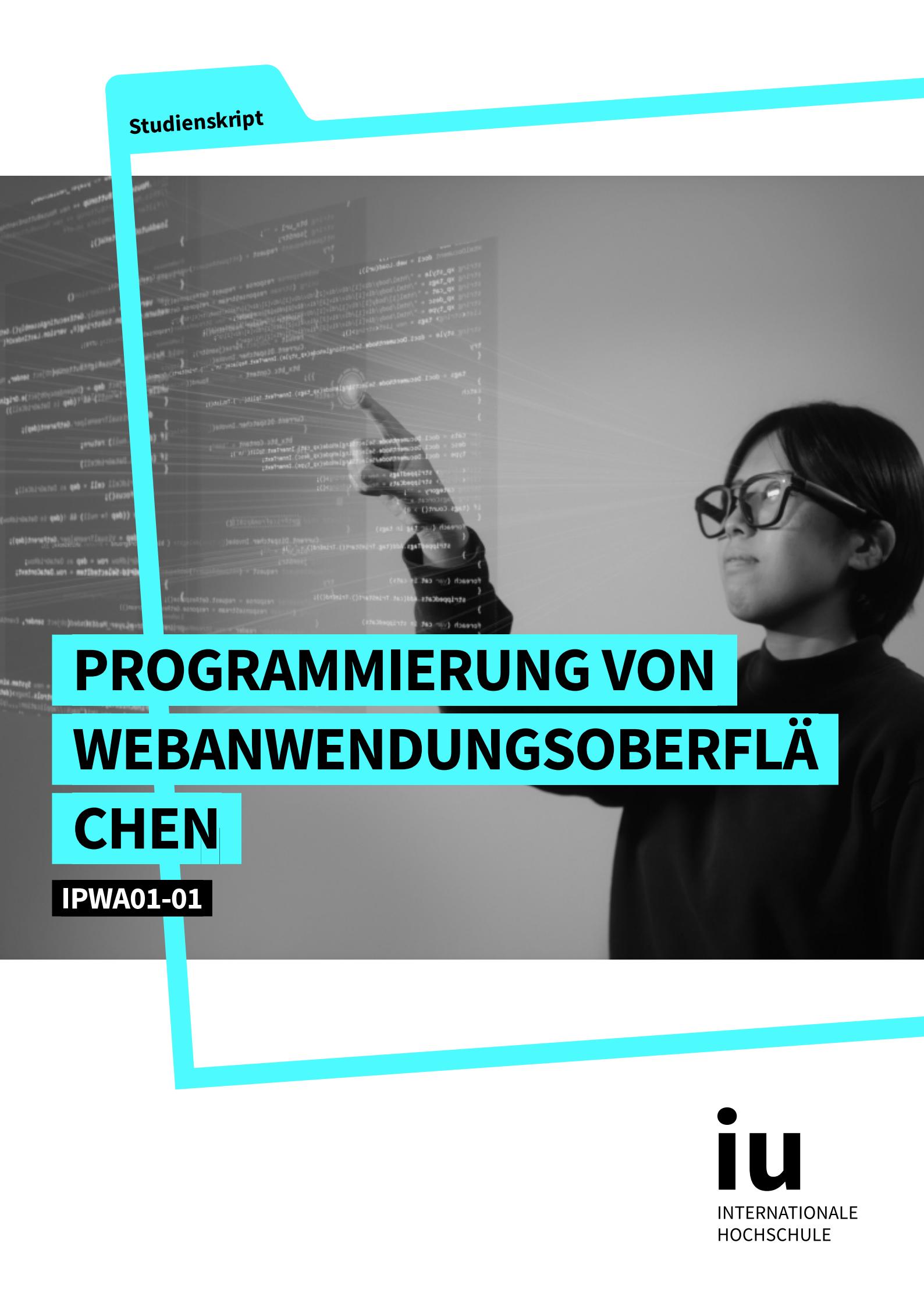


Studienskript



# PROGRAMMIERUNG VON WEBANWENDUNGSOBERFLÄ CHEN

IPWA01-01



# **PROGRAMMIERUNG VON WEBANWENDUNGSOBERFLÄCHEN**

## **IMPRESSUM**

Herausgeber:  
IU Internationale Hochschule GmbH  
IU International University of Applied Sciences  
Juri-Gagarin-Ring 152  
D-99084 Erfurt

Postanschrift:  
Albert-Proeller-Straße 15-19  
D-86675 Buchdorf  
[media@iu.org](mailto:media@iu.org)  
[www.iu.de](http://www.iu.de)

IPWA01-01  
Versionsnr.: 001-2023-1113  
Christian Winkler

© 2023 IU Internationale Hochschule GmbH  
Dieser Lehrbrief ist urheberrechtlich geschützt. Alle Rechte vorbehalten.  
Dieser Lehrbrief darf in jeglicher Form ohne vorherige schriftliche Genehmigung der IU Internationale Hochschule GmbH nicht reproduziert und/oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.  
Die Autor:innen/Herausgeber:innen haben sich nach bestem Wissen und Gewissen bemüht, die Urheber:innen und Quellen der verwendeten Abbildungen zu bestimmen.  
Sollte es dennoch zu irrtümlichen Angaben gekommen sein, bitten wir um eine dement sprechende Nachricht.



## **WISSENSCHAFTLICHE KURSLEITUNG**

### **PROF. DR. MARIAN BENNER-WICKNER**

Herr Benner-Wickner promovierte an der Universität Duisburg-Essen im Fachgebiet Software Engineering. Neben verschiedenen Beratungstätigkeiten in der Energie-, Finanz- und Versicherungsbranche liegt sein beruflicher Schwerpunkt heute in der Weiterbildung von IT-Fachkräften. Als Trainer führt Herr Benner-Wickner seit vielen Jahren sowohl Online- als auch Präsenzseminare zu verschiedenen Themen aus der Softwaretechnik durch. In diesem Zusammenhang unterstützt er Unternehmen bei der Planung und Durchführung von Maßnahmen zur Weiterqualifikation ihrer IT-Mitarbeiter.

An der IU Internationale Hochschule übernimmt Herr Benner-Wickner eine Reihe von Modulverantwortungen im Bereich des Software-Engineerings mit dem Schwerpunkt auf mobile Anwendungen.

# INHALTSVERZEICHNIS

## PROGRAMMIERUNG VON WEBANWENDUNGSOBERFLÄCHEN

Wissenschaftliche Kursleitung .....	3
<b>Einleitung</b>	
Wegweiser durch das Studienskript .....	8
Basisliteratur .....	9
Weiterführende Literatur .....	10
Übergeordnete Lernziele .....	11
<b>Lektion 1</b>	
Architektonische Fundamente .....	13
1.1 Struktur und Geschichte des Internets .....	14
1.2 Internetprotokolle und URLs .....	15
1.3 Architektur von Webanwendungen .....	18
1.4 Aktuelle Trends .....	21
<b>Lektion 2</b>	
Werkzeuge der Webentwicklung .....	25
2.1 Entwicklungstools .....	26
2.2 Versionsverwaltung .....	27
2.3 Paketmanager .....	29
2.4 Upload/Bereitstellung .....	30
<b>Lektion 3</b>	
Entwicklung von statischen Webseiten .....	33
3.1 Grundlagen von HTML5 .....	34
3.2 Grundlagen von Cascading Style Sheets .....	43
<b>Lektion 4</b>	
Erweiterte Konstruktionstechniken .....	53
4.1 Responsives Web-Design .....	54
4.2 Seitenlayout .....	58
4.3 Media Queries .....	64
4.4 CSS-Frameworks .....	67

<b>Lektion 5</b>	
Webseitenentwicklung mit JavaScript	75
5.1 JavaScript-Geschichte, ES5/ES6 .....	76
5.2 JavaScript-Grundlagen .....	78
5.3 Verwendung von JSON .....	88
5.4 Gängige JavaScript-Frameworks .....	90
<b>Lektion 6</b>	
Testen und Sicherheit von Webanwendungen	95
6.1 Testen von Webanwendungen .....	96
6.2 Grundlegende Sicherheitskonzepte und -prinzipien .....	101
<b>Verzeichnisse</b>	
Literaturverzeichnis .....	108
Abbildungsverzeichnis .....	114



# EINLEITUNG

# **HERZLICH WILLKOMMEN**

## **WEGWEISER DURCH DAS STUDIENSKRIPT**

Dieses Studienskript bildet die Grundlage Ihres Kurses. Ergänzend zum Studienskript stehen Ihnen weitere Medien aus unserer Online-Bibliothek sowie Videos zur Verfügung, mit deren Hilfe Sie sich Ihren individuellen Lern-Mix zusammenstellen können. Auf diese Weise können Sie sich den Stoff in Ihrem eigenen Tempo aneignen und dabei auf lerntypspezifische Anforderungen Rücksicht nehmen.

Die Inhalte sind nach didaktischen Kriterien in Lektionen aufgeteilt, wobei jede Lektion aus mehreren Lernzyklen besteht. Jeder Lernzyklus enthält jeweils nur einen neuen inhaltlichen Schwerpunkt. So können Sie neuen Lernstoff schnell und effektiv zu Ihrem bereits vorhandenen Wissen hinzufügen.

In der IU Learn App befinden sich am Ende eines jeden Lernzyklus die Interactive Quizzes. Mithilfe dieser Fragen können Sie eigenständig und ohne jeden Druck überprüfen, ob Sie die neuen Inhalte schon verinnerlicht haben.

Sobald Sie eine Lektion komplett bearbeitet haben, können Sie Ihr Wissen auf der Lernplattform unter Beweis stellen. Über automatisch auswertbare Fragen erhalten Sie ein direktes Feedback zu Ihren Lernfortschritten. Die Wissenskontrolle gilt als bestanden, wenn Sie mindestens 80 % der Fragen richtig beantwortet haben. Sollte das einmal nicht auf Anhieb klappen, können Sie die Tests beliebig oft wiederholen.

Wenn Sie die Wissenskontrolle für sämtliche Lektionen gemeistert haben, führen Sie bitte die abschließende Evaluierung des Kurses durch.

Die IU Internationale Hochschule ist bestrebt, in ihren Skripten eine gendersensible und inklusive Sprache zu verwenden. Wir möchten jedoch hervorheben, dass auch in den Skripten, in denen das generische Maskulinum verwendet wird, immer Frauen und Männer, Inter- und Trans-Personen gemeint sind sowie auch jene, die sich keinem Geschlecht zuordnen wollen oder können.

# BASISLITERATUR

Libby, A, Gupta, G. & Talesra, A. (2016). *Responsive Web Design with HTML5 and CSS3 Essentials*. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=nlebk&AN=1343890&site=eds-live&scope=site>

Ferguson, R. (2019). *Beginning JavaScript. The Ultimate Guide to Modern JavaScript Development*. (3. Aufl.). Apress. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsebk&AN=2117465&site=eds-live&scope=site>

# WEITERFÜHRENDE LITERATUR

## LEKTION 1

Sunyaev, A. (2020). *Internet computing. Principles of Distributed Systems and Emerging Internet-Based Technologies*. Springer. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsdnl&AN=edsdbl.books.sp.Sunyaev20&site=eds-live&scope=site>

## LEKTION 2

Abildskov, J. (2020). *Practical Git. Confident Git Through Practice*. (1. Aufl.). Apress. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.036860289&site=eds-live&scope=site>

## LEKTION 3

Bühler, P., Sinner, D. & Schlaich, P. (2017). *HTML5 und CSS3. Semantik - Design – Responsive Layouts*. Springer. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.45827&site=eds-live&scope=site>

## LEKTION 4

Dease, N. (2021). Bridging the User Interface Gap: Bulma, Bootstrap, and Foundation. *Online Searcher*, 45(2), 20–24. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsgbe&AN=edsgcl.654815370&site=eds-live&scope=site>

## LEKTION 5

Flanagan, D. (2020). *JavaScript: the definitive guide: master the world's most-used programming language*. (7. Aufl.). O'Reilly. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.51155&site=eds-live&scope=site>

## LEKTION 6

Saleh, H. (2013). *JavaScript Unit Testing: Your Comprehensive and Practical Guide to Efficiently Performing and Automating JavaScript Unit Testing*. Packt Publishing. <http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsebk&AN=526921&site=eds-live&scope=site>

# ÜBERGEORDNETE LERNZIELE

Ziel des Kurses **Programmierung von Webanwendungsoberflächen** ist es, Sie in die Lage zu versetzen, einfache Webanwendungen mit etablierten Technologien zu programmieren. Zunächst erhalten Sie wichtige Einblicke in den typischen Aufbau aktueller Webanwendungsarchitekturen. Softwareentwicklung erfordert Versionsmanagement. Damit verbunden ist der Einsatz einschlägiger Tools zur Entwicklungs- und Quellcode-Verwaltung.

Sie beginnen mit der Hypertext Markup Language (HTML), um einfache Webseiten zu entwickeln. In Ergänzung werden Sie sich mit den wichtigsten und gängigsten Elementen des Cascading Stylesheet (CSS) Standards vertraut machen, um die in HTML geschriebenen Inhalte zu layouten. CSS ist im Gegensatz zu HTML sehr komplex, sodass Sie sich in echten Projekten häufig entsprechender Frameworks bedienen werden.

Um dynamische Webseiten zu realisieren, werden Sie sich mit den Grundfunktionen von JavaScript beschäftigen, um sich damit eine einfache Beispielanwendung zu bauen. Schließlich lernen Sie die Grundlagen des Testens und der Sicherheit von Webseiten kennen und können diese bewerten.



# LEKTION 1

## ARCHITEKTONISCHE FUNDAMENTE

### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Struktur und Geschichte des Internets wiederzugeben.
- unterschiedliche Internetprotokolle einzuordnen.
- zu erklären, wie eine URL aufgebaut ist.
- die Architektur von Webanwendungen zu verstehen.
- aktuelle Trends in der Entwicklung von Weboberflächen zu nennen.

