Verteilte Systeme 1

Technologien des World Wide Web

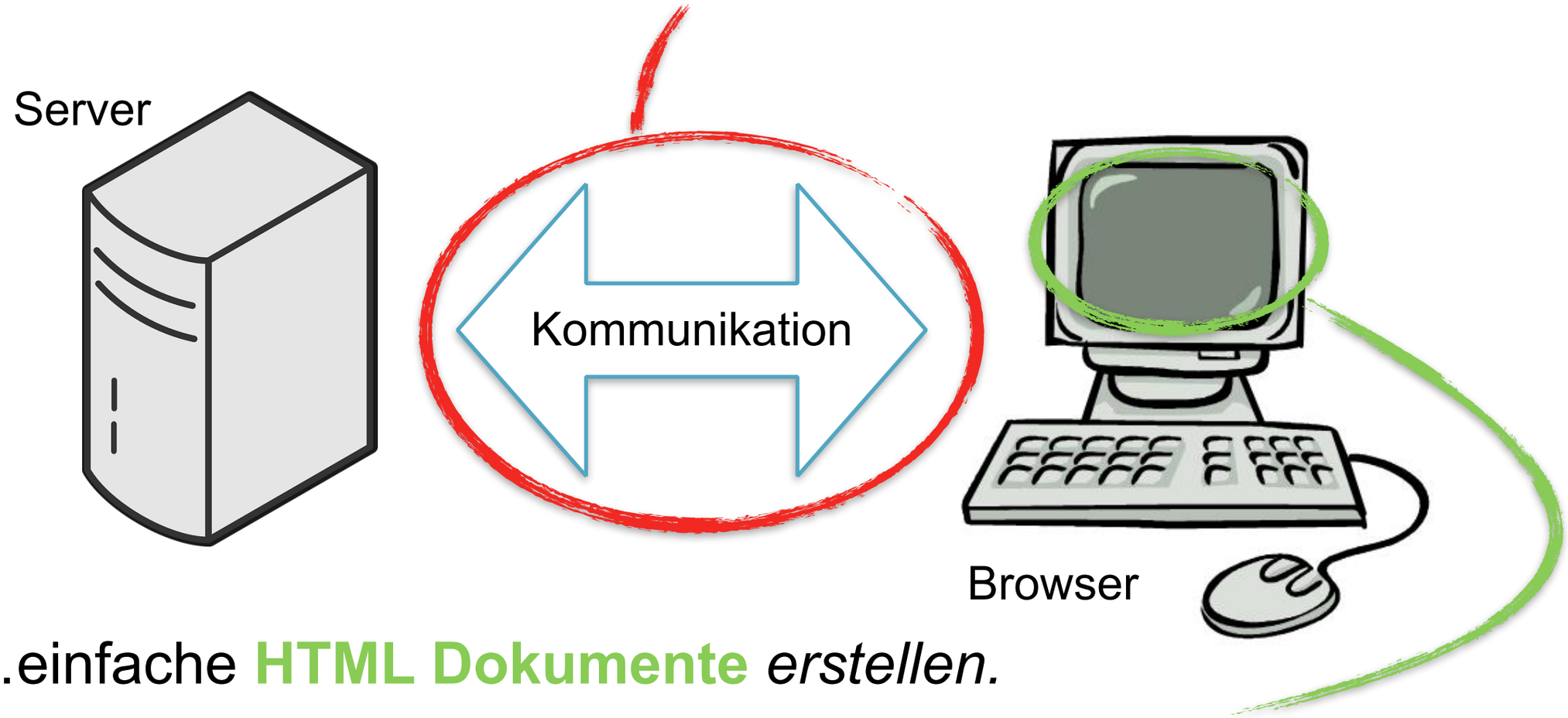
christian.zirpins@hs-karlsruhe.de

Web Apps mit HTML5

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Nach dieser Vorlesung können Sie…

- …**Basic HTTP Authentication** *verstehen* und *einsetzen*.
- …den Unterschied zwischen **HTTP vs. HTTPS** *erklären*



- …einfache **HTML Dokumente** *erstellen.*
- *...*die Ziele und Inhalte von **HTML5** *beschreiben*.
- …Benutzereingaben mit **Formularen** *umsetzen*.

Authentifizierung

Authentifizierung

Bislang: HTTP als **anonymes**, **zustandsloses** Request/Response Protokoll. Der gleiche Request, von verschiedenen Clients gesendet, wird in der exakt gleichen Weise behandelt.

Jetzt: Identifikation mit
A. HTTP Header
B. Client IP Adressverfolgung
C. User Login (HTTP Basic Authentication)
D. Fat URLs

In späterer Vorlesung: Cookies & Sessions.

User-spezifische HTTP Header Felder

From	Request	User Email Adresse	Hauptsächlich von Crawlern benutzt
User-Agent	Request	User Browser	Erlaubt Anpassung an ein Gerät
Referer	Request	Seite von der der User kommt	Grobe Art, User auszuspionieren
Authorization	Request	Username & password	
X-Forwarded-For	Request (Extension)	Client IP Adresse	
Cookie	Request	Server-generiertes ID Label	

Client IP Adresse

- **Client IP Adresse** als User ID (wenn nicht im HTTP Header, dann über TCP Connection)
- Möglich, **wenn** jeder User eine eigene IP Adresse hat, diese IP sich selten ändert und der Server die IP Adresse für jeden Request feststellen kann.

Probleme

A. IP Adressen beschreiben die **Maschine**, nicht den User
B. Internet Service Provider weisen Usern IP Adressen **dynamisch** zu
C. User verbinden sich mit dem Web über **Firewalls** (verbergen die echte IP Adresse)
D. HTTP **Proxies** und **Gateways** öffnen neue TCP Connections (IP des Proxy/Gateway wird angezeigt), `X-Forwarded-For` hilft ggf.

Client IP Adresse

- **Client IP Adresse** als User ID (wenn nicht im HTTP Header, dann über TCP Connection)
- Möglich, **wenn** jeder User eine eigene IP Adresse hat, diese IP sich selten ändert und der Server die IP Adresse für jeden Request feststellen kann.

Probleme

A. IP Adressen beschreiben die **Maschine**, nicht den User
B. Internet Service Provider weisen Usern IP Adressen **dynamisch** zu
C. User verbinden sich mit dem Web über **Firewalls** (verbergen die echte IP Adresse)
D. HTTP **Proxies** und **Gateways** öffnen neue TCP Connections (IP des Proxy/Gateway wird angezeigt), `X-Forwarded-For` hilft ggf.

