHttpRequest API.
5. What is basic authentication and what is its main drawback?
6. What is an advantage of JSON compared to XML?
7. How do you avoid SOP issues?