# 1. ARCHITEKTONISCHE FUNDAMENTE

## Einführung

Wer heute einfach „ins Internet geht“ oder das Handy „online“ nutzt, macht das ganz selbstverständlich. Besonders jüngere Menschen kennen keine Zeit mehr vor „dem Internet“. Dahinter steckt aber eine Menge unterschiedliche Technologie, die sich über die Jahre entwickelt hat. Dabei war die Entwicklung nicht besonders zielgerichtet, sondern hat sich Schritt für Schritt eher zufällig ergeben. Wenn man heute über „das Internet“ spricht, meint man fast immer das „World Wide Web“ (WWW) und damit verbundene Dienste. Bis es so weit war, mussten einige Hindernisse überwunden werden.

Seine Anfänge hat das Internet in militärischer Technologie, die besonders ausfallsicher sein sollte. Tatsächlich profitiert das Netz auch heute noch von dieser Verlässlichkeit. Probleme an einer Stelle des Internets können oftmals durch „Umleitungen“ behoben werden, sodass allenfalls kleine Geschwindigkeitseinbußen spürbar werden.

Das Netz selbst ist allerdings mehr oder weniger nur ein „Transportmedium“ und wurde erst durch die damit möglichen Anwendungen populär. Davon war die Entwicklung des World Wide Web 1991 eine der ersten. Das WWW verbreitete sich zunächst langsam, dann aber immer schneller. Anwendungen, wie Suchmaschinen, Online-Shops und Wikipedia, sind aus dem heutigen Leben nicht mehr wegzudenken. Das Web bietet jedoch noch mehr Architekturen und Anwendungsmöglichkeiten, die hier in diesem Kapitel als Überblick vorgestellt werden.

### 1.1 Struktur und Geschichte des Internets

#### ARPA

Die Abkürzung steht für Advanced Research Projects Agency. Später wurde Defense ergänzt, sodass sich der Begriff zu DARPA erweiterte. Damit ist eine Forschungseinrichtung des amerikanischen Verteidigungsministeriums gemeint.

Das Internet als Netzwerk hat seinen Ursprung in den 1960er-Jahren. Zu diesem Zeitpunkt förderte die sog. **ARPA** (Teil des amerikanischen Verteidigungsministeriums) Forschung zu einer ausfallsicheren Kommunikationsinfrastruktur. Die Grundidee war, nicht mehr mit verbindungsorientierten, sondern mit paketvermittelten Diensten zu arbeiten. Das passte gut zusammen, denn damit konnten leichter „Umleitungen“ implementiert werden, wenn eine Leitung ausfiel.

Im nächsten Jahrzehnt wurde die Technologie zunehmend kommerziell interessant. Die Anbieter verstanden, dass sie mithilfe der Paketvermittlung eine wesentlich bessere Auslastung ihrer Infrastruktur erreichen konnten. Das führte zu einem starken Wachstum und einer explosiven Verbreitung der entsprechenden Technologie. Dadurch konnten ideale Voraussetzungen für das weitere Wachstum des Internets selbst geschaffen werden.

Die eigentliche Nutzung des Internets als Netzwerk begann in den 1980er-Jahren. Eine Bedingung dafür war die Verfügbarkeit von relativ günstigen Personal Computern, die z. B. durch Apple mit dem Apple II (Computer History, n. d.) und auch Commodore (C64) (com-

puterhistory.org, n. d.) zur Verfügung gestellt wurden. Neben der reinen Programmierung von Spielen wurden diese Geräte mehr und mehr zur Kommunikation untereinander genutzt – der Beginn der Vernetzung.

Parallel dazu wurden Verwaltungsstrukturen aufgebaut. Am bekanntesten ist die sog. Internet Engineering Taskforce (IETF), die hauptsächlich technologiegetrieben und bis heute kein offizielles Gremium ist – man kann nicht Mitglied werden, sondern einfach etwas beitragen. Das ist exemplarisch für die Entwicklung des Internets und stellt auch die Basis für die nächste entscheidende Entwicklung dar, das World Wide Web.

Das WWW wurde von Tim Berners-Lee am CERN im Jahr 1991 entwickelt. Obwohl es ursprünglich als Informationsplattform für Forschende gedacht war, unterschied es sich ganz erheblich von den bisherigen Anwendungen, die im damals schon existierenden Internet verwendet wurden. Diese waren nämlich größtenteils technisch und für Informatiker:innen oder andere Fachspezialist:innen gedacht. Protokolle, wie FTP (IETF, 1985) zum Datentransfer oder telnet (IETF, 1983) zur Fernsteuerung von Servern, waren für die Allgemeinheit wenig spannend. Durch die bunten Webseiten änderte sich das plötzlich und das Internet wurde immer mehr zum Breitenmedium. Suchmaschinen, wie Altavista (Submit Express, n. d.) und Katalog bzw. ein Verzeichnis, wie Yahoo, trugen immens zur Popularität bei. Entscheidend war aber auch die einfache Auszeichnungssprache HTML für Webseiten, die sich Berners-Lee ausgedacht hatte. Damit war es praktisch jeder Person möglich, selbst eine Webseite zu „bauen“.

Den ersten Webseiten ist auch anzusehen, dass es sich mehr um ein Hobby handelte. Mit dynamischen Webseiten, die durch JavaScript (Web Archive, 1995) realisierbar waren, wurde das Internet mehr und mehr kommerzialisiert. Die wirtschaftliche Bedeutung wurde vielen durch den Börsengang von Netscape deutlich, welches eine für damalige Verhältnisse unglaubliche Bewertung erhielt (und wie viele andere Unternehmen aus der Zeit auch wieder verschwand). Durch das zur Verfügung stehende Kapital konnten die Technologien weiterentwickelt werden, so folgte auf JavaScript die Cascading Stylesheets (MDN, n. d.-e), die eine deutlich bessere Gestaltung der Webseiten ermöglichten.

Spätestens mit dem Einzug des E-Commerce durch Amazon hat das Internet die Schwelle der Kommerzialisierung überwunden. Hardware, wie Laptops, aber insbesondere Smartphones und Tablets führten zu einer Demokratisierung, sodass jede:r Zugriff auf das Netz hatte. Unterstützt wurde diese Entwicklung durch immer schnellere und gleichzeitig günstigere Internet-Zugänge (DSL, Kabel und Glasfaser), die dank moderner und schneller Mobilfunktechnologien, wie LTE und 5G, auch mobiles Internet ermöglichen.

## 1.2 Internetprotokolle und URIs

Das Internet ist ein sog. IP-Netz, wobei IP für Internet Protocol (Cerf & Kahn, 1974) steht. Traditionell werden Netzwerkprotokolle in unterschiedliche Schichten unterteilt. Die am weitesten verbreitete Klassifikation ist das sog. ISO/OSI-Referenzmodell (Siegmund, 1992),

das bereits 1983 von der ITU veröffentlicht (ITU, 1983) und 1984 von der ISO (ISO, 1984) standardisiert wurde. Eine etwas vereinfachte Variante dieses Modells, das für die Praxis in der Regel völlig ausreicht, stellt das TCP/IP-Referenzmodell dar.

Abbildung 1: TCP/IP-Referenzmodell



Quelle: Christian Winkler, 2022.

Für diese Diskussion genügt es, das vereinfachte Modell aus der Abbildung zu betrachten. Der Netzzugriff erfolgt für die Endgeräte meistens über Ethernet (Roberts, 2006), WLAN (IEEE, 1997, S. 11) oder LTE (Web Archive, n. d.-a), die sowohl die Bits übertragen als auch für die Sicherung der Übertragung zuständig sind. Teilweise wird das IP-Protokoll bereits in den Endgeräten verwendet (siehe „2 Internet“ in der Abbildung). Hier spielt auch die Signalisierung („Host nicht erreichbar“) eine große Rolle. Die Daten fließen dann entweder direkt über LTE, DSL, Kabel oder Glasfaser weiter in die sog. **Internet-Backbones** der großen Anbieter. Der Datenaustausch zwischen diesen, das Peering, erfolgt meistens kostenlos – jeder Anbieter ist für seine eigene Infrastruktur verantwortlich.

#### Backbone

Dies sind die Hauptkanäle für die Kommunikation im Internet. Sie verbinden große Netzwerke miteinander.

#### TCP

Mit dem Begriff Transmission Control Protocol ist ein verbindungsorientiertes, Zustandsbehaftetes IP-Protokoll gemeint.

#### HTTP-Protokoll

Das Hypertext Transfer Protocol (HTTP) ist ein TCP-basiertes Protokoll zur Übertragung von HTML-Seiten und anderer Ressourcen.

In Schicht „3 Transport“ gibt es unterschiedliche Möglichkeiten für Protokolle, die häufig verwendet werden. Am bekanntesten ist das **TCP (Transmission Control Protocol)**, das verbindungsorientiert arbeitet. Es wird z. B. meistens zum Transfer von Webseiten eingesetzt. Als Alternative für andere Dienste wird häufig UDP (z. B. für DNS-Abfragen) verwendet, das verbindungslos arbeitet. Es ist zwar schneller, gibt aber im Gegensatz keine Quittungen – Daten können dabei verloren gehen.

Auf Anwendungsebene gibt es eine große Vielzahl von Protokollen. Das wohl Bekannteste und Meistgenutzte ist **HTTP** (W3C, n. d.-a), das Tim Berners-Lee konzipiert hat, um die HTML-Seiten zu transferieren.

HTTP und seine verschlüsselte Variante HTTPS sind im Moment sicherlich die am meisten verwendeten Protokolle im Internet. Ursprünglich war das Protokoll sehr einfach aufgebaut: Zunächst wird eine Anfrage (der sog. Request) gestellt, der eine **Methode** und eine URL (Berners-Lee et al., 2005) enthält. Die Antwort besteht aus zwei Teilen: einem Header mit Metadaten (Änderungsdatum, Größe, Übertragungsverfahren) und der angefragten Ressource, also eine Webseite, ein Bild, JavaScript usw.

#### HTTP-Methode

HTTP unterstützt verschiedene Methoden zur Übertragung der Daten, nämlich GET, POST PUT und DELETE.

Ein zentrales Element von HTTP ist die URL (Uniform Resource Locator). Die grundsätzliche Idee der URL ist, damit eindeutig jede Ressource im Internet adressieren und referenzieren zu können. Eine URL besteht aus unterschiedlichen Komponenten:

Abbildung 2: Struktur einer URL



Quelle: Christian Winkler, 2022.

Für Webanwendungen wird als Protokoll nur HTTP und HTTPS verwendet. Es gibt auch URLs (hauptsächlich zum Download), die **FTP** als Protokoll verwenden. Ab und zu wird „mailto“ verwendet, das ist aber kein Protokoll im engeren Sinne. Die Domain sagt, auf welchem Server die entsprechende Ressource zu finden ist. Der Name wird über das sog. **DNS** in eine IP-Adresse gewandelt. Der Pfad zeigt, unter welchem „Ordner“ die Ressource auf dem Server gefunden werden kann, während der Query-String zusätzliche Parameter übergeben kann (in diesem Fall einen Parameter mit dem Namen „search“ und dem Wert „web app“, vermutlich die Eingaben aus einem Formularfeld zu einer Textsuche). Manchmal soll ein Verweis nicht auf den Anfang der Seite erfolgen, sondern an eine bestimmte Stelle. Dafür dient der „Anchor“. Normalerweise wird der Anchor nicht an den Server übermittelt, sondern ist ein Hinweis für den Browser, zu welcher Stelle der Seite er scrollen soll.

#### FTP

Das File Transfer Protocol wird zur Übertragung großer Datenmengen verwendet. Weitestgehend wurde es heute durch das HTTP-Protokoll ersetzt.

#### DNS

Das Domain Name System dient im Internet der Zuordnung von Hostnamen zu IP-Adressen.

HTTP unterstützt unterschiedliche Methoden (manchmal auch HTTP-Verben genannt), womit URLs abgerufen werden können. Am bekanntesten ist die GET-Methode, mit der Ressourcen vom Server abgerufen werden können, z. B. die Homepage der IU. Sollen Ressourcen auf dem Server verändert werden (z. B. Setzen eines neuen Passwortes), muss die POST-Methode verwendet werden. Das neue Passwort wird dabei nicht, wie oben dargestellt, per Parameter in der URL, sondern im Header des Requests übermittelt. Wird ein neuer User angelegt, soll dafür die PUT-Methode verwendet werden. Zum Löschen von Ressourcen gibt es die DELETE-Methode.

HTTP ist ein (noch) komplizierteres Protokoll, das mit der neuen Version HTTP/2 auch Multiplexing und Websockets erlaubt. Mithilfe von Multiplexing können über eine bestehende Verbindung mehrere unterschiedliche Datenströme gleichzeitig übertragen werden, während Websockets es erlauben, Daten vom Server direkt zum Browser zu übertragen, und somit eine bidirektionale Kommunikation ermöglichen.

Hier können aus Platzgründen nur die wichtigsten Features angesprochen werden. Für Details sei auf die Literatur W3C (n. d.-a) verwiesen.

## 1.3 Architektur von Webanwendungen

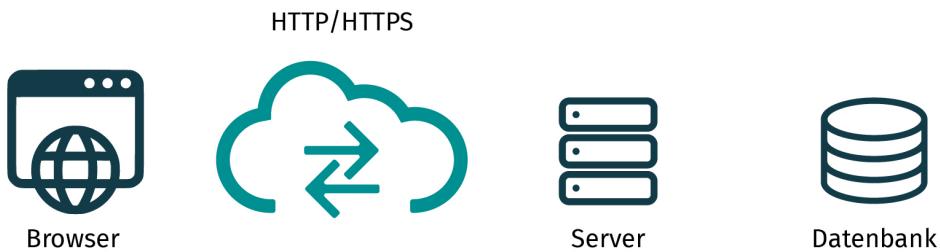
### Client-Server

In einem Client-Server-Szenario unterscheidet man zwischen einem Server (hier einem Webserver) und einem Client (hier einem Browser).

Das World Wide Web ist ein typisches **Client-Server**-Szenario. Zu Beginn gab es vor allem statische HTML-Seiten, die besonders an Universitäten und Hochschulen beliebt waren. Jeder konnte seine eigene Seite erstellen und sie der gesamten (damals noch kleinen) Internet-Welt zur Verfügung stellen.

Sehr schnell zeigte sich, dass auch dynamische Webseiten benötigt werden. Eine der ersten Anwendungen war SPIRES (SPIRES, n. d.), womit sich wissenschaftliche Artikel durchsuchen ließen. Mehr und mehr Intelligenz und Dynamik wurde in die Server verschoben, für die dynamische Darstellung verfügen fast alle Server über Datenbanken. Nachdem viele Operationen (wie Animationen) dort nur schwierig durchführbar sind, wurden auch die Clients (Browser) immer leistungsfähiger und programmierbar.