Fat URLs

```
<a href="/browse/-/229220/ref=gr_gifts/002-1145265-8016838">Gifts</a>
<a href="/wishlist/ref=gr_pl1_/002-1145265-8016838">Wish List</a>
```

- Verfolgung durch **Generierung eindeutiger URLs** pro User
  - Beim ersten Besuch einer Seite (innerhalb einer Website) generiert der Server eine **eindeutige ID**
  - Server leitet Client zur fat URL um ('**redirect**', Status Code 3**)
  - Server **schreibt HTML um**, wenn HTTP Requests mit Fat URLs eintreffen (ergänzt ID zu allen Hyperlinks um die Information zu erhalten)
- Unabhängige HTTP Transaktionen können zu einer zusammenhängenden "Session" verbunden werden.

Frage: Was ist ein mögliches Problem bei Fat URLs?

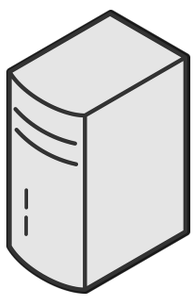
Fat URLs

- Fat URLs sind **nicht schön**
- Fat URLs können **nicht geteilt** werden (URL könnte später verschwinden oder User teilen unabsichtlich private Informationen)
- Fat URLs vereiteln das **Caching** (eine URL pro User/Seite statt eine URL pro Seite)
- Extra **Server Last** (HTML Seite muss umgeschrieben werden)
- User könnte “**entkommen**” (ID ist verloren wenn User außerhalb der Website navigiert, außer er speichert sie als Bookmark)

HTTP Basic Authentication

- Server fragt den User **explizit** nach Authentisierung (Username und Passwort)
- HTTP hat eingebauten Mechanismus, um User-Informationen zur Website zu leiten:
  - `WWW-Authenticate` und
  - `Authorization` Header.
- HTTP ist zustandslos: einmal angemeldet sendet der Browser die Login-Informationen mit jedem Request.

HTTP Basic Authentication



```
GET /index.html HTTP/1.1
host: www.microsoft.com
```

```
HTTP/1.1 401 Login Required
WWW-Authenticate: Basic realm="RealmName"
```

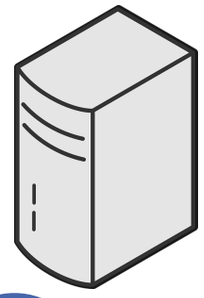
1

2

Auth.-Algorithmus und ‚Sicherheitsbereich' (realm)

Browser zeigt Login Screen

```
GET /index.html HTTP/1.1
host: www.microsoft.com
Authorization: Basic am9objpteXB3ZA==
```



```
HTTP/1.1 200 OK
Content-length: 1234
Content-type: text/html
```

3

Base-64 Codierung

In weiteren HTTP-Requests zur Website, überträgt der Browser automatisch Username/Passwort wenn gefragt (bzw. immer).

HTTP Basic Authentication

- Username und Passwort werden durch Doppelpunkt verbunden und zu **Base64 Kodierung** konvertiert (z.B. **`john:mypwd`**)

Base64 Kodierung stellt sicher, dass nur HTTP-kompatible ASCII-Zeichen in Messages kommen (Eingabe: 8-Bit Binärdaten; z.B. Text mit internationalen Zeichen)

Bilder können so kodiert werden

Normandië	Tm9ybWFuZGnDqw==
Karlsruhe	S2FybHNydWhl
España	RXNwYcOxYQ==

https://opinionatedgeek.com/Codecs/Base64Decoder

HTTP Basic Authentication: sicher?

- Username und Passwort können **einfach entschlüsselt** werden (werden als,"Klartext" über das Netz gesendet)
- User tendieren zur **Wiederholung** von Login/Passwort Kombinationen. Eine unkritische Website könnte Basic Authentication ohne SSL verwenden. Jemand könnte dies dann abfangen und auf kritischen Websites probieren.
- Kein Schutz gegen **gefälschte Server** (die anstatt der eigentlichen Server tätig werden)

HTTP Basic Authentication: Fazit

Basic Authentication verhindert **versehentlichen Zugriff** von neugierigen Usern (Privatsphäre ist erwünscht aber nicht unbedingt erforderlich).

Basic Authentication ist für **Personalisierung** und Zugriffskontrolle in einer "freundlichen" Umgebung (Intranet) nützlich.

"In der Wildnis" sollte Basic Authentication immer in Kombination mit **sicherem HTTP (z.B. https)** verwendet werden **-** Vermeidet das Senden von Username/Passwort im Klartext über das Netzwerk.