Abbildung 3: Interaktion zwischen Browser und Server



Quelle: Christian Winkler, 2022.

### Clients

#### Websites und Webseiten

Eine Web-Site kann aus vielen unterschiedlichen Webseiten bestehen und ist unter einer URL (mit Domain) im Internet erreichbar.

Fast alle Clients für **Websites** sind heute Web-Browser. Diese gibt es in unterschiedlichen Varianten und für unterschiedliche Plattformen. Die Clients sind für die Darstellung der Inhalte verantwortlich, während diese von Servern ausgeliefert werden. Server können dabei dynamische Inhalte liefern (wie z. B. bei einem Online-Shop), sodass eine dynamische Darstellung der Webseiten durch die Clients mithilfe von JavaScript realisierbar wird.

Bei Computern ist der Chrome-Browser von Google zurzeit der beliebteste Client, weil er häufig aktualisiert wird und sehr schnell arbeitet. Für Windows-Betriebssysteme liefert Microsoft den Edge-Browser mit, der in erheblichen Teilen auf Chrome basiert bzw. die gleiche **Browser-Engine** verwendet. Das macht es für Web-Entwickler etwas einfacher, weil die Kompatibilität der Seiten dann nicht separat überprüft werden muss. Unter MacOS liefert Apple den Safari-Browser, der ein etwas anderes Engine (nämlich Webkit

#### Browser-Engine

Die Browser-Engine ist für die Interpretation der HTML-Seiten im Browser und deren Darstellung mit CSS verantwortlich.

[MARC, n. d.]) verwendet. Firefox ist auch auf allen Plattformen verfügbar und setzt das Servo-Engine ein (Servo, n. d.). Alle anderen Browser mit nicht zu vernachlässigenden Marktanteilen nutzen auch die oben genannten Engines.

In der mobilen Welt sieht es etwas einfacher aus. Dort gibt es im wesentlichen Chrome (für Android) und Safari (für iOS). Dabei spielt es keine große Rolle, ob Smartphones oder Tablets verwendet werden. Die Darstellung kann an mobile Anforderungen angepasst werden. Das geschieht über CSS-Regeln, die später noch genauer erklärt werden. Ein großer Unterschied besteht vor allem in der Interaktion mit dem Browser, weil der sog. **Mouseover-Effekt** auf den Touch-Geräten nicht umgesetzt werden kann.

Alle gängigen Browser können heute mit HTML 5, JavaScript und CSS 3 als zentrale Internet-Technologien, die später vorgestellt werden, umgehen. Die noch relativ neue Technologie, WebAssembly, die sehr viel schneller ist als JavaScript und ganz andere Anwendungen erlaubt, ist noch nicht sehr weit verbreitet. Das wird sich aber auf absehbare Zeit ändern.

Schließlich gibt es nicht nur Browser als Clients. Viele Apps oder auch Programme auf Computern nutzen auch Web-Services, um mit Servern zu kommunizieren. Die Darstellung erfolgt dann eventuell anders. Überraschenderweise sind aber auch Programme, wie Visual Studio Code, eigentlich Web-Browser, die mit den gleichen Methoden programmiert werden, aber trotzdem wie ein „normales“ Computerprogramm funktionieren.

## Server

Auch der Markt für Webserver hat sich sehr stark vereinheitlicht. Aus sehr viel unterschiedlicher Software, die in der Anfangszeit zur Verfügung stand, sind nur wenige große Player übriggeblieben. Neben Apache (Apache Software Foundation, n. d.), dem Webserver und ersten Produkt der Apache Software Foundation, spielt heute vor allem nginx (nginx, n. d.) eine sehr große Rolle. Beide Server laufen auf Linux-Betriebssystemen und haben zusammen mit dem Internet Information Server einen Marktanteil von etwa 80 % (Hostadvice, 2022) – der Rest sind Nischenanbieter. Ergänzt werden Webserver durch sog. Application-Server, die sich um die serverseitige Erzeugung von dynamischen Webseiten kümmern.

Meistens stellen die Webserver nur den statischen Content zur Verfügung und sorgen für das sog. **Load Balancing**. Die dynamisch produzierten Seiten (z. B. eines Online-Shops) werden mit spezieller Software erzeugt, die häufig in PHP (PHP, n. d.), Java (Oracle, n. d.) oder Python (Python, n. d.) geschrieben ist. Sehr häufig ist eine Datenbank daran beteiligt, oft MySQL (MySQL, n. d.). Bei besonders beliebten Webseiten bietet sich die Nutzung eines sog. **Content-Distribution-Networks** (CDN) an. Besonders häufig angefragte Ressourcen werden dann dezentral ausgeliefert und führen zu einem wesentlich schnelleren Seitenaufbau. Eigener Content lässt sich bei einem Webspace-Anbieter für sehr geringe Monatsbeiträge hosten. Noch einfacher geht es mit Github, das ein kostenloses Hosting ermöglicht (Github, n. d.-c.).

**Mouseover-Effekt**  
Häufig passiert es bei Webseiten, dass bei der Berührung mit der Maus bestimmte Aktionen (Hervorhebung etc.) ausgelöst werden, die gesammelt als Mouseover-Effekt bezeichnet werden.

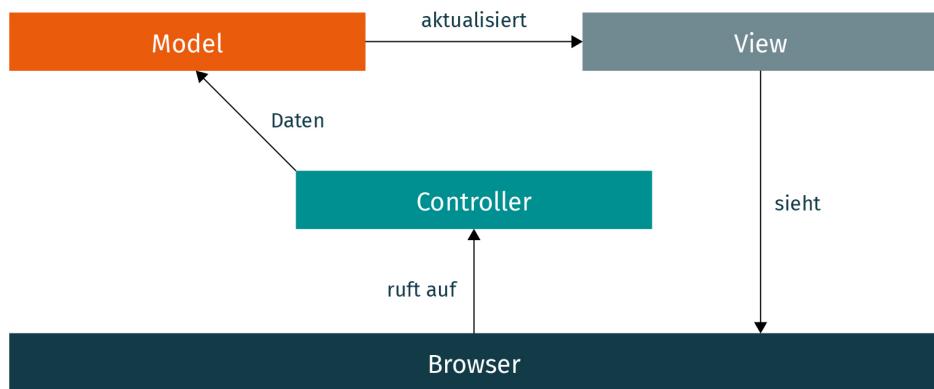
**Load Balancing**  
Um die Überlastung oder den Ausfall eines Webserver zu verhindern, wird ein Angebot durch zwei Server gehostet, die sich abwechseln.  
**Content-Distribution-Networks**  
Ein CDN dient der Verteilung von Inhalten und kann zur Beschleunigung von Webanwendungen eingesetzt werden.

## MVC-Muster

Browser ermöglichen komplexe, clientseitige Programmierung. Auch die serverseitige Programmierung erlaubt sehr viele Freiheiten. Um die Verantwortlichkeiten zu verteilen und Applikationen sauber zu strukturieren, können Entwurfsmuster verwendet werden.

Besonders beliebt bei serverseitigen Web-Applikationen ist das sog. Model-View-Controller-Muster (MVC), das die Komponenten strukturiert ordnet.

Abbildung 4: Model-View-Controller Muster



Quelle: Christian Winkler, 2022.

Der Einsprungpunkt für den Browser ist dabei der Controller, eine Komponente in der serverseitigen Web-Applikation, die z. B. als Klasse realisiert sein kann. Abhängig von der aufgerufenen URL entscheidet die Routing-Komponente (hier im Bild nicht dargestellt), welche Methode im Controller aufgerufen werden soll. Wenn notwendig, kann der Controller Daten aus einem Modell laden, die z. B. aus einer Datenbank kommen. Welche Daten das sind, kann der Controller an der URL ablesen. Anschließend werden die Modelldaten an den View weitergegeben, der eine Art Schablone für HTML-Seiten darstellt. Dabei kann der View berücksichtigen, dass sich bestimmte Elemente einer Seite (wie die Navigation) immer wiederholen und nur einmal abgelegt sein müssen (sog. **Templating**). Der View erzeugt eine fertige HTML-Seite und liefert diese an den Browser aus. Damit ist die Arbeit des Browsers noch nicht fertiggestellt, weil Inhalte, wie Stylesheets, JavaScript, CSS und Bilder, noch nachgeladen werden müssen. Der Browser verfügt nun aber über alle notwendigen Informationen, diese Ressourcen vom Server anzufordern. In den letzten Jahren konnte dieser Prozess erheblich beschleunigt werden und führt damit zu einem schnelleren Seitenaufbau. Grundvoraussetzung dafür ist aber, dass die HTML-Seite vom Server schnell ausgeliefert werden kann. Klickt der Nutzende auf einen Link, der auf die gleiche Website verweist, beginnt der Zyklus von Neuem.

### Templating

Die Nutzung von Schablonen für Seiten, um Wiederholungen zu vermeiden, nennt man Templating.

## 1.4 Aktuelle Trends

Das Web ist immer in Bewegung. Daran hat sich auch in den letzten 30 Jahren nichts geändert – ständig gibt es neue Geschäftsideen, aber auch neue technische Realisierungen.

### Single Page Applications

Bei traditionellen Websites erfolgt die Navigation von Seite zu Seite. Wird ein Link angeklickt, wird eine neue Anfrage vom Browser zum Webserver gestellt, der eine neue HTML-Seite zurückliefert. Anschließend lädt der Browser alle damit verbundenen Ressourcen und baut die Seite auf.

Weil sich aufeinanderfolgende Seiten häufig gleichen, ist dieses Vorgehen nicht besonders effizient. Der Browser hat viel Arbeit, ein fast identisches Seitenlayout aufzubauen, in dem sich eventuell nur Kleinigkeiten geändert haben, der Großteil aber unverändert bleibt. Erschwerend kommt hinzu, dass der Zustand des Browsers immer mit dem des Servers abgeglichen werden muss, damit der Server die richtigen Informationen senden kann.

Bei Single Page Applications wird das anders gelöst. Es wird immer „nur“ der veränderte Content ausgetauscht, das Fragment der Seite bleibt bestehen. Dies führt zu einer wesentlich schnelleren Darstellung und viel geringerem Datentransfer. Weil technisch eine Seite erhalten bleibt, kann dort auch der Zustand im Browser gespeichert bleiben. **Gmail** war die erste echte Single Page Application. Heute gibt es sehr viele dieser Art. Fast alle Web-Applikationen, die sich wie „richtige“ Programme verhalten, sind als Single Page Application implementiert.

**Gmail**  
Der Mail-Dienst von Google war die erste Webanwendung, die sich wie eine echte Applikation angefühlt hat.

### Progressive Web Applications

Rufen Nutzende auf ihren mobilen Geräten Webseiten auf, so möchten sie die Anbieter gerne dazu bewegen, möglichst immer wieder zu kommen, ohne umständlich die URL eingeben zu müssen. Da auch die Behandlung von Bookmarks auf Mobilgeräten nicht ganz einfach ist, sollten die Websites am besten zu **Apps** werden. Ganz so funktioniert das zwar nicht, aber mithilfe von Progressive Web Apps können Anwendende die Websites zu ihren Apps (bzw. zu ihrem Startbildschirm) hinzufügen. Optisch sieht das identisch zu den nativen Apps aus und der gewünschte Kundenbindungseffekt kann damit erreicht werden.

**Web-Apps**  
Die sogenannten Web-Apps basieren auf HTML-Seiten. Man sollte sie von Apps unterscheiden, die auf Handys installiert werden und mit anderen Technologien realisiert sein können.

Durch die Nutzung von bestimmten Programmierungsmöglichkeiten können die Web-Applikationen auch offline funktionieren, d. h. ohne Kontakt zum Server starten. Auch Aktualisierungsmechanismen sind möglich. Falls das Mobilgerät gerade online ist und eine neue Web-App-Version bereitsteht, kann diese aktualisiert werden. Dieses Hybrid-Modell ist zwar etwas schwieriger zu programmieren, aus Sicht der Kundenbindung und des nicht vorhandenen Medienbruchs aber optimal.

## Alles im Browser

Immer mehr Anwendungen laufen heutzutage im Browser. Anfang 2004 war es eine Sensation, als Google seinen E-Mail-Dienst Gmail als Browser-Anwendung implementiert hat, die sich wie eine echte Applikation angefühlt hat. Das war damals revolutionär – wenn auch der dazu notwendige Aufwand sehr erheblich war.

Die technischen Möglichkeiten sind heute weit größer, weil sich insbesondere die Browser, aber auch die Protokolle weiterentwickelt haben. Anwendungen, wie Google Docs, erlauben schon eine lange Zeit das kollaborative Arbeiten an Dokumenten, auch Office von Microsoft steht in einer browserbasierten Variante zur Verfügung.

### Plattform

Eine Plattform bringt viele Basisfunktionalitäten mit, die dann für die einzelnen Anwendungen nicht mehr separat implementiert werden müssen.

Diese Entwicklung, den Browser als **Integrationsplattform** zu verwenden, wird sich in den nächsten Jahren noch beschleunigen. Entscheidend dafür ist erneut eine ganz bestimmte Technologie, das sog. WebAssembly. Dabei handelt es sich um eine übersetzte Sprache, die im Browser abgearbeitet werden kann, dabei jedoch fast so schnell ist wie herkömmliche Programme. Nachdem Compiler für die üblichen Programmiersprachen zur Verfügung stehen, können viele vorhandene Programme in WebAssembly übersetzt werden. Außerdem wird im Gegensatz zu herkömmlichen Web-Applikationen nicht der (wenn auch häufig verschleierte) Quellcode mit ausgeliefert – es ist damit deutlich schwieriger eine Applikation zu kopieren. Selbst große Applikationen, wie LibreOffice, wurden schon in WebAssembly übersetzt (Allotropia, n. d.) und laufen dann im Browser wie eine echte Anwendung. Es ist spannend, zu beobachten, was mit WebAssembly alles bevorsteht.