Sicheres HTTP

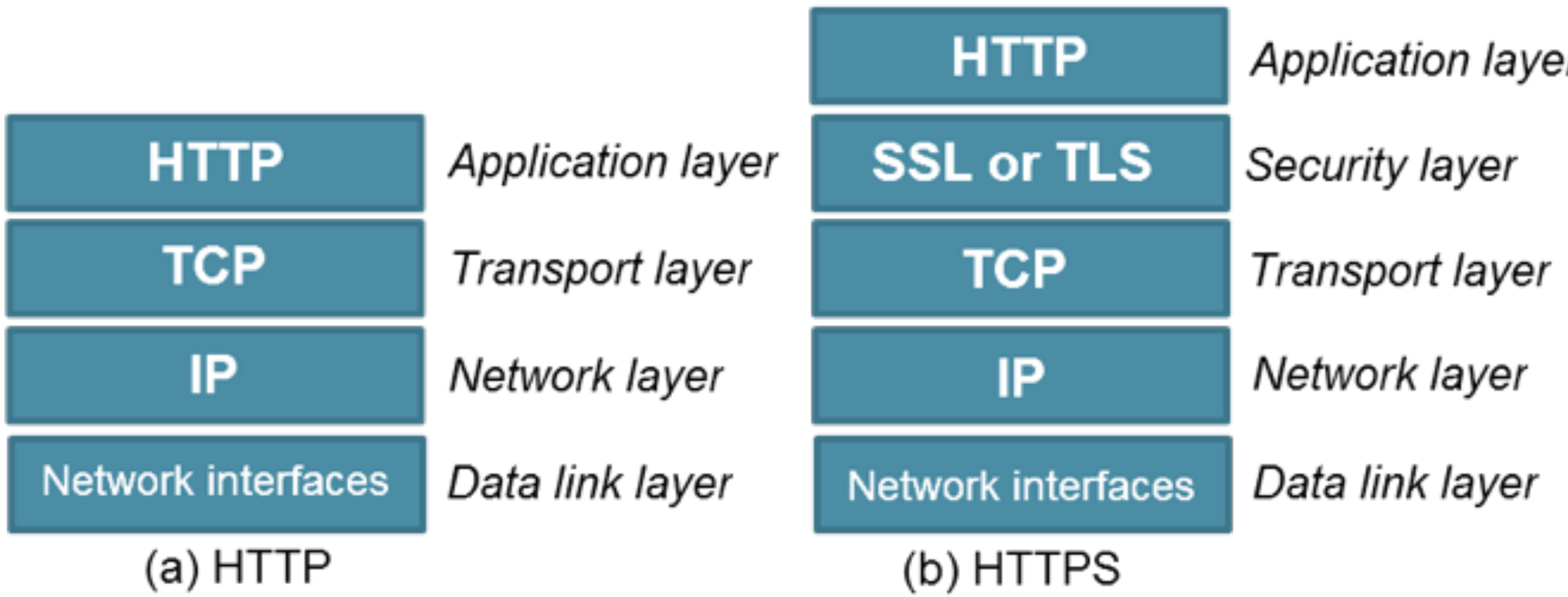
Sicheres HTTP

- Bisher: leichtgewichtige Authentifizierung
  - Nicht sinnvoll für Einkäufe, Bank Transaktionen etc.
- Sicheres HTTP sollte Folgendes bereitstellen:

A. **Server Authentifizierung**
   (Client ist sicher mit dem richtigen Server zu sprechen)
B. **Client Authentifizierung**
   (Server ist sicher mit dem richtigen Client zu sprechen)
C. **Integrität**
   (Client und Server sind sicher, dass die Daten korrekt/unverfälscht sind)
D. **Vertraulichkeit**
   (durch Verschlüsselung können nur Client und Server Inhalte lesen)
E. **Effizienz**
F. **Anpassbarkeit**
   (auf den aktuellen Stand der Verschlüsselungstechnik)

Sicheres HTTP: HTTPS

- HTTPS ist die populärste sichere Form von HTTP
- URL-Scheme ist **`https://`** statt **`http://`**
- Request und Response Daten werden verschlüsselt, bevor sie über das Netz gesendet werden (**TLS: Transport Level Security**)



Client & Server handeln die kryptografischen Protokolle aus

OpenSSL & HTTPS

- SSL ist ein kompliziertes binäres Protokoll.
- **OpenSSL** ist das populärste Open Source Werkzeug um SSL und TLS (Transport Layer Security) sowie eine Reihe von kryptografischen Algorithmen zu implementieren.

Authentifizierung

Zusammenfassung

HTTP stellt sicher, dass Inhalte …

A. …**korrekt identifiziert** werden können.

B. …**richtig entpackt** werden können.

C. …**aktuell** sind.

D. …in **passendem Format** vorliegen.

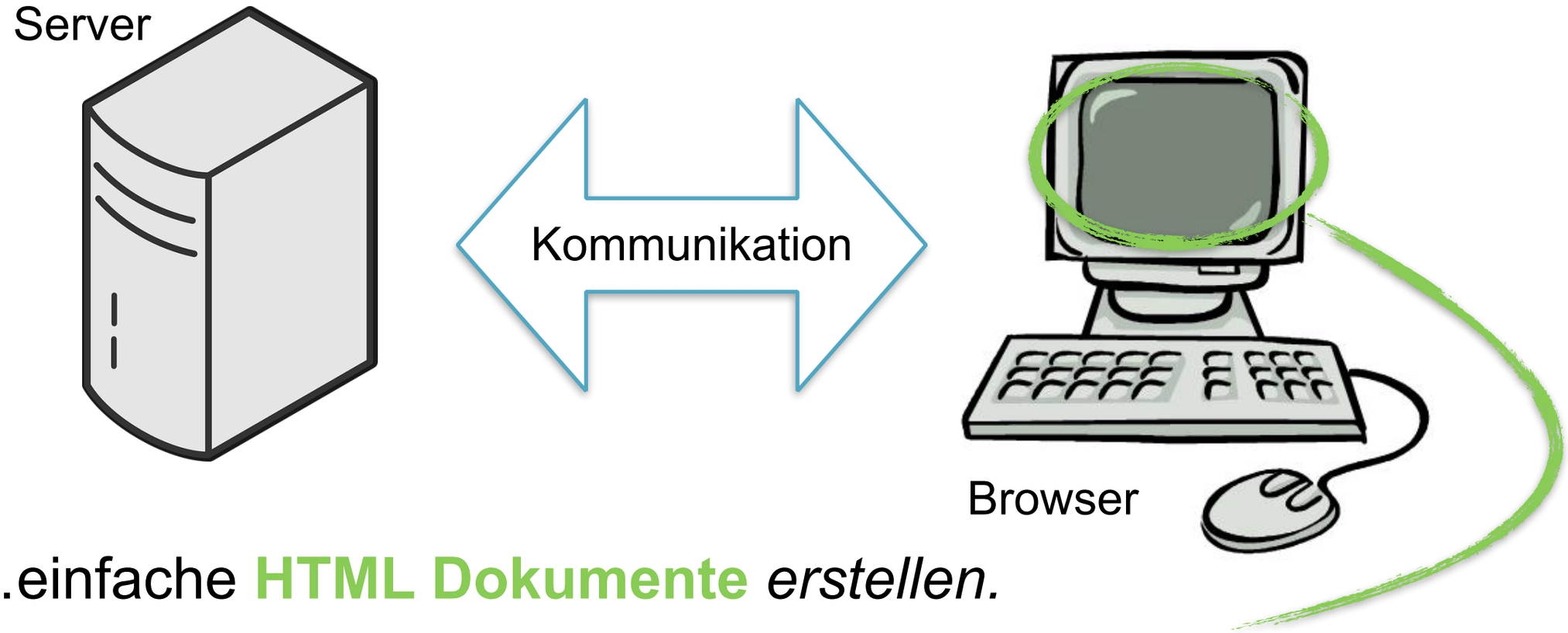
E. …**vollständig** und **unverfälscht** ankommen.

HTTPS stellt sicher, dass Inhalte …

A. …**sicher transportiert** werden.

HTML5 näher betrachtet

Nach dieser Vorlesung können Sie…



- …einfache **HTML Dokumente** *erstellen.*
- *…*die Ziele und Inhalte von **HTML5** *beschreiben*.
- …Benutzereingaben mit **Formularen** *umsetzen*.

Nach Lesen von Kapitel 2 des Begleitbuchs sollten Sie Folgendes können…

- Erstellen einer einfachen **HTML5 Seite** die korrekt validiert wird
- Nutzung von **`img`, `a`, `ul`, `p`, `div`, …** **Tags**
- Erklären der Verwendung von **HTML vs. CSS**
- Die **Struktur** einer gegebenen HTML Seite identifizieren
- Erstellen eines **DOM Baums** aus einer gegebenen Struktur

... oder nicht?

Kapitel 2 auf einer Folie

```
<!doctype html>
<html>
<head>
    <title>My First Web App</title>
</head>

<body>
    <h1>Hello World!</h1>
    <p align="center">Nice to meet you.</p>
</body>
</html>
```

- Ein **HTML Dokument** enthält **Tags** und **Inhalt**
- **Tags** sind Metadaten
- Tags strukturieren den Inhalt des Dokuments

Kapitel 2 auf einer Folie

Zeigt dem Browser die HTML Version an

```
<!doctype html>
<html>
<head>
    <title>My First Web App</title>
</head>
<body>
    <h1>Hello World!</h1>
    <p align="center">Nice to meet you.</p>
</body>
</html>
```

header beschreibt das Dokument

Tags

Inhalt

body enthält den Seiteninhalt

- Ein **HTML Dokument** enthält **Tags** und **Inhalt**
- **Tags** sind Metadaten
- Tags strukturieren den Inhalt des Dokuments

Die 'gerenderte' Seite zeigt nicht die Tags, nur den Inhalt

Websites vs. Web Apps

"As Web browsers and the Web engine components that power them become ubiquitous […], developers are increasingly **using Web technologies** to build **applications** and are relying on Web engines as **application runtime environments**.

Examples of applications now commonly built using Web technologies include […] **games**, **multimedia applications**, **maps**, […] **interactive design applications**, and PIM (**email**, **calendar**, etc) systems.

W3C: Web Applications Working Group

Was ist HTML5?

- Eine Menge zusammenhängender Technologien (core HTML5, CSS, JavaScript) die gemeinsam Web Inhalte ermöglichen.
- Core **HTML5**: zeichnet Inhalte aus (Markup).
- **CSS**: steuert das Erscheinungsbild von ausgezeichnetem Inhalt.
- **JavaScript**: manipuliert Inhalte von HTML Dokumenten, reagiert auf Benutzerinteraktion, programmiert diverse Plattform APIs.

- Moderne Web (App) Entwicklung bedingt Wissen über alle drei Techniken.
- Vor HTML5: XHTML und HTML 4.01

Nicht alle Browser unterstützen alle Funktionen.

http://caniuse.com

Entwicklungsgeschichte von HTML5

- Initiale Liste der HTML Tags (1991/92) war einfach & **statisch**:
  `<title> <a> <isindex> <plaintext> <listing> <p> <h1> <address> <hp1> <dl> <dt> <ul>`

- JavaScript erschien 1995, entwickelt von Netscape - Beginn von clientseitigem **dynamischem** Scripting für den Browser.

JavaScript ist nicht Teil von HTML, aber HTML5 nimmt es als gegeben an.

- Plugins (z.B. Adobe Flash, 1996) wurden entwickelt, um die Möglichkeiten von HTML zu erweitern.

HTML5: bringt reiche Inhalte *direkt* in den Browser zurück.

- *Semantisches* HTML wurde ein populärer Wunsch für die automatische Verarbeitung von Web Inhalten im großen Stil.

`<div> vs. <footer>`

Wer entscheidet über den HTML Standard?

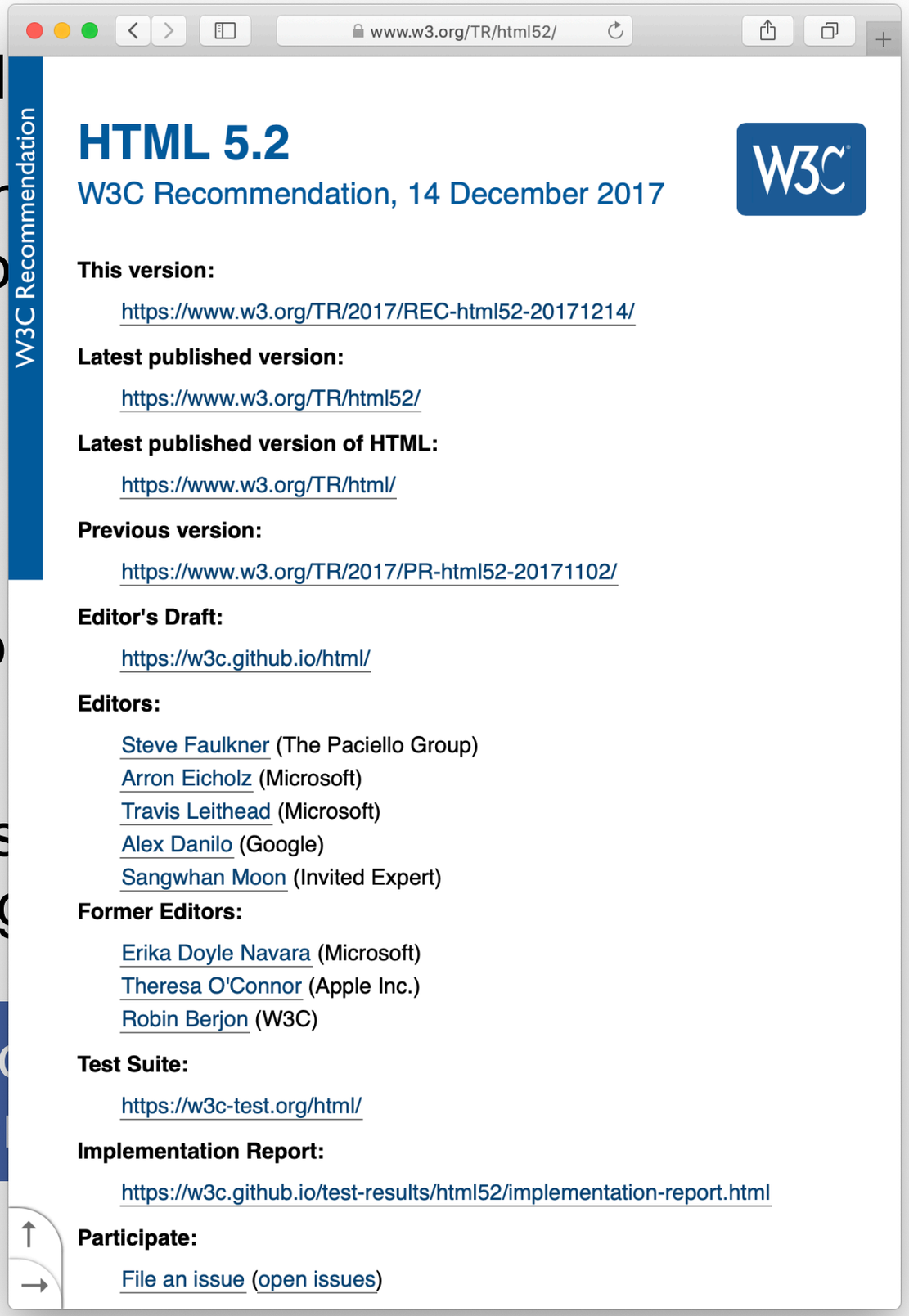
- HTML ist weit verbreitet - das macht die Standardisierung zäh.
- Viele verschiedene Interessengruppen sind Teil des W3C **HTML Working Group** (Microsoft, Apple, Google, Mozilla, Nokia, Adobe, Intel, Baidu, etc.)
- HTML5: Candidate Recommendation (Empfehlungs-Kandidat) in Q4-2012, **W3C Recommendation in Q4-2014**
- HTML5.1: Candidate Recommendation in Q1-2015, **W3C Recommendation in Q4-2016**
- In der Praxis: W3C standardisiert, was die Browser-Hersteller für die Implementierung gewählt und als Einigung erzielt haben.