## JavaScript auf dem Server

Etwas umgekehrt zu der oben angesprochenen Entwicklung von WebAssembly gibt es die Tendenz, mit möglichst wenig Programmiersprachen in einer Web-Applikation zu arbeiten. Da JavaScript als Programmiersprache im Browser nicht einfach geändert werden kann (bzw. konnte, durch WebAssembly ist es nun möglich) führte der Weg zunächst dazu, auch auf der Serverseite JavaScript als Programmiersprache zu verwenden.

### Serverseitige Programmierung

Auch Server können dynamische Webseiten ausliefern, wie z. B. in Online-Shops. Dafür muss serverseitige Programmierung eingesetzt werden.

Dazu wurde JavaScript um zusätzliche Funktionen ergänzt, die für die **serverseitige Programmierung** notwendig und sinnvoll sind. Das dazugehörige Framework heißt node.js (Node JS, n. d.), darum hat sich in der Zwischenzeit ein gesamtes Ökosystem von Tausenden Bibliotheken und Paketen gebildet. Diese bauen teilweise aufeinander auf und benötigen Basissoftware in einer bestimmten Version. Da diese Abhängigkeiten nicht immer leicht zu lösen sind, gibt es dazu eigene Software, sog. Paketmanager.

Ob es tatsächlich sinnvoll ist, komplexe serverbasierte Anwendungen in einer dynamisch (und schwach) typisierten Sprache, wie JavaScript, zu entwickeln, muss von Fall zu Fall entschieden werden. Eine Google-Trend-Abfrage nach node.js scheint den Eindruck zu vermitteln, dass die Technologie ihren Zenit bereits überschritten hat.



## ZUSAMMENFASSUNG

Auch 30 Jahre nach seiner Erfindung ist das Web ein dynamisches Medium, was sich ständig weiterentwickelt. Ursprünglich für den wissenschaftlichen Austausch gedacht, hat es mehr und mehr Domänen teilweise stürmisch erobert. Dabei ist auch die technische Komplexität gewachsen. Heutige Webanwendungen sind nicht mehr mit denen zu Beginn der Entwicklung vergleichbar und werden heute mit modernen Entwurfsmustern, wie MVC, gestaltet.

Browser wurden im Laufe der Zeit zu immer mächtigeren Instrumenten. Dazu hat besonders die Programmierbarkeit mit JavaScript beigetragen. Aus der ursprünglichen Idee der Browser, lediglich die Darstellung der Webseiten zu übernehmen, sind in der Zwischenzeit umfangreiche Plattformen geworden, auf denen komplexe Applikationen laufen können. Selbst für Programme, wie Word oder Excel gibt es heute browserbasierte Varianten, die fast alle Funktionalitäten ihrer nativen Pendants beherrschen und auf jedem Browser ausgeführt werden können.

Neben der Komplexität ist außerdem zu beobachten, dass das Web auf immer mehr Endgeräten verfügbar ist. Egal ob es Handys, Tablets oder Fernseher sind, fast überall steht ein Browser zur Verfügung. Viele „echte“ Apps werden im Hintergrund auch mit Web-Technologie entwickelt, weil sie dann mit geringem Aufwand auch im Browser zur Verfügung stehen, wenn jemand keine App installieren möchte. Spannend sind auch die Überlegungen, die dritte Dimension mit in Webanwendungen zu übernehmen und diese auf VR- oder AR-Brillen verfügbar zu machen.

Aus der Web-Entwicklung ist somit eine Disziplin der Software-Entwicklung geworden. Deswegen werden zunehmend auch die Methoden des Software-Engineerings auf Web-Entwicklung angewendet und das gesamte Thema professionalisiert sich.



# LEKTION 2

## WERKZEUGE DER WEBENTWICKLUNG

### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- mit relevanten IDEs zur Entwicklung von Webseiten umzugehen.
- mit dem Quellcode-Verwaltungssystem „git“ die wesentlichen Aufgaben zu erledigen.
- zu verstehen, was ein „Paketmanager“ als Software leistet.
- Web-Apps zu veröffentlichen.

## 2. WERKZEUGE DER WEBENTWICKLUNG

### Einführung

Webseiten und -applikationen lassen sich mit relativ wenigen Hilfsmitteln entwickeln. Die Ursache liegt darin, dass Anfang der 1990er-Jahre noch gar nicht viele solcher Tools zur Verfügung standen. Fast alles wurde damals mit einfachen Textbearbeitungsprogrammen (Editoren) eingegeben. Prinzipiell ist das auch heute noch möglich, denn die für Webseiten notwendigen Dateien (HTML, CSS und JavaScript) sind allesamt einfache Textdateien, die nicht kompiliert werden müssen. Mit den heutigen Mitteln ist das aber relativ unkomfortabel. Auf nahezu jedem Computer gibt es Software, mit dem sich die Entwicklung viel einfacher erledigen lässt. Die meisten davon sind sogar kostenlos und betriebssystemunabhängig verfügbar.

Die Entwicklung begann mit wissenschaftlichen Websites, rief dann aber viele Hobbyisten auf den Plan. Zwischenzeitlich hat sich daraus eine ganze Industrie entwickelt. Websites sind essenziell für fast alle Unternehmen und Online-Shops eigene Geschäftsmodelle, die Firmen, wie Amazon, erst ermöglicht haben. Es gibt nicht nur kommerzielle Angebote, sondern auch komplexe Applikationen in der öffentlichen Verwaltung. Daran arbeiten immer Teams aus Entwickler:innen, die mithilfe von Versionsverwaltungen weitestgehend unabhängig voneinander den Code verändern können, ohne sich gegenseitig zu stören.

Heutige Websites basieren auf sehr vielen unterschiedlichen Komponenten, die oft in komplexer Art voneinander abhängen. So ist z. B. für ein genutztes Layout eine bestimmte Javascript-Bibliothek erforderlich. Solche Abhängigkeiten können automatisch aufgelöst werden, sodass die entstehenden Webseiten dann funktionsfähig auf Webservern im Internet bereitgestellt werden können.

### 2.1 Entwicklungstools

Die Entwicklung von Webseiten kann auch heutzutage weiterhin mit einem Texteditor erfolgen. Besonders zweckmäßig ist das nicht, weil es zwischenzeitlich sehr leistungsfähige Software gibt, die dabei wesentliche Unterstützung leistet. Tatsächlich ist es eher schwierig, in der Menge der verfügbaren Programme das richtige zu finden.

#### Entwicklungsumgebungen

Damit ist ein Programm gemeint, mit dessen Hilfe man z. B. Webseiten komfortabler entwickeln kann. Es gibt aber auch Entwicklungsumgebungen für andere Programmiersprachen, wie Java etc.

Die dafür geeigneten Programme nennen sich **Entwicklungsumgebungen**. Diese gibt es für nahezu alle Programmiersprachen. Web-Entwicklung stellt eine besondere Herausforderung dar, da durch die Kombination von HTML, CSS und JavaScript immer gleich mehrere Technologien dabei berücksichtigt werden müssen. Dazu kommt noch eine weitere Schwierigkeit: Web-Applikationen werden als Client-/Server-Software entwickelt, Fehler können also sowohl auf dem Client als auch auf dem Server auftreten.

Die gängigen Entwicklungsumgebungen haben sich darauf eingestellt und können all diese Probleme elegant lösen. Dabei gibt es unterschiedliche Alternativen, die zum Einsatz gebracht werden können:

- **Visual Studio Code (VS Code)** (Visual Studio Code, n. d.-b)

Diese Entwicklungsumgebung wurde von Microsoft implementiert. Sie basiert selbst auf dem Browser-Engine von Chromium und ist daher plattformübergreifend verfügbar. Es gibt jede Menge Plugins, mit der sich die IDE ergänzen lässt. VS Code ist plattformunabhängig kostenlos verfügbar.

- **WebStorm** (JetBrains, n. d.)

WebStorm ist eine kommerzielle Entwicklungsumgebung der tschechischen Firma JetBrains, die auch die populäre Java-IDE IntelliJ im Programm hat. WebStorm ist sehr leistungsfähig und kann für private Zwecke kostenlos verwendet werden.

- **Online-Editoren**

Einige Editoren können auch ohne Installation online verwendet werden. Dazu gehört z. B. Visual Studio Code Web (Visual Studio Code, n. d.-a) (eine Web-Variante von Visual Studio Code), Phoenix und auch der in Github integrierte Editor (der wiederum auf Visual Studio Code beruht).

Die Auswahl einer IDE ist hauptsächlich Geschmackssache. Da Visual Studio Code plattformübergreifend frei verfügbar ist und in der Popularität noch stark wächst, wird er in diesem Kurs zum Einsatz kommen. Alle vorgestellten Beispiele funktionieren aber auch mit jeder anderen IDE.

Visual Studio Code kann man sich direkt von Microsoft herunterladen (Visual Studio Code, n. d.-b). Alternativ ist die Nutzung der völlig offenen Umgebung VSCode möglich (VSCode, 2022). Visual Studio Code verfügt über ein Plugin-System. In einzelnen Phasen des Kurses sollten Plugins installiert werden, um komfortabler mit der IDE arbeiten zu können.

## 2.2 Versionsverwaltung

Eine einfache Website lässt sich allein aufbauen. Für komplexe Webapplikationen wird fast immer ein Entwicklerteam engagiert. Dadurch entstehen einige Herausforderungen.

1. Wie synchronisiert man den Code, wenn mehrere Entwickelnde an den gleichen Dateien arbeiten?
2. Wie kann man Änderungen im Notfall rückgängig machen? **Revisionssicherheit** hilft auch einem einzelnen Entwickelnden.
3. Wie kann man bereits implementierte Features wieder entfernen, ohne das Gesamt- system zu beschädigen?
4. Wie kann man parallel an Features arbeiten, die zu unterschiedlichen Zeiten in das Online-System überführt werden sollen?

**Revisionssicherheit**  
Der Begriff Revisionssicherheit bezeichnet die Möglichkeit, jederzeit auf jeden alten Versionsstand der Software zurückgreifen zu können.

Entwickelnde großer Programme kennen diese Problematik seit Jahrzehnten. Aus RCS (GNU, n. d.) entstand so das verteilte **Versionskontrollsystem CVS** (CVS, n. d.), das in den 2000er-Jahren durch Subversion (Apache Subversion, n. d.) abgelöst wurde. Eines der

**Versionskontrollsystem**  
Ein Versionskontrollsyste ermöglicht es, mit unterschiedlichen Versionen von Dateien zu arbeiten.

größten Open Source-Projekte, der Linux Kernel, hatte lange Zeit kein Versionskontrollsystem im Einsatz, weil dem Linux-Erfinder Linus Torvalds keines davon geeignet erschien. Seine Anforderungen übersetzte er direkt in Software – dabei entstand ein neues Versionskontrollsystem, was sich zwischenzeitlich als weltweiter Standard etabliert hat: git (git SCM, n. d.). Besonders durch die Kollaborationsplattform Github (Github, n. d.-c) hat es immense Popularität erreicht.

Tatsächlich verhält sich git anders als die bis dahin üblichen Versionskontrollsysteme. Im Unterschied dazu benötigt git keinen Server, wenn man es in der Basisfunktionalität für sich allein nutzen möchte. Möchte man mit anderen zusammenarbeiten, ist es möglich, z. B. Github als Austauschplattform zu nutzen. Man kann git mit einem Befehl mitteilen, dass es die Kontrolle eines gesamten Verzeichnisses übernehmen soll. Daraufhin markiert es alle unbekannten Dateien, welche anschließend manuell hinzugefügt werden müssen. Jede:r Entwickler:in verfügt bei git immer über eine Kopie des gesamten Quellcodes, die sog. **Working Copy**.

**Working Copy**  
Bei git arbeitet jede:r Entwickelnde mit einer eigenen Arbeitskopie, die unabhängig von der der anderen ist.

Die grundlegendsten Operationen von git sind hier vereinfacht beschrieben:

- **init**

Hiermit wird ein Repository angelegt. Das aktuelle Verzeichnis steht ab dann unter Versionskontrolle.

- **add**

Ein Repository enthält zunächst keine Dateien, diese müssen einzeln hinzugefügt werden.

- **commit**

Um eine Änderung in die git-Historie zu übernehmen, muss sie „committed“ werden. Ein „commit“ hat immer eine Erklärung dabei, damit man später noch weiß, was man (und warum) gemacht hat.

- **status**

Damit kann man sich anzeigen lassen, welche Dateien schon „committed“ sind, welche verändert wurden und von welchen git gar nichts weiß.

- **log**

Dieser Befehl dient zum Anzeigen der Historie aller Änderungen.

## Datenaustausch mit Remote-Repositories

Um miteinander kollaborieren zu können, wird ein Remote-Repository benötigt, über den der Austausch von Code bzw. Dateien stattfindet. Das kann z. B. bei Github liegen, es ist aber ebenso möglich, eigene git-Server zu betreiben. Die wesentlichen Operationen, um mit Remote Repositories zu arbeiten, sind wie folgt vereinfacht beschrieben:

- **clone**

Damit wird ein Remote-Repository initial heruntergeladen.

- **push**

Die lokalen Änderungen werden einem Remote-Repository zur Verfügung gestellt.

- **pull**

Änderungen an einem Remote-Repository werden lokal integriert.

Git ist noch weit umfangreicher, aber für diesen Kurs genügen die oben genannten Funktionen. Glücklicherweise ist git hervorragend in Visual Studio Code integriert, sodass man alle Operationen mit einem Mausklick durchführen kann. Git ist für die allermeisten gängigen Betriebssysteme verfügbar und kann unter git SCM (n. d.) installiert werden.

## 2.3 Paketmanager

Moderne Applikationen werden immer komplexer. Es entstehen Abhängigkeiten zwischen unterschiedlichen JavaScript-Frameworks, CSS-Frameworks und den Klassen, die in den Webseiten auch wirklich verwendet werden. Um dem Browser die Arbeit zu erleichtern, möchte man möglichst nur den Code ausliefern, der auch verwendet wird.

Dazu wird wie in traditionellen Applikationen auch bei Websites ein sog. **Build-Prozess** verwendet. Dieser ist teilweise schon in Frameworks integriert, wie z. B. in Tailwind (Tailwind CSS, n. d.). Andere Frameworks, wie Vue.js (Vue.js, n. d.) oder React (React, n. d.-a), unterstützen verschiedene Alternativen, die dann separat aufgesetzt werden müssen.