W3C Recommendation: Feature sind Stabil und in mehreren (2+) Browsern implementiert

Wer entscheidet über den HTML Standard?

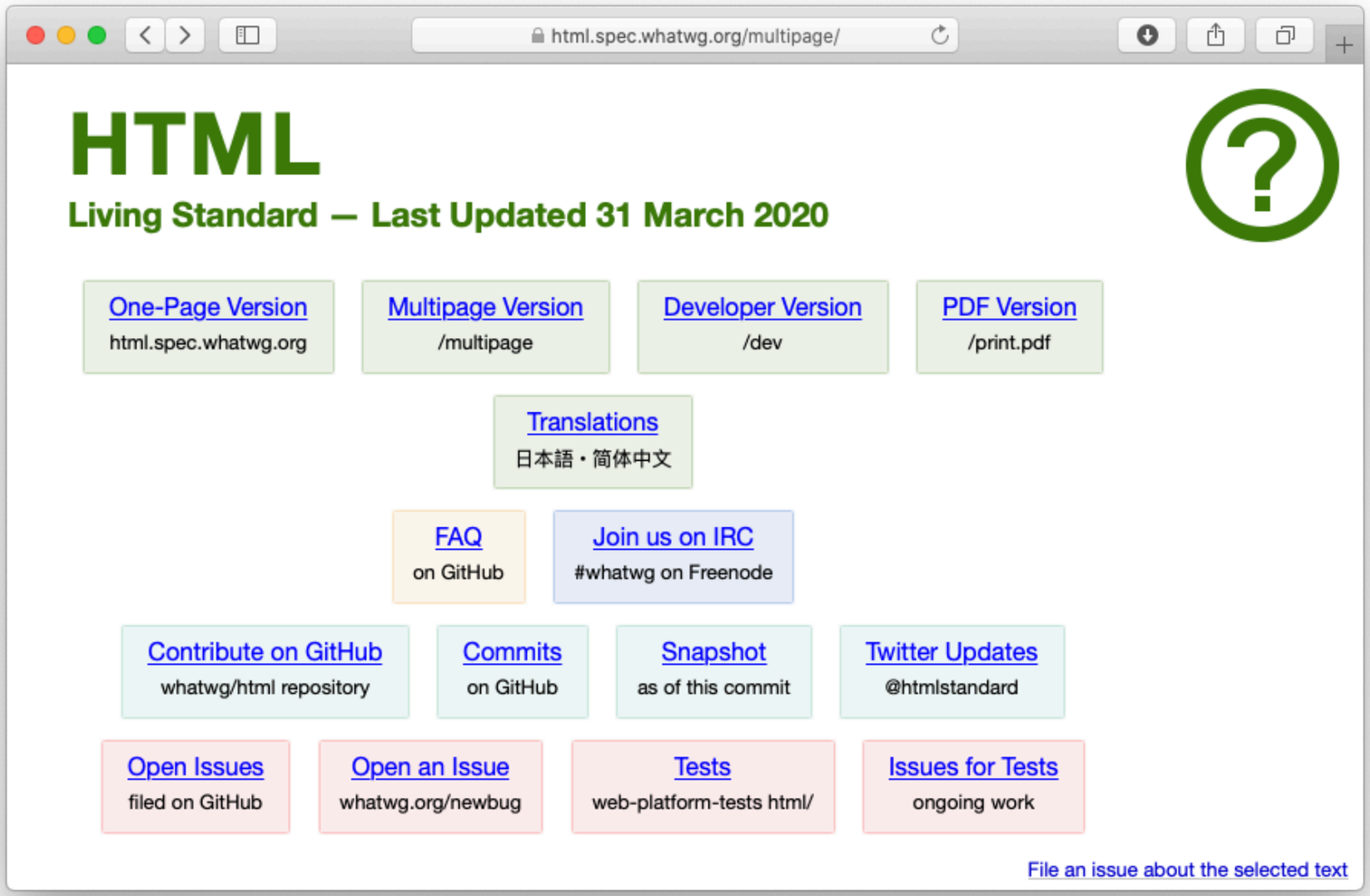
- HTML ist weit verbreitet - das macht d
- Viele verschiedene Interessengruppen **Working Group** (Microsoft, Apple, Go Intel, Baidu, etc.)
- HTML5: Candidate Recommendation Q4-2012, **W3C Recommendation in**
- HTML5.1: Candidate Recommendatio **Recommendation in Q4-2016**
- In der Praxis: W3C standardisiert, was Implementierung gewählt und als Einig

W3C Recommendation: Feature sinc in mehreren (2+) Browsern impleme



WHATWG (what-wee-gee)

- Die **Web Hypertext Application Technology Working Group** wurde 2004 als Arbeitsgruppe gegründet, um Web Technologien unabhängig vom W3C weiterzuentwickeln.
- Steering Group: Mozilla, Microsoft, Google, Apple.
- Standardisierung von HTML erfolgt seit May 2019 als WHATWG „Living Standard“



HTML5 ist Modular and Komplex

Es gibt mehr als nur Core HTML5.

- **Web Workers**: Web Applikationen können Worker abspalten, um Prozesse (Skripte) auszuführen, die parallel zur Hauptseite laufen.
- **Web Storage**: clientseitiger Speicher.
- **WebSocket API**: bidirektionale Kommunikation mit serverseitigen Prozessen
- **WebRTC**: Echtzeit Kommunikation *zwischen* Browsern (für Videokonferenzen u.a.).
- **HTML Media Capture**: erlaubt Benutzerzugriff zu den Medien-Aufnahmemechanismen eines Gerätes (z.B. webcamtoy Demo).
- ...

W3C TR Specifications (153 aktuelle Specs):
http://html5-overview.net

HTML Media Capture: webcamtoy.com

Semantics vs. Präsentation

- HTML5 führte eine Reihe semantischer HTML Elemente ein u.a. **`<article> <footer> <header> <main> <aside> <section> <output>`**
- Semantische Elemente tragen Bedeutung aber erzwingen keine bestimmte Präsentation - **POSH** (Plain Old Semantic HTML)
- Einige ältere HTML-Elemente (vor HTML5) erzwingen eine bestimmte Präsentation, z.B. **`<b>`** or **`<i>`** .
- Stark genutzte HTML Elemente können nicht einfach als "obsolet" erklärt werden.

Beim Erstellen eines HTML Dokuments sollte immer das am stärksten spezifische Element zur Repräsentation eines Inhalts gewählt werden.

Es gibt viele HTML Elemente

- Normalerweise wird nur eine kleine Anzahl genutzt.
- HTML5 führte eine große Menge neuer Elemente ein.
- Der HTML Living Standard erweitert diese Menge kontinuierlich.
- Viele sind noch in einem experimentellen Zustand, d.h. sie werden noch nicht von allen großen Browser-Herstellern unterstützt.

HTML 5 NEW TAG

TAG NOT SUPPORTED IN HTML 5

<a>	Defines a hyperlink href, hreflang, media, rel, sizes, type
<abbr>	Defines an abbreviation
<acronym>	Used to define an embedded acronyms
<address>	Defines an address element
<applet>	Used to define an embedded applet
<area>	Defines an area inside an image map alt, coords, href, hreflang, media, rel, shape, sizes, target, type
<article>	Defines an article
<aside>	Defines content aside from the page content
<audio>	Defines sound content autobuffer, autoplay, controls, src
<b>	Defines bold text
<base>	Defines a base URL for all the links in a page href, target
<basefont>	Used to define a default font-color, font-size, or font-family for all the document
<bdo>	Defines the direction of text display dir
<big>	Used to make text bigger
<blockquote>	Defines a long quotation cite
<body>	Defines the body element
 	Inserts a single line break
<button>	Defines a push button autofocus, disabled, form, formaction, formenctype, formmethod, formnovalidate, formtarget, name, type, value
<canvas>	Defines graphics height, width
<caption>	Defines a table caption
	Used to center align text and content
<cite>	Defines a citation
<code>	Defines computer code text autobuffer, autoplay, controls, src
<col>	Defines attributes for table columns
<colgroup>	Defines groups of table columns span
	Defines a command button

<datalist>	Defines a dropdown list
<dd>	Defines a definition description
	Defines deleted text
<details>	Defines details of an element open
<dialog>	Defines a dialog (conversation)
<dir>	Used to define a directory list
	Defines a definition list
<dt>	Defines a definition term
<embed>	Defines external interactive content or plugin
<fieldset>	Defines a fieldset disabled, form, name
<figure>	Defines a group of media content, and their caption
<font>	Used to define font face, font size, and font color of text
<footer>	Defines a footer for a section or page
<form>	Defines a form accept-charset, action, autocomplete, enctype, method, name, novalidate, target
<frame>	Used to define one particular window (frame) within a frameset
<frameset>	Used to define a frameset, which organized multiple windows (frames)
<h1> to <h6>	Defines header 1 to header 6
<head>	Defines information about the document
<header>	Defines a header for a section or page
<hgroup>	Defines information about a section in a document
<hr>	Defines a horizontal rule
<html>	Defines an html document manifest, xmlns
<i>	Defines italic text
<iframe>	Defines an inline sub window height, name, sandbox, seamless, src, width
<img>	Defines an image alt, src, height, ismap, usemap, width
<input>	Defines an input field accept, alt, autocomplete, autofocus, checked, disabled, form, formaction, formenctype, formmethod, formnovalidate, formtarget, height, list, max, maxlength, min, multiple,

<ins>	Defines inserted text cite, datetime
<keygen>	Defines a generated key in a form autofocus, challenge, disabled, form,
<kbd>	
<label>	Defines an inline sub window for, form
<legend>	
	Defines a list item value
<link>	href, hreflang, media, rel, sizes, type
<map>	Defines an image map name
<mark>	
<menu>	Defines a menu list label, type
<meta>	Defines meta information charset, content, http-equiv, name
<meter>	Defines measurement within a predefined range high, low, max, min, optimum, value
<nav>	Defines navigation links
<noframes>	Used to display text for browsers that do not handle frames
<noscript>	Defines a noscript section
<object>	Defines an embedded object data, form, height, name, type, usemap, width
	Defines an ordered list reversed, start
<optgroup>	Defines an option group label, disabled
<option>	Defines an option in a drop-down list disabled, label, selected, value
<output>	Defines some types of output for, form, name
<p>	Defines a paragraph
<param>	Defines a parameter for an object name, value
<pre>	Defines preformatted text
<progress>	Defines progress of a task of any kind max, value
<q>	Defines a short quotation cite
<rp>	Used in ruby annotations to define what to show browsers that to not support the ruby element
<rt>	Defines explanation to ruby annotations
<ruby>	Defines ruby annotations

<samp>	Defines sample computer code
<script>	Defines a definition list async, type charset defer, src
<section>	Defines a section cite
<select>	Defines a selectable list autofocus, disabled, form, multiple, name, size
<small>	Defines small text
	Defines media resources media, src, type
	Defines a section in a document
	Defines strong text
	Defines a style definition type, media, scoped
	Defines sub/super-scripted text
<table>	Defines a table summary
<tbody>	Defines a table body summary
<td>	Defines a table cell colspan, headers, rowspan
<textarea>	Defines a text area autofocus, cols, disabled, form, maxlength, name, placeholder, readonly, readonly, required, rows, wrap
<tfoot>, <thead>	Defines a table footer / head
<th>	Defines a table header colspan, headers, rowspan, scope
<time>	Defines a date/time datetime
<title>	Defines the document title
<tr>	Defines a table row datetime
<tt>	Used to define teletype text
<u>	Used to define underlined text
	Defines an unordered list
<var>	Defines a variable
<video>	Defines a video autobuffer, autoplay, controls, height, loop, src, width