**Build-Prozess**  
Software, die aus unterschiedlichen Komponenten besteht, muss vor der Verwendung zusammengebaut werden.

Die meisten echten Web-Applikationen greifen auf viele unterschiedliche, bereits existierende JavaScript-Bibliotheken und CSS-Frameworks zurück. Auch diese liegen in verschiedenen Versionen vor. Um Abhängigkeiten richtig aufzulösen, wird die Applikation häufig mit einem „Paketmanager“ gebaut. Dafür gibt es unterschiedliche Alternativen:

1. **Npm (Node Package Manager)** (npm, n. d.)

Hierbei handelt es sich um den Paketmanager von node.js, dem ersten und immer noch bekanntesten serverseitigen JavaScript-Engine.

Npm arbeitet mit vielen Frameworks zusammen und ist der Standard-Paketmanager von node.js.

2. **Yarn (Yet Another Resource Negotiator)** (Yarn, n. d.)

Yarn wurde von Facebook als Ersatz für npm konzipiert und überwindet einige der Schwächen von npm. Großen Wert legt yarn auf Geschwindigkeit, Verlässlichkeit und Sicherheit. Yarn hat einen lokalen Cache von Paketen und installiert diese parallel. Daher ist es meist viel schneller als npm.

Für einfache Webanwendungen ist kein Paketmanager notwendig. Verlangt ein Framework einen Paketmanager, sollte man den nutzen, der dort schon vorgesehen ist. Eigentlich ist das erst dann sinnvoll, wenn es komplexer wird und Abhängigkeiten aufgelöst werden müssen. Dann sollte auch nur der Teil in der Webapplikation enthalten sein, der wirklich gebraucht wird. Dazu wird ein sog. Packer benötigt, für den es auch unterschiedliche Alternativen gibt:

1. **Webpack** (webpack, n. d.)

Webpack kann entscheiden, welche Dateien in einer Applikation alle benötigt werden und diese nach Wunsch umformen (um z. B. TypeScript [Turner, 2014] in JavaScript zu übersetzen). Webpack lässt sich auch interaktiv benutzen und hat dafür einen Webserver integriert. Änderungen werden so automatisch erkannt und lösen im Browser einen Reload aus.

2. **Gulp** (gulp.js, n. d.)

Gulp arbeitet ähnlich wie webpack, ist aber durch JavaScript-Code selbst noch deutlich konfigurierbarer. Plugins können besonders einfach implementiert werden.

3. **Babel** (BabelJS, n. d.)

Hierbei handelt es sich um einen sog. Transpiler, mit dem neuere JavaScript-Varianten in solche übersetzt werden können, die von allen Browsern verstanden werden. Auch „Mischungen“ von JavaScript und HTML (**JSX**) kann babel konvertieren.

4. **Parcel** (Parcel, n. d.)

Parcel hat in letzter Zeit enorm an Popularität gewonnen, weil es die Arbeit an sog. Worker-Prozesse delegiert und damit Mehrprozessorsysteme sehr effizient nutzt. Durch die Integration eines Caches wird der Gesamtprozess dadurch wesentlich schneller.

#### JSX

Häufig hängt HTML mit CSS und JavaScript eng zusammen. JSX erlaubt deren gemeinsame Speicherung in einer Datei als Komponente.

## 2.4 Upload/Bereitstellung

Das Web ist eine Client-Server-Technologie. HTML-, CSS- und JavaScript-Dateien werden auf einem Server gespeichert und von dem Client (Browser) angefragt. Der Server muss mindestens HTTP als Protokoll anbieten, besser sogar die verschlüsselte Variante HTTPS (um z. B. geheime Daten, wie Kreditkartennummern, gesichert zu übertragen). Dafür braucht man allerdings Zertifikate, die man z. B. von LetsEncrypt.org kostenlos beziehen kann. Daher setzt sich HTTPS als Protokoll immer mehr durch.

Am einfachsten kann man Webspace (auf dem die Dateien abgelegt werden können) über einen externen Dienst im Internet mit einer eigenen Domain bei einem der zahlreichen Anbieter buchen. Auch Cloud-Dienste können dafür zum Einsatz kommen. Für viele Open Source-Projekte oder Übungen eignet sich auch Github mit den sog. Github Pages (Github, n. d.-b).

Die Übertragung der Daten von dem eigenen Computer auf den Server nennt sich Upload. Vor dem Upload müssen die Daten vorbereitet werden, z. B. muss der Build-Prozess angestoßen werden. Auch dafür gibt es Automatisierungslösungen (**Continuous Integration**), wie Jenkins (Jenkins, n. d.) oder Github Actions (Github, n. d.-a). Die damit verbundenen Prozesse werden häufig als **Deployment** bezeichnet. Es gibt sehr unterschiedliche Varianten, wie die Dateien auf den Webserver übertragen werden – das hängt stark vom Anbieter ab.

Für die Übungen im Kurs wird ein integrierter Webserver von Visual Studio Code verwendet, der Änderungen erkennt und den Browser automatisch informiert. Das ist besonders komfortabel für die Entwicklung, weil dann nicht nach jeder Änderung die Seite manuell neu geladen werden muss (Visual Studio, n. d.).

**Continuous Integration**  
Wenn Software fortlaufend (d. h. bei jeder Änderung) gebaut wird, nennt sich das Continuous Integration.

**Deployment**  
Darunter versteht man die Installation von Software (wie auch Webseiten) auf einem Server.



## ZUSAMMENFASSUNG

Es gibt eine Vielzahl von Entwicklungstools, mit denen Webseiten erstellt werden können. Technisch sind diese gar nicht unbedingt notwendig, denn durch das textbasierte HTML-Format könnte alles auch direkt mit einem einfachen Texteditor entwickelt werden. Das ist heute nur noch in den wenigsten Fällen praktikabel.

Stattdessen kommen Entwicklungsumgebungen zum Einsatz, die viele Routineaufgaben deutlich erleichtern und sehr gut in die Webentwicklung integriert werden können. Es gibt kommerzielle Tools, wie z. B. Webstorm, aber auch viel frei verfügbare Software. In diesem Kurs wird das äußerst beliebte Visual Studio Code von Microsoft eingesetzt, das sich sehr komfortabel bedienen und über Plugins erweitern lässt – dabei ist es kostenlos und frei verfügbar.

Durch die Größe und Komplexität der Webanwendungen werden diese heute nur noch selten von Einzelpersonen entwickelt – fast immer sind ganze Teams beteiligt. Um sich bei Änderungen nicht zu stören und die Revisionssicherheit zu gewährleisten, erfordert diese Zusammenarbeit ein Konfigurationsmanagement. In dem Bereich hat sich git als Tool durchgesetzt und wird auch von fast allen großen Projekten in der Softwareentwicklung eingesetzt. Git lässt sich hervorragend in Visual Studio Code integrieren.

Webentwicklung basiert heute häufig auf existierenden Frameworks und Bibliotheken, die teilweise komplexe Abhängigkeiten haben. Damit man sich hiermit nicht selbst beschäftigen und diese auflösen muss, gibt es Paketmanager, die diese Aufgabe automatisiert übernehmen und nur den Code mit ausliefern, der auch wirklich benötigt wird. Die Weban-

wendung muss nach Fertigstellung noch auf den Server übertragen werden – dieser Prozess nennt sich Deployment und wird auch von Tools unterstützt.

# LEKTION 3

## ENTWICKLUNG VON STATISCHEN WEBSEITEN

### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- einfache statische Webseiten mit HTML zu realisieren.
- Webseiten mit CSS und den dazugehörigen Frameworks zu gestalten.
- das Grundgerüst einer einfachen Website zu konzipieren.

## 3. ENTWICKLUNG VON STATISCHEN WEBSEITEN

### Einführung

Schon in den frühen 1990er-Jahren wurde die Sprache HTML (Hypertext Markup Language) zur Programmierung von Webseiten entwickelt und eingesetzt. Auch heute sind Grundkenntnisse von HTML wichtig, um eigene Websites erstellen zu können. Häufig wird zwar zur Pflege des Contents ein CMS wie Wordpress (WordPress, n. d.) verwendet, das ist aber gar nicht immer nötig. Außerdem ist auch bei einem CMS die Anpassung von Templates häufig erforderlich, wofür wieder HTML benötigt wird.

Mit HTML allein wären Webseiten ziemlich langweilig. Deren Gestaltung übernimmt CSS, die Cascading Stylesheets (Web Archive, n. d.-b). Für die Interaktivität ist eine weitere Programmiersprache zuständig, nämlich JavaScript. Mithilfe von HTML, CSS und JavaScript sind anspruchsvolle Webseiten realisierbar. Häufig kommen noch Bilder dazu, die in unterschiedlichen Formaten wie PNG, JPG oder SVG vorliegen können, damit ein interaktives Multimedia-Erlebnis entsteht.

Content im Web wird hauptsächlich gelesen, daher spielt die Typografie eine große Rolle. Durch die Nutzung von CSS können die Schriftarten beeinflusst werden, sodass sich ein guter Lesefluss ergibt.

Weil das Zusammenspiel dieser Technologien komplex ist, bietet es sich in vielen Fällen an, fertige Frameworks einzusetzen. Viele davon verfügen auch über Templates, die für unterschiedliche Arten von Webseiten geeignet sind. Ganze Prototyp-Websites lassen sich auch kaufen und anschließend einfach in Wordpress oder in ein anderes CMS integrieren. Anpassungen erfordern aber immer Kenntnisse in HTML.

### 3.1 Grundlagen von HTML5

Ein Startup wurde gerade gegründet und möchte seine erste eigene Website aufbauen. Da es plant, seine Dienste auch webbasiert anzubieten, soll ein universeller Ansatz verwendet werden, sodass sich ein möglichst hoher Wiedererkennungswert ergibt.

#### Die erste HTML-Seite

Ein guter Startpunkt ist die Entwicklungsumgebung. Nach dem Start von Visual Studio Code muss zunächst ein neues Projekt angelegt werden. Dazu ist es sinnvoll, zunächst einen neuen Ordner (z. B. mit dem Namen IPWA01-01) anzulegen. Anschließend lassen sich Ordner für die einzelnen Lektionen innerhalb dieses Verzeichnisses hinzufügen. Am

einfachsten geschieht das über die Selektion des Ordner-Symbols ganz oben links mit der anschließenden Auswahl „Open Folder“. Danach erscheint eine Abfrage, ob man dem:der Autor:in vertrauen möchte, die bedenkenlos mit „Yes“ (ja) beantwortet werden kann:

Jetzt ist das erste Projekt angelegt und die Arbeit an der Website kann beginnen. Sinnvoll ist ein Start mit der Homepage, diese wird üblicherweise unter dem Namen index.html abgelegt. Mittels des „+“-Symbols beim Dokument lässt sich eine neue Datei mit diesem Namen anlegen, wenn die Maus das Projekt berührt.

**Abbildung 5: Anlegen einer neuen Datei bei Visual Studio Code**



---

Quelle: Christian Winkler, 2022.

Die Datei ist zunächst leer. Ein guter Anfang ist eine einfache HTML-Datei, die nur einen Titel und eine Überschrift enthält. Die einzelnen Tags/Elemente werden im Laufe des Kapitels noch genau erklärt, für den Start können sie zunächst einfach so übernommen werden.

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <title>IPWA01</title>
  </head>
  <body>
    <h1>IPWA01</h1>
  </body>
</html>
```

Wie sieht die Darstellung dieser Datei aus? Man kann die Datei im Explorer (oder Finder) anklicken und sie wird im Webbrowser dargestellt. Viel komfortabler geht das mit Visual Studio Code, indem „Live Server“ (Visual Studio, n. d.) als Extension installiert wird. Extensions können über „File > Preferences > Extensions“ gesucht und installiert werden. Nach der erfolgreichen Installation findet sich unten rechts ein „Go Live“-Button. Nach Klick darauf öffnet sich ein Browser mit der HTML-Datei. Im Hintergrund hat Visual Studio Code einen Webserver gestartet – der Browser ruft die davon gelieferte Seite auf (siehe folgende Abbildung).

Abbildung 6: Darstellung der ersten HTML-Datei im Browser



Quelle: Christian Winkler, 2022.

Es ist nun einfach möglich, den Inhalt der HTML-Datei zu modifizieren. Beim Speichern aktualisiert sich die Darstellung im Browser automatisch.

## Struktur der HTML-Datei

### HTML-Tag

Das ist das zentrale Konzept von HTML, mit dem Inhalte strukturiert werden können. Ein Element hat ein Start- und einen Ende-Tag.

HTML ist eine Auszeichnungs-Sprache (sog. Markup-Language), die für die Gliederung und Formatierung von Texten gedacht ist. Typisch für diese Sprachen sind die **Tags**, die man auch in dem obigen HTML-Dokument findet. In HTML und verwandten Sprachen werden diese in spitze Klammern gesetzt.

Betrachten wir das HTML-Dokument im Einzelnen:

- `<!DOCTYPE html>`

Schon beim ersten Tag handelt es sich um eine Spezialität, nämlich um die sog. Dokumententypdeklaration (DTD). Hier gibt es zwar theoretisch unterschiedliche Möglichkeiten, allerdings beginnen alle HTML5-Dokumente mit genau dieser Sequenz.

- `<html lang="de">`

In HTML bilden die meisten Tags sog. Elemente. Solche Tags müssen auch wieder geschlossen werden. Der `html`-Tag ist ein solcher und „kapselt“ das gesamte HTML-Dokument. Tags können **Attribute** haben, in diesem Fall ist das der Parameter „`lang`“, der ausdrückt, dass das Dokument auf Deutsch verfasst ist. Ein HTML-Dokument beginnt (abgesehen von der DTD) immer mit `<html>` und endet mit `</html>`.

- `<head>`

HTML-Dokumente bestehen aus einem Head und einem Body. Das Head-Element kapselt den Head. Dort sind Metainformationen über das Dokument enthalten. Außerdem können dort auch Stylesheets geladen werden.

- `<title>`

Das Title-Element enthält den Titel der Webseite, wie er z. B. als Fenster-/Tab-Titel, bei Bookmarks, oder in Suchmaschinen angezeigt wird. Dieses Element ist das erste, das direkt Text zwischen dem öffnenden und dem schließenden Tag enthält.

- <body>  
Dieses Element kapselt den gesamten sichtbaren Inhalt der Seite. Innerhalb dieses Elements können sehr viele andere Tags verwendet werden, die den Inhalt strukturieren und auch formatieren.
- <h1>  
HTML erlaubt Überschriften auf bis zu sechs Ebenen – h1 ist dabei die höchste Ebene und stellt damit die wichtigste Überschrift dar.

HTML bietet sehr viel mehr Möglichkeiten als in diesem einfachen Beispiel. Es gibt z. B. Elemente, die einen Textabschnitt als Überschrift, Absatz oder Liste kennzeichnen. Andere fügen Zeilenumbrüche oder Bilder ein oder geben an, dass ein bestimmter Abschnitt fett oder kursiv dargestellt werden soll. Sogar Formulare mit Eingabefeldern und Schaltflächen können mit HTML-Tags definiert werden. Die Syntax der Tags ist dabei stets gleich: Sie bestehen aus einem Start- und einem Ende-Tag. Dazwischen befindet sich der Inhalt, auf den sich das Element bezieht:

```
<tagname>Inhalt</tagname>
```

Wenn ein Element keinen Inhalt hat, z. B. weil es lediglich einen Zeilenumbruch oder ein Bild einfügt, dann darf es auch in einer verkürzten Schreibweise ohne Ende-Tag definiert werden. Zum Beispiel fügt <br> einen Zeilenumbruch ein, da ist es nicht sinnvoll, einen Inhalt in dieses Tag zu schreiben.

## Wichtige Tags für die Seitenstruktur

HTML als Auszeichnungssprache erlaubt die **semantische Strukturierung** der Seiten mit Elementen. Typische Webseiten enthalten immer wieder die gleichen Strukturelemente. Am einfachsten sieht man das, wenn man eine einzelne Webseite des hypothetischen Startups betrachtet (siehe folgende Abbildung).

**Semantisches Markup**  
Dieser Begriff bezeichnet HTML-Code, dessen Struktur einen eindeutigen und klaren Bezug zum Inhalt hat. Dies ist wichtig für maschinelles Erkennen des Inhalts, Sprachsuche und Screen-Reader.

**Abbildung 7: Grafik zur Seitenstruktur des Startups**

---

The screenshot shows a website layout for 'WebAppStartup'. At the top is a navigation bar with a search input field and links to 'Instagram', 'Facebook', and 'Kontakt'. Below the navigation is a horizontal menu with 'Homepage', 'Unsere Leistungen', 'Unsere Produkte', 'Unsere Preise', and 'Über uns'. The main content area features a hero section with the heading 'Willkommen auf der Homepage von WebAppStartup' and a subtext 'Wir konzipieren und implementieren fantastische Web-Applikationen!'. Below this are three service sections: 'Unsere Leistungen' (with a subtext about web design), 'Unsere Produkte' (with a subtext about website templates), and 'Unsere Preise' (with a table showing prices for consulting packages). The 'Unsere Preise' table has two columns: 'Leistung' and 'Einzelpreis'. The data is as follows:

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,--
Tagessatz Beratung	€ 600,--
Homepage Baukasten pro Monat	€ 1.000,--

At the bottom right of the main content area are links to 'Datenschutz' and 'Impressum'.

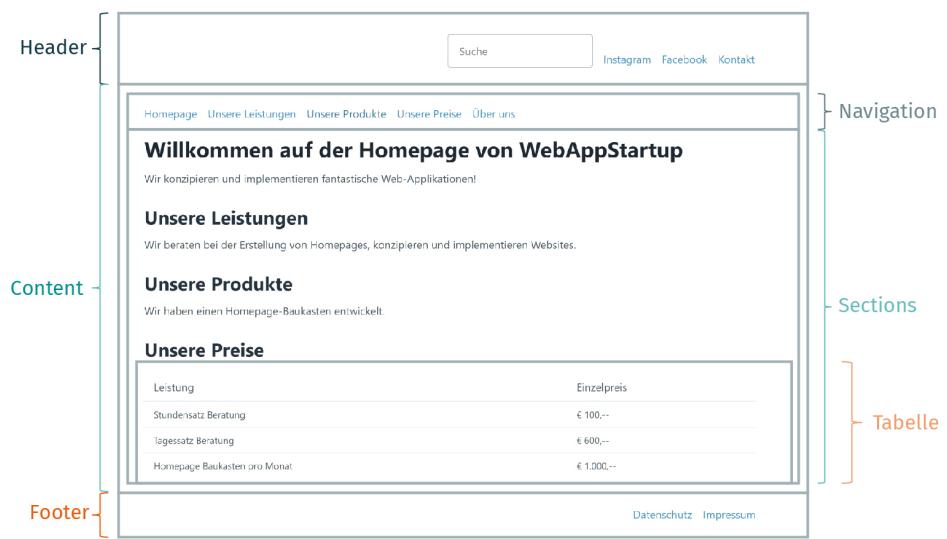
---

Quelle: Christian Winkler, 2022.

So sieht die Seite im Browser aus. Tatsächlich enthält bereits diese HTML-Seite fast alle Strukturelemente, die üblicherweise benötigt werden. Um deren Verwendung zu verstehen, ist es sinnvoll, sich die einzelnen Teile genauer anzusehen.

Die Struktur kann man nur optisch erkennen, deshalb wurde sie in der unteren Abbildung nochmal grafisch hervorgehoben.

Abbildung 8: HTML-Seite mit hervorgehobener Inhaltsstruktur



Quelle: Christian Winkler, 2022.

Die unterschiedlichen Elemente kann man hier deutlich erkennen.

## Header

Der Header ist auf allen Seiten gleich und enthält eine Möglichkeit zur Suche, Links zu den sozialen Netzwerken und Kontaktinformationen. Die Links bilden semantisch eine Liste.

```
<header>
  <nav>
    <ul>
      <li>
        <form>
          <input type="text" name="suche" placeholder="Suche">
        </form>
      </li>
      <li><a href="https://www.instagram.com/WebAppStartup">Instagram</a></li>
      <li><a href="https://www.facebook.com/WebAppStartup">Facebook</a></li>
      <li><a href="contact.html">Kontakt</a></li>
    </ul>
  </nav>
</header>
```

Der Header wird durch den header-Tag eingeleitet. Nav steht für eine Navigation, die in Form einer unnummerierten Liste ul dargestellt wird. Die einzelnen Listen-Elemente werden durch li gekapselt. Das erste Listenelement enthält ein Formular, das wiederum aus einem Input-Element besteht. Dieses hat Attribute für die Art der Eingabe („text“), für den Namen, unter dem es ansprechbar ist („suche“) und einen Platzhalter („Suche“), der verschwindet, sobald er angeklickt wird. Die anderen Listenelemente bestehen aus a-Elementen, die die Links zu anderen Seiten enthalten. Die verlinkte URL ist in dem „href“-Attribut genannt.

## Content

Der Content-Bereich wird durch das <main>-Element gekapselt und gliedert sich auf in

- **Navigation**

Auch hier bilden die Navigationspunkte eine Liste.

```
<nav>
  <ul>
    <li><a href="home.html">Homepage</a></li>
    <li><a href="services.html">Unsere Leistungen</a></li>
    <li><a href="products.html">Unsere Produkte</a></li>
    <li><a href="prices.html">Unsere Preise</a></li>
    <li><a href="about.html">Über uns</a></li>
  </ul>
</nav>
```

- **Sektionen mit Überschriften**

Es gibt einen Abschnitt mit einer Hauptüberschrift und einer Erklärung des Firmenzwecks in Form eines Textabsatzes. Anschließend folgen Abschnitte mit etwas kleineren Überschriften.

```
<section>
  <h1>Willkommen auf der Homepage von WebAppStartup</h1>
  <p>
    Wir konzipieren und implementieren fantastische Web-Applikationen!
  </p>
</section>
```

Hier gibt es neue HTML-Elemente. Ein Abschnitt wird durch section gekapselt. In diesem Abschnitt ist die Hauptüberschrift h1 enthalten sowie ein Absatz p (wie Paragraf). Die section hat in diesem Beispiel (noch) keine Auswirkungen auf das Layout, sondern dient rein der semantischen Gliederung. Später könnte über CSS-Regeln auch noch ein Layout dafür hinterlegt werden. Der Unterabschnitt unterscheidet sich bis auf die Hierarchie der Überschrift h2 nicht von dem Hauptabschnitt. Überschriften sind bis in die sechste Hierarchieebene h6 definiert.

- **Tabelle**

Tabellen werden in HTML durch ein table-Element dargestellt. Dieses hat zwei Unter-  
elemente, den Header der Tabelle mit thead und den Inhalt mit tbody. Der Header ent-  
hält Zeilen tr (table rows) mit Heading-Elementen th.

Der Body der Tabelle enthält ebenfalls Zeilen, hier allerdings mit td (table data).

```
<table>
  <thead>
    <tr>
      <th>Leistung</th>
      <th>Einzelpreis</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Stundensatz Beratung</td>
      <td>&euro; 100,--</td>
    </tr>
    <tr>
      <td>Tagessatz Beratung</td>
      <td>&euro; 600,--</td>
    </tr>
    <tr>
      <td>Homepage Baukasten pro Monat</td>
      <td>&euro; 1.000,--</td>
    </tr>
  </tbody>
</table>
```

## Footer

Der Footer enthält wieder eine Liste von Links, die auf allen Seiten zugänglich sein sollen  
(bzw. beim Impressum und Datenschutz sogar zugänglich sein müssen).