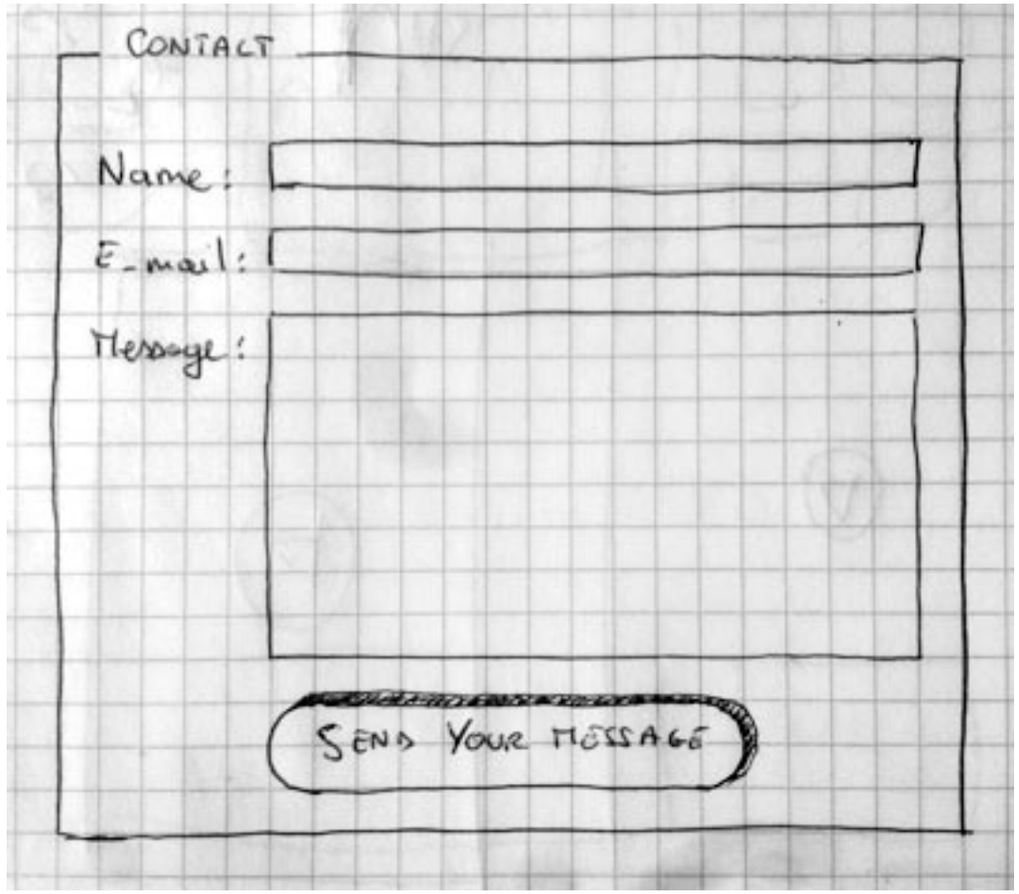
HTML5 TAG CHEAT SHEET

Created by WebsiteSetup.org

HTML Formulare

HTML Formulare

- Daten an den Server senden: mit JavaScript oder purem HTML.
- Die HTML-Variante basiert auf **Formularen**: HTML-Elemente, die eine einfache Möglichkeit bieten, Daten beim Client abzufragen und an den Server zu senden.



https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/My_first_HTML_form

Einfaches Formular


```
<form action="http://www.google.com/search" method="get" target="_blank">
            Input: <input name="q" type="text" />
            <input type="submit" value="Search Google" />
</form>
```

- `action` Attribut enthält URL eines Skripts, das die Daten verarbeiten wird.
- Formular ist mit Pure.css aufgehübscht (http://purecss.io).
- Eine Webseite kann viele Formulare enthalten.

Struktur eines Formulars

```
<form action="X" method="post|get">
                form element
                form element
                form element
                <input type="submit" />
</form>
```

Struktur eines Formulars HTML5

```
<form>
            form element
            form element
            form element
            <input type="submit"  formaction="X1" formmethod="post|get" />
            <input type="submit"  formaction="X2" formmethod="post|get" />
            <input type="submit"  formaction="X3" formmethod="post|get" />
</form>
```

Formularattribute

Attribut	Wert
target	Ort der Antwort
action	URL
autocomplete[HTML5]	*on* oder *off*
method	*get* oder *post*
name	Name des Formulars
novalidate[HTML5]	novalidate

GET vs. POST

- GET/POST erzeugen ein **assoziatives Datenfeld (Map)**
  - **Name-Wert-Paare**
    - *Name* des Interaktionselements
    - *Wert* der Eingabe
- **Beispiele**
  - Email => mailbox@hs-karlsruhe.de
  - Telefon => +49(0)721 925-0

GET vs. POST

- **GET**
  - Name-Wert-Paare erscheinen in der URL als **Query Parameter**.
    - http://www.google.de/search**?q=hska**
  - URLs können nicht übermäßig lang sein.
    - Empfehlung: 255 Bytes / Praxis: ca. 2000 Zeichen.
  - Bookmarks enthalten Formularwerte.
- **POST**
  - Name-Wert-Paare sind in der URL nicht sichtbar.
  - Unbegrenzte Menge an Daten.
    - Dateien sollten auf diese Weise übertragen werden.
  - Bookmarks enthalten keine Formularwerte.

Die gängigsten Interaktionselemente

- **input** (diverse Eingabefelder)
- **textarea** (mehrzeilige Texteingabe)
- **button** (Knopf)
- **select** (Drop Down Liste)
- **option** (Listenoption)
- **optgroup** (Listengruppe)
- **fieldset** (Gruppierung von Feldern)
- **label** (Beschriftung)
- **hidden** (unsichtbar)

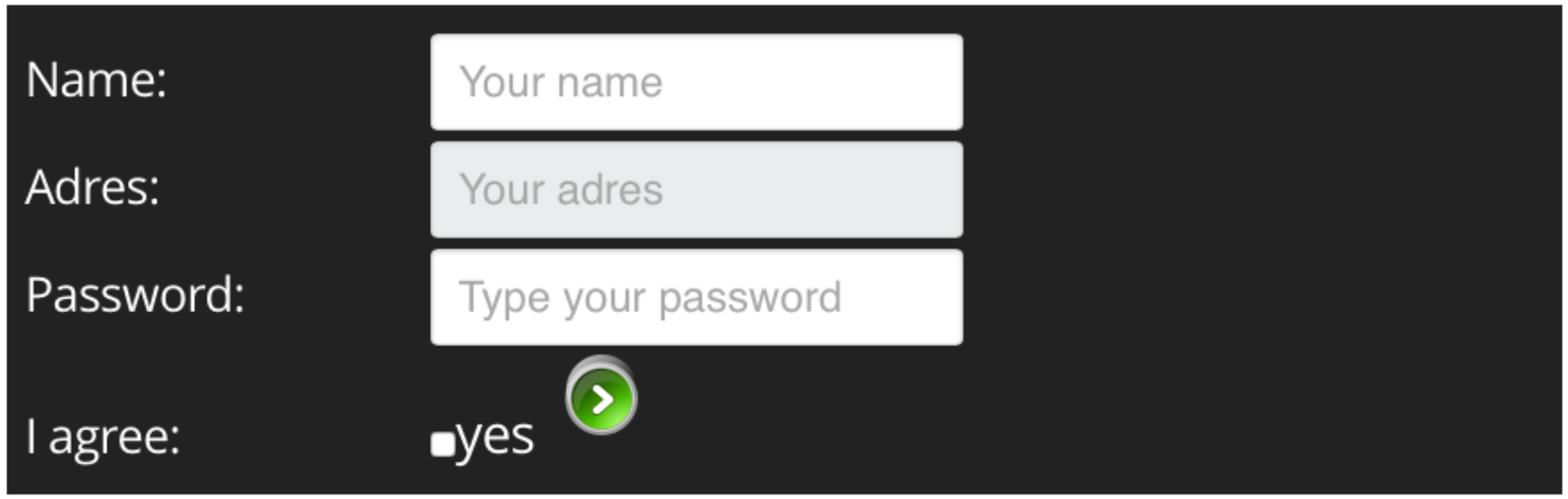
Das `input` Element

- `name` Attribut: Name des **Query Parameters** beim Submit.
- `value` Attribut: Der (initiale) **Text** des Interaktionselements.
- `type='checkbox'` wähle *keinen / einen / mehrere*
- `type='radio'` wähle *keinen / einen*


```
<form action="php/vars.php" method="get">
  Textfeld:         <input type="text" name="text_field_name" size="10" />
  Password Feld:    <input type="password" name="password_field_name" size="20" />
  Radio Button:     <input type="radio" name="gender" value="M" />Male
                    <input type="radio" name="gender" value="F" />Female
  Checkbox:         <input type="checkbox" name="sports" checked="checked" />sports
                    <input type="checkbox" name="travel" />traveling
                    <input type="checkbox" name="reading" />reading
  <input type="submit" value="Submit data">
</form>
```

Einige Input Attribute

placeholder[HTML5],
required[HTML5],
pattern[HTML5],
maxlength




```
<form action="php/vars.php" method="get">
 Name:<input type="text" name="n" placeholder="Your name" pattern="[A-Z]+" />

 Address:<input type="text" name="a" placeholder="Your address" disabled />

 Password:<input type="password" name="p" placeholder="Your password" maxlength=4/>

 I agree:<input type="checkbox" name="agree" value="yes" required />yes

        <input type="image" src="arrow_button.png" height="60" name="submit"
          alt="submit button" />
</form>
```

`input` Attribut `hidden`

- Element erscheint nicht in sichtbarer Form.
- Kann nicht von Benutzer gelesen oder geändert werden.
- Nützlich um Zusatzinformationen zu verfolgen (z.B. von welcher Seite eines Web Portals ein Benutzer ein Formular genutzt hat)


```
<form action="php/vars.php" method="get">
  Text input: <input type="text" name="text_input" />
  <input type="hidden" name="hidden_page" value="3" />
  <input type="submit" value="Submit" />
</form>
```

HTML `data-*` Attribute

- **Eigene Datenattribute** sind für alle HTML Elemente möglich.
- **Eigene Attribute sind unsichtbare Daten**
- Attribute haben den Prefix `data-`

```
<a class="todo" data-creator="john"
   data-done="22-12-2014">Christmas
shopping</a>
```

`hidden` vs. `data-*`

- `hidden` wird oft in Formularen genutzt, um Daten zum Server zurückzusenden.
- `data-*` Attribute werden Elementen beigefügt und sollen diese Beschreiben.
- Eigene Attribute sind ein einfacher Weg um zusätzliche Informationen über ein Element in einer Ressource zu senden.

`name` vs. `id` vs. `value`

- `name`
  - Tags: `form`, `input`, `select`, `textarea` (u.a.)
  - Müssen nicht eindeutig sein.
  - Name-Wert-Paare durch Formulare übertragen.
- `id`
  - Globales Attribut: kann für jedes Element definiert werden.
  - Müssen eindeutig sein.
  - CSS: #
- `value`
  - Name-Wert-Paare durch Formulare übertragen.

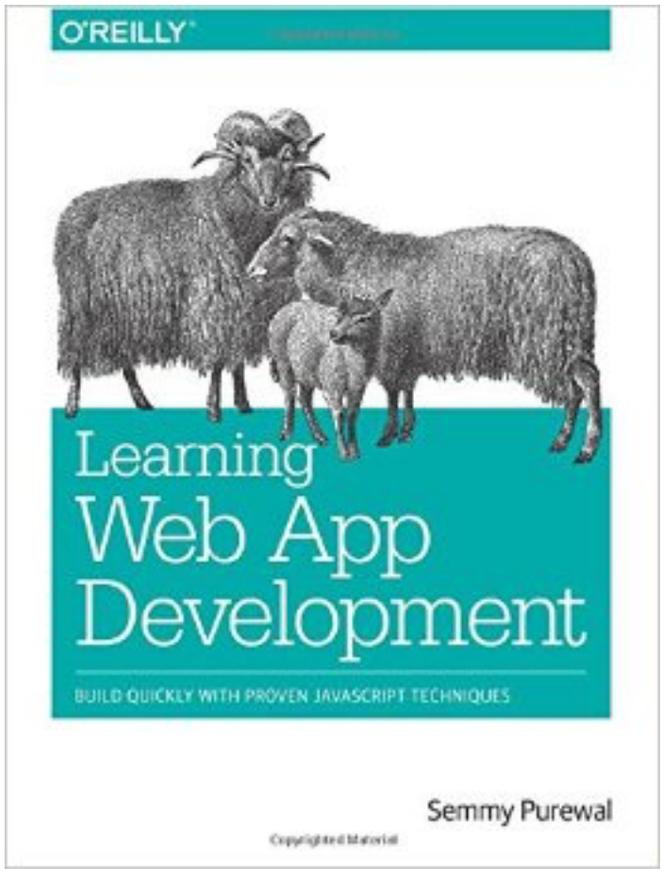
Zusammenfassung

Heute haben wir Folgendes behandelt

- Authentifizierung
- HTML5 näher betrachtet
- HTML Formulare

Literatur

- **Learning Web App Development, Kapitel 2**
- Empfehlung: Mark Pilgrim, *"HTML5 Up and Running"*, O'Reilly, 2010 (Online: http://diveintohtml5.info)



- Die nächste Vorlesung setzt CSS3 Grundkenntnisse voraus!
  - **Bitte lesen Sie *vorab***
    Learning Web App Development, Kapitel 3