```
<footer>
  <nav>
    <ul>
      <li><a href="privacy.html">Datenschutz</a></li>
      <li><a href="imprint.html">Impressum</a></li>
    </ul>
  </nav>
</footer>
```

Der Footer ist ganz analog dem Header der Seite aufgebaut, folgt nur einem anderen  
Zweck. Header und Footer werden manchmal „festgehalten“ und unterliegen dann nicht  
dem Scrolling der Seite.

## **Bestimmung des Layouts**

Der Browser muss nun anhand des HTML-Files ein geeignetes Layout der Seite ausrechnen und diese darstellen. Das hat überraschend gut funktioniert – hier wurde allerdings ein sehr einfaches CSS-Framework verwendet, was die Erklärung der einzelnen Teile der Seite erleichtert. Ohne CSS würde es so aussehen, wie in der folgenden Abbildung dargestellt.

**Abbildung 9: Darstellung der Seite ohne Layout**

---

- 
- [Instagram](#)
- [Facebook](#)
- [Kontakt](#)
  
- [Homepage](#)
- [Unsere Leistungen](#)
- [Unsere Produkte](#)
- [Unsere Preise](#)
- [Über uns](#)

## **Willkommen auf der Homepage von WebAppStartup**

Wir konzipieren und implementieren fantastische Web-Applikationen!

### **Unsere Leistungen**

Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites.

### **Unsere Produkte**

Wir haben einen Homepage-Baukasten entwickelt.

### **Unsere Preise**

<b>Leistung</b>	<b>Einzelpreis</b>
Stundensatz Beratung	€ 100,--
Tagessatz Beratung	€ 600,--
Homepage Baukasten pro Monat	€ 1.000,--

- [Datenschutz](#)
- [Impressum](#)

---

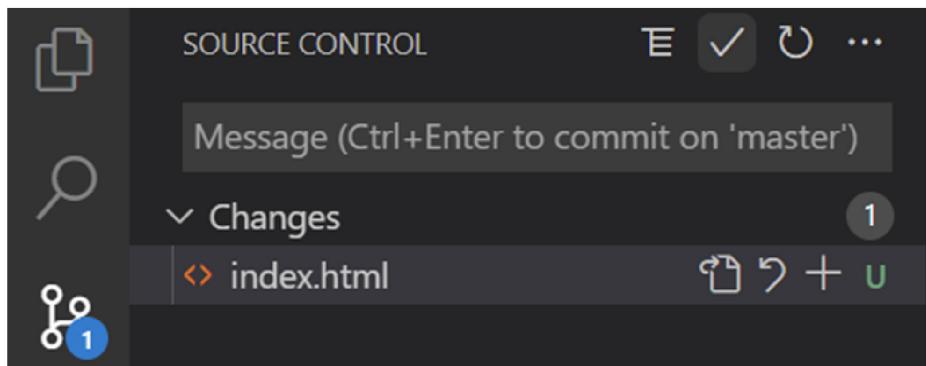
Quelle: Christian Winkler, 2022.

Semantisch ist das identisch, aber optisch weniger ansprechend. Eine genaue Erklärung von CSS inklusive möglicher Anpassungen folgt in den nächsten Abschnitten.

## Einchecken in git

Schließlich sollte das Projekt noch in git überführt werden. Das geht mit Visual Studio Code ganz einfach, dazu muss nur auf der linken Seite das Repository-Icon angeklickt und das Repository initialisiert werden. Anschließend kann mit dem Haken das geänderte (bzw. in diesem Fall neue) File committed werden (siehe folgende Abbildung).

Abbildung 10: Einchecken in git



Quelle: Christian Winkler, 2022.

Der Hinweis, dass die Änderungen bisher nicht „gestaged“ sind, kann mit „Always commit directly“ quittiert werden. Nach der Eingabe eines Textes für den Commit sind die Änderungen im Repository enthalten (eventuell wird beim ersten Mal noch nach einem Benutzernamen gefragt, die Befehle dazu müssen im Terminalfenster eingegeben werden). Mit Hilfe von „git log“ lassen sich dort auch die bisherigen Änderungen anzeigen.

## 3.2 Grundlagen von Cascading Style Sheets

Mit HTML konnte das Grundgerüst einer Webseite aufgebaut werden. Allerdings sieht die Seite noch nicht besonders ansprechend aus. Hier kommen nun die „Cascading Style Sheets“ ins Spiel. Sie verleihen den Webseiten (und Web-Apps) das Aussehen. In diesem Kapitel geht es nun darum, das Layout mithilfe von CSS so zu ändern, dass die Seite nach und nach besser aussieht.

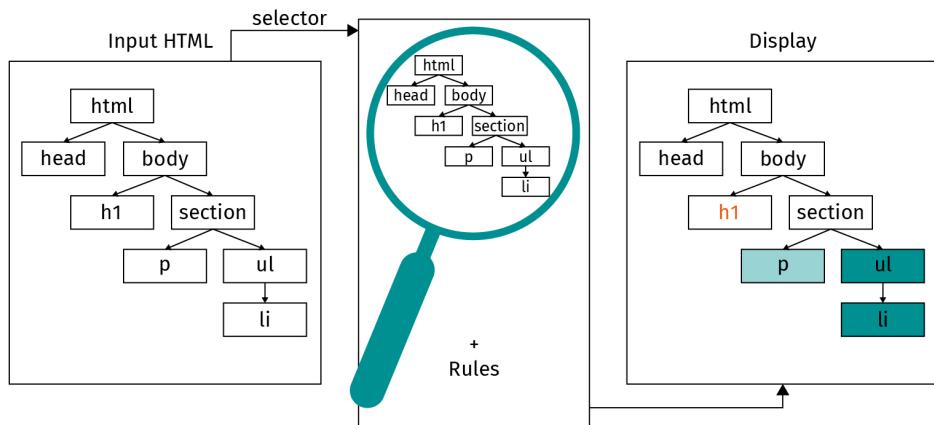
### Aufgabe von CSS

HTML gibt Webseiten eine inhaltliche Struktur, mithilfe von CSS können die Elemente angeordnet werden. Dadurch werden Inhalt und Design voneinander entkoppelt und die Darstellung kann separat angepasst werden. CSS kann z. B. auch unterschiedliche Endgeräte unterscheiden und eine optimale Darstellung auf Handy, Tablet und PC finden – das

HTML bleibt dabei unverändert. Die HTML-Seiten können durch den Austausch der CSS-Styles ihr Aussehen komplett verändern. Eine eindrucksvolle Demonstration dafür findet sich bei CSS Zen Garden (n. d.).

Die folgende Abbildung zeigt die HTML-Elemente einer Seite als hierarchische Struktur. Intern baut der Browser ebenfalls eine solche Struktur (den sog. DOM-Tree, eine hierarchisch aufgebaute Baumstruktur) auf, um mithilfe von CSS die richtige Darstellung für jedes Element zu finden.

**Abbildung 11: HTML wird mithilfe von CSS gestyliert**



Quelle: Christian Winkler, 2022.

### CSS-Selektoren

Dies sind Regeln zum Selektieren von HTML-Elementen. Die wichtigsten sind: HTML-Tags, CSS-Klassen und einmalige CSS-IDs.

Der linke Teil der Abbildung zeigt die hierarchische HTML-Struktur einer Beispielseite. Über die Auswahlregeln in den CSS-Dateien (den sog. **Selektoren**) werden durch den Browser bestimmte HTML-Elemente selektiert, auf die dann die dort genannten Layoutregeln angewendet werden. Dadurch ändern die HTML-Elemente ihr Aussehen und werden, wie auf der rechten Seite der Abbildung zu sehen, anders dargestellt. In diesem Beispiel ändern sich Hintergrund- und Schriftfarbe.

### CSS-Selektoren

Um mit CSS HTML-Elemente gezielt anzusprechen, gibt es Auswahlregeln. Diese Selektoren können ein HTML-Element oder eine Gruppe von Elementen selektieren. Im Prinzip gibt es drei unterschiedliche Arten von Selektoren, die miteinander kombiniert werden können. Hinter dem Selektor steht immer die Regel, die auf das Objekt angewendet werden soll. Die Beispiele sind selbsterklärend gewählt – eine detaillierte Einführung erfolgt etwas später:

- **Tag**

Hierdurch werden alle HTML-Elemente selektiert, die diesen Tag-Namen haben.

Beispiel: `h1 { color: green; }`

- **Klassen**

HTML-Elemente haben ein Attribut mit dem Namen „class“. Will man über verschiedene Elemente selektieren, ist das sehr nützlich, denn es erlaubt die Selektion aller HTML-Elemente mit dem gleichen Klassennamen. Dieser Selektor beginnt mit „.“ (Punkt), gefolgt vom Klassennamen.

Beispiel: `.green { color: green; }`

- **IDs**

Jedes HTML-Element hat außerdem auch die Möglichkeit, ein Attribut „id“ zu vergeben. Dieses muss eindeutig sein. Damit lassen sich Eigenschaften eines einzelnen Elements sehr präzise (ohne Nebeneffekte auf andere Elemente) setzen. Die ID wird mit einem „#“-Zeichen als CSS-Selektor formuliert.

Beispiel: `#main-content { background-color: grey; }`

CSS-Selektoren können in unterschiedlicher Weise **kombiniert** werden:

- Alle Regeln gelten auch für untergeordnete Elemente. In dem oberen Beispiel würde eine Regel für `section` also auch die darin enthaltenen `h1`- und `p`-Elemente betreffen.
- Möchte man eine Regel nur für eine Klasse eines bestimmten Elements nutzen, kann man z. B. `h1.green` als Selektor verwenden.
- Möchte man die gleiche Regel für mehrere Selektoren verwenden, kann man die Selektoren mit Komma getrennt hintereinander schreiben:  
Beispiel: `h1, section, table { color: green; }`
- Soll die Regel für alle Kindelemente eines übergeordneten Elements gelten, setzt man nach dem übergeordneten Element ein Leerzeichen: `div p`.
- Möchte man die Regel aber nur für das erste Kindelement gelten lassen, verwendet man das plus-Symbol: `div+p`.

#### Kombination von Selektoren

Die Möglichkeit, Selektoren miteinander zu kombinieren und Eigenschaft zu vererben, machen CSS sehr leistungsfähig.

Die Möglichkeiten der Selektoren gehen noch deutlich weiter, für den Zweck dieses Kurses genügt dies als Überblick. Sollten später weitere Selektoren benötigt werden, werden diese anhand von Beispielen eingeführt.

## CSS-Regeln

Jeder Selektor enthält im Anschluss eine Regel, die für die ausgewählten Elemente angewendet werden soll. Eine Regel besteht immer aus mehreren Eigenschaft-Wert-Paaren, also z. B. `color: green`. Die Menge an Eigenschaften („properties“) ist sehr umfangreich, daher werden hier nur die wichtigsten vorgestellt.

- `color`

Damit wird die Farbe eines Elements festgelegt.

- `background-color`

Ähnlich wie die Farbe wird damit die Hintergrundfarbe eines Elements definiert.

Farben können in unterschiedlicher Weise spezifiziert werden. Wie oben im Beispiel können dafür die echten Namen der Farben genutzt werden, es können aber auch **RGB**-Werte in hexadezimaler Notation Anwendung finden. Der Wert der einzelnen Farbkomponenten

**RGB**  
Steht für Rot-Grün-Blau und ist die Basis der additiven Farbmischung. Werte für die einzelnen Komponenten können zwischen 0 und 255 liegen.

geht dabei von 0 bis 255. Die Farbe #ffffff entspricht somit weiß, #ff0000 rot. Will man etwas Platz sparen, kann man auch mit Tripeln arbeiten, die sind dann weniger detailliert abgestuft. Hier entspricht dann #00f blau und #0f0 grün.

Neben den Farben kann CSS auch Größen festlegen.

- **font-size**

Damit wird die Schriftgröße des Texts in einem Element festgelegt.

- **width** und **height**

Diese Eigenschaften legen die Breite und die Höhe eines Elements fest.

- **margin**

Hiermit kann der Rand um ein Element festgelegt werden.

- **padding**

Das padding funktioniert so ähnlich wie der Rand, allerdings wird es in das Element integriert und trägt dieselbe Farbe.

- **border**

Hierdurch kann der Rand um ein Element festgelegt werden. Der Rand kann neben der Größe auch noch eine Farbe enthalten.

Größenangaben sind in unterschiedlichen Einheiten möglich: Oftmals werden Pixel (angelehnt an die am Bildschirm) verwendet, dazu dient die Einheit „px“. Größenangaben können außerdem relativ erfolgen, dazu ist die Verwendung der Einheit „em“ möglich, die der Standard-Schrifthöhe entspricht. CSS beherrscht noch weit mehr Größeneinheiten, die hier nicht relevant sind.

Zusätzlich gibt es noch Eigenschaften, die die Positionierung der Elemente zueinander festlegen und ob Elemente relativ, absolut oder nach ganz bestimmten anderen Regeln platziert werden. Diese Logik ist sehr komplex und stellt auch CSS-Profis immer wieder vor große Herausforderungen. Wenn solche Regeln im Kurs benötigt werden, werden sie kurz erklärt. Das Gesamtkonzept zu erläutern, würde jedoch den Umfang des Kurses sprengen (MDN, n. d.-f).

**CSS-Eigenschaften**  
Es gibt eine sehr große Menge an CSS-Eigenschaften. Selbst professionelle Entwickler verlieren hier schnell den Überblick.

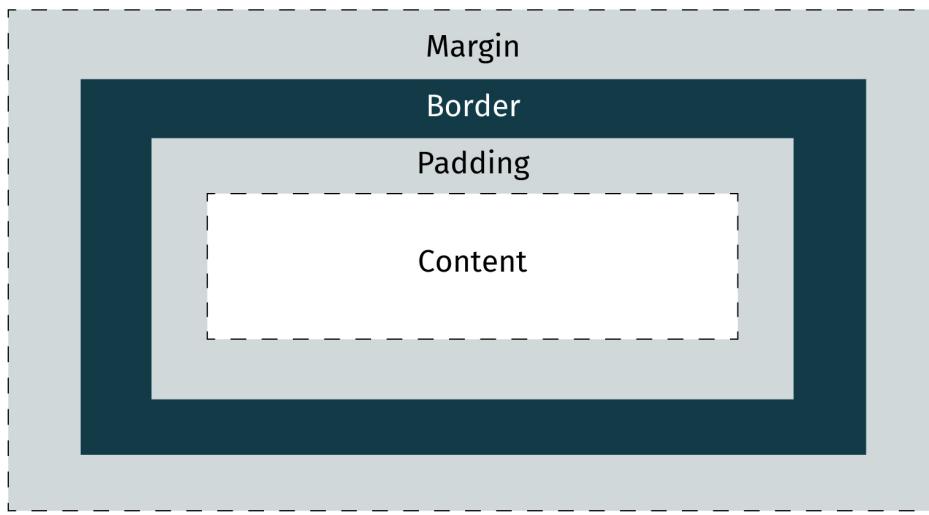
CSS hat auch **Eigenschaften** für eigene Fonts, Hintergrundbilder, Übergänge und Animationen. Im Vergleich zu HTML ist CSS damit viel komplexer. Auch diese Eigenschaften werden bei Bedarf in diesem Skript an der Stelle vorgestellt, wo sie verwendet werden. Eine Liste aller CSS-Eigenschaften findet sich bei (MDN, n. d.-b).

## **CSS-Layout**

HTML-Elemente werden in sog. Block- und Inline-Elemente unterschieden. Vereinfacht gesagt benötigt ein Block-Element immer die gesamte Breite und beginnt in einer neuen Zeile. Inline-Elemente werden „im Text“ dargestellt. Ein typisches Inline-Element ist z. B. em (für emphasize), mit dem Text hervorgehoben wird, p für einen Absatz ist hingegen ein Block-Element.

Die oben genannten Eigenschaften margin, border und padding bestimmen die Größe eines Elements, das zugehörige Modell heißt CSS Box Model.

Abbildung 12: CSS Box Model



Quelle: Christian Winkler, 2022.

Dabei ist zu beachten, dass sich die Margins von angrenzenden Elementen überlappen dürfen, nicht aber die Borders, Paddings und Contents.

### Beispiel mit CSS

Ausgehend von dem obigen HTML-Beispiel kann nun CSS-Code hinzugefügt werden, um das Layout etwas ansprechender (bzw. bunter) zu gestalten. In den **Head**-Bereich wird dazu folgender Code eingefügt:

```
<style type="text/css">
  p { color: red; }
  h1 {
    background-color: green;
    padding: 1em;
    border: 5px solid black;
  }
  h1, h2 { color: orange; }
  table { font-size: 200 %; }
</style>
```

#### CSS im Head

CSS muss immer im Head der HTML-Seite referenziert oder eingefügt werden.

Im Einzelnen bedeuten die Anweisungen:

- Die Farbe für die Textblöcke soll rot sein.
- Die Überschrift h1 soll einen grünen Hintergrund, ein „padding“ von einer Schrifthöhe (rundum) und einen fünf Pixel breiten schwarzen Rand rundum erhalten.
- Die Textfarbe für h1 und h2 soll orange sein.

- Die Größe des Textes in der Tabelle wird auf 200 % (also doppelt so groß als normal) gesetzt.

Das Ergebnis besticht nicht durch seine Ästhetik, illustriert aber viele der Basis-Konzepte von CSS.

Abbildung 13: Ergebnis des CSS-Experiments

---

The screenshot shows a website layout with a green header containing the text "Willkommen auf der Homepage von WebAppStartup". Below the header, there is a red paragraph: "Wir konzipieren und implementieren fantastische Web-Applikationen!". Underneath this, there are several sections with orange headings and red text:

- Unsere Leistungen**: "Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites."
- Unsere Produkte**: "Wir haben einen Homepage-Baukasten entwickelt."
- Unsere Preise**: A table showing prices for different services:

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,--
Tagessatz Beratung	€ 600,--
Homepage Baukasten pro Monat	€ 1.000,--

---

Quelle: Christian Winkler, 2022.

## Überprüfen mit dem Web Inspector

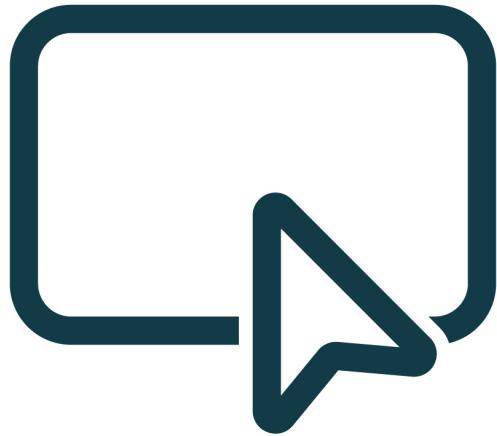
### Web Inspector

Alle Browser verfügen heute über einen Web Inspector. Die Grundidee kam von einem Plugin „Firebug“, das in der Zwischenzeit eingestellt wurde.

Hat man eine CSS-Regel mit einem Selektor „gebaut“, ist sie am einfachsten im Browser mit dem **Web Inspector** zu überprüfen. Alle gängigen Browser haben eine solche Möglichkeit integriert, normalerweise wird diese mit Ctrl-Shift-I (bei MacOS mit Cmd-Alt-I) aufgerufen. Ein Element lässt sich anschließend mit dem Element-Selektor auswählen oder direkt mit der Tastenkombination Ctrl-Shift-C (bzw. Cmd-Alt-C bei MacOS).

Abbildung 14: Element-Selektor

---



---

Quelle: Christian Winkler, 2022.

Anschließend lassen sich **Elemente auswählen** und Infos über das gewählte Element werden in der rechten unteren Ecke des Browsers dargestellt.

Damit können mehrere Dinge erreicht werden:

- Es lässt sich ermitteln, ob eine der CSS-Regeln auf das gewünschte Element zutrifft.
- Die Regeln selbst sind veränderlich, Eigenschaftswerte können modifiziert oder auch neue Eigenschaften hinzugefügt werden. In Web-Inspector lassen sich z. B. die Eigenschaftswerte ändern, um die Überschriften in einer anderen Farbe erscheinen zu lassen.

#### Element-Selektor

Der Element-Selektor ist eine der wichtigsten Funktionen des Web Inspectors und bewährt sich auf „fremden“ Webseiten zur Analyse des Codes.

Abbildung 15: Web Inspector mit CSS-Regeln für <h1>

---

The screenshot shows the 'Styles' tab of a browser's developer tools. At the top, there are tabs for 'Styles', 'Computed', 'Layout', 'Event Listeners', and a 'More' icon. Below the tabs is a toolbar with icons for ':hov', '.cls', a plus sign, a magnifying glass, and a refresh symbol. A 'Filter' input field is also present. The main area displays the following CSS code:

```
element.style {  
}  
h1, h2 {  
    color: orange;  
}  
h1 {  
    background-color: green;  
    padding: 1em;  
    border: 5px solid black;  
}
```

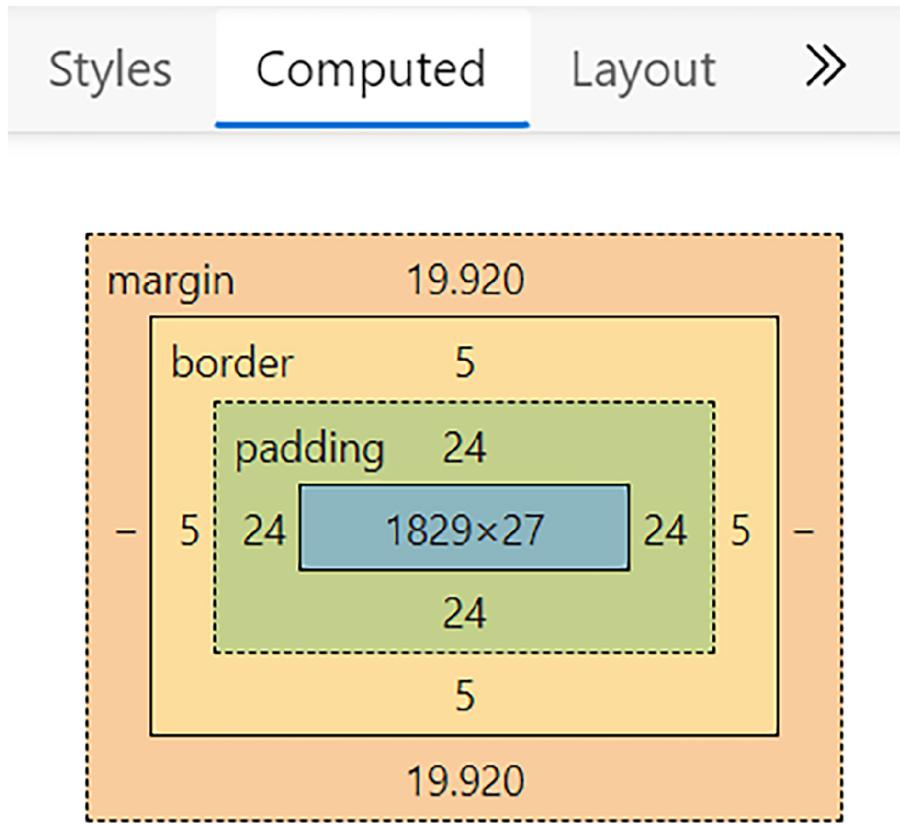
Two source file paths are shown on the right side of the rules: 'home.html:14' and 'home.html:9'.

---

Quelle: Christian Winkler, 2022.

- Über „Computed Layout“ kann man sich anzeigen lassen, wie der Browser die Regeln umgesetzt hat. Auch diese Darstellung wird durch die oben beschriebene Modifikation angepasst.

Abbildung 16: Web Inspector mit berechnetem Layout für <h1>



Quelle: Christian Winkler, 2022.

Wenn man CSS-Layouts selbst ändern möchte, ist der Web Inspector ein essenzielles Tool, mit dem man auch interaktiv experimentieren und sich den CSS-Fundamenten spielerisch nähern kann.



### ZUSAMMENFASSUNG

HTML und CSS stellen die Basis für die Entwicklung ganzer Websites, aber auch Online-Shops usw. dar. Dabei bilden sie ein sehr ungleiches Paar. HTML ist eine einfache, semantische Seitenbeschreibungssprache mit einer übersichtlichen Menge an Tags. Für viele Anforderungen gibt es eine „richtige“ Variante, wie diese in HTML abgebildet werden können. Als Daumenregel kann man sich merken, dass der HTML-Markup möglichst einfach und semantisch sein sollte – dann hat man alles richtig gemacht.

Demgegenüber steht CSS mit einem sehr komplexen System an Selektoren, die hierarchisch aufgebaut sind und Eigenschaften an die darunterliegenden Elemente vererben. CSS ist eng verbunden mit dem Layout der Webseiten, für die es sehr komplexe Regeln gibt, die hier nur im Ansatz besprochen wurden. Auch für professionelle Web-Entwickler:innen sind diese nicht immer einfach zu durchschauen. Dazu kommt eine riesige Menge an Eigenschaften und Eigenschaftswerten, die über CSS gesetzt werden können. Hier hilft oft nur ein Nachschlagewerk wie MDN (n. d.-a), um alle möglichen Varianten nutzen zu können. Eigenschaften und die dazugehörigen Werte sind auch einem ständigen Wandel unterworfen und werden durch die Browserhersteller regelmäßig ergänzt.

Glücklicherweise besitzen alle modernen Browser einen sog. Web Inspector, mit dem sich die CSS-Selektoren und Regeln überprüfen lassen. Aber nicht nur das, sie lassen sich auch auf der aktuellen Webseite verändern und haben dann direkt Einfluss auf das Layout der Seite. So kann man sich CSS explorativ erschließen und mit den Selektoren und Eigenschaften experimentieren, um die Ergebnisse schließlich in die Seite selbst übernehmen zu können.

# LEKTION 4

## ERWEITERTE KONSTRUKTIONSTECHNIKEN

### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- das Konzept von „Responsive Design“ zu erklären.
- eigene „Media Queries“ in CSS zu entwerfen.
- ein professionelles CSS-Framework in Ihre Web-App einzubinden.

## 4. ERWEITERTE KONSTRUKTIONSTECHNIKEN

### Einführung

Auch wenn HTML eine einfache, semantische Sprache ist, können damit komplexe Websites konstruiert werden. HTML repräsentiert dabei den Content und die Verlinkung zwischen den Seiten. Das Layout wird mit CSS aufgebaut, was eine deutlich komplexere Sprache darstellt. Auch wenn man dies nicht direkt als Programmierung bezeichnen kann, sind die zur Verfügung stehenden Möglichkeiten überwältigend und können auch leicht überfordern. Layouts, die man mit einfachen Regeln schaffen kann, sind gut nachvollziehbar, entsprechen aber oft nicht dem gewünschten, ästhetischen Ergebnis. Der Weg zu einer komplexen, optisch ansprechenden Website ist da noch weit.

Erschwerend kommt hinzu, dass in den letzten Jahren Geräte mit unterschiedlichsten Bildschirmauflösungen aufgetaucht sind. Das beginnt bei den Desktop-Computern mit Ultra-HD-Monitoren. Auch Laptops haben sehr unterschiedliche Bildschirmauflösungen. Hinzu kommen Endgeräte, wie Tablets, die man im Hoch- oder Querformat bedienen kann und Handys, die mit noch kleineren Bildschirmen arbeiten und auch gedreht werden. Bei den letzten beiden Gerätetypen muss man außerdem darauf achten, dass die Bedienung nicht über ein Zeigegerät, sondern über das Touch-Display erfolgt. Bei kleineren Displays sollte sich der Content zudem anders verteilen und z. B. eine Navigation einklappen.

Viele dieser komplexen Anforderungen wiederholen sich für jede Website, die man baut. Daher wurden leistungsfähige Frameworks entwickelt, die viele dieser Routineaufgaben erledigen und ein angepasstes Layout ermöglichen.

### 4.1 Responsives Web-Design

Es gibt bereits viele verschiedene Endgeräte und es kommen ständig weitere hinzu. Die Bandbreite erstreckt sich vom Smartphone bis zu HD-Fernsehern. Früher wurde jede Website für jede Endgerätekategorie (teilweise sogar für unterschiedliche Bildschirmauflösungen) optimiert. Durch die vielen Varianten ist das heute gar nicht mehr möglich.

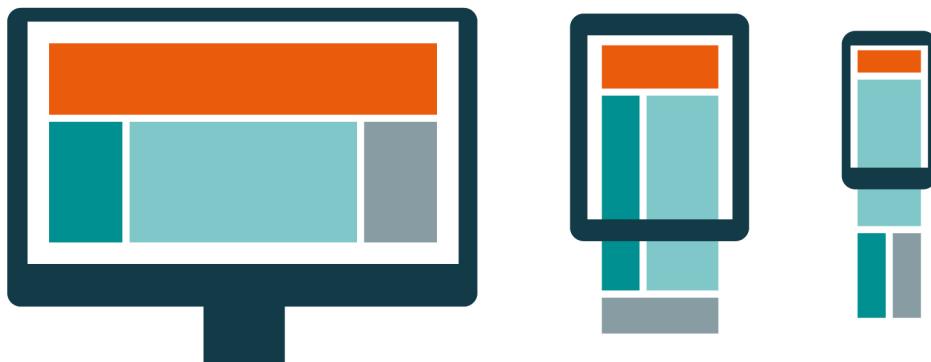
Bei den meisten Websites kann man sehen, dass sich pro Endgerät meist nur die Features ändern, während das Grunddesign gleich bleibt. Also hat man nach einem Weg gesucht, wie sich das Design flexibel an die unterschiedlichen Bildschirmauflösungen der Endgeräte anpasst. Dabei soll die Darstellung auf jedem Gerät gut aussehen – idealerweise so, als ob sie auf das Gerät spezifisch angepasst wurde. Das Ergebnis nennt sich „responsives Web-Design“, auf Englisch „Responsive Design“.

Die Realisierung ist nicht sehr kompliziert. Meistens versucht man, den Content einer Webseite in kleinere Einheiten (sog. **Kacheln**) zu unterteilen. Diese werden dann auf einem Gitter positioniert. Die Größe des Gitters hängt vom Endgerät und dessen Orientierung ab. Die Kacheln werden auf dem Gitter angeordnet. Dazu erhalten diese zusätzliche Attribute, die die gewünschte Breite in den unterschiedlichen Geräteklassen enthalten. Bilder müssen auch für die Geräte richtig skaliert werden. Es ist außerdem wichtig, keine absoluten Einheiten (wie Pixel) im CSS zu verwenden, weil diese sich auf den Geräten unterschiedlich verhalten.

### Kacheln

Kacheln sind ein entscheidendes Strukturelement moderner Websites, damit diese sich leichter auf unterschiedliche Endgeräte anpassen lassen.

Abbildung 17: Anordnung von Kacheln auf unterschiedlichen Endgeräten



Quelle: Tomáš Procházka, 2012, [CC0].

Neben der Darstellung der Inhalte geht es auch um das Weglassen von unwichtigen Bestandteilen. Besonders bei kleinen Geräten, wie Handys, müssen nicht alle Informationen gleichzeitig dargestellt werden, die auf dem Desktop auf einen Blick sichtbar sein sollten. Menüs oder Zusatzinformationen sollten dort eher ausblendet werden, um mit dem **geringen Platz** des Bildschirms ökonomisch umzugehen. Auch dies ist ein integraler Bestandteil des reaktionsfähigen Webdesigns.

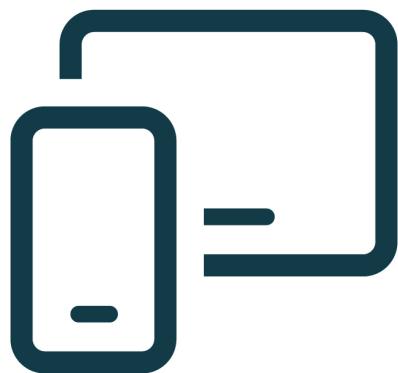
### Platznot auf dem Handy

Obwohl die Bildschirme von Handys immer größer werden, steht eigentlich immer zu wenig Platz für die Darstellung der Informationen zur Verfügung.

Der Test und die Optimierung von Webseiten für so viele Endgeräte ist eine echte Herausforderung. Für eine:n Web-Entwickler:in wäre es sehr umständlich, alle Änderungen einer Webseite immer in allen möglichen Endgeräten zu testen. Zum Glück ist das auch nicht notwendig, denn die (Desktop-)Browser unterstützen die Simulation unterschiedlicher Endgeräte. Das funktioniert auch wieder über den Web Inspector, dort kann dann der Shortcut Ctrl-Shift-M (Cmd-Alt-M bei MacOS) gewählt oder das Geräteauswahl-Icon verwendet werden. Die Darstellung ändert sich sofort und kann dann in der obersten Zeile des Browsers so angepasst werden, wie sie auf den entsprechenden Endgeräten aussehen würde.

Abbildung 18: Geräteauswahl im Browser

---



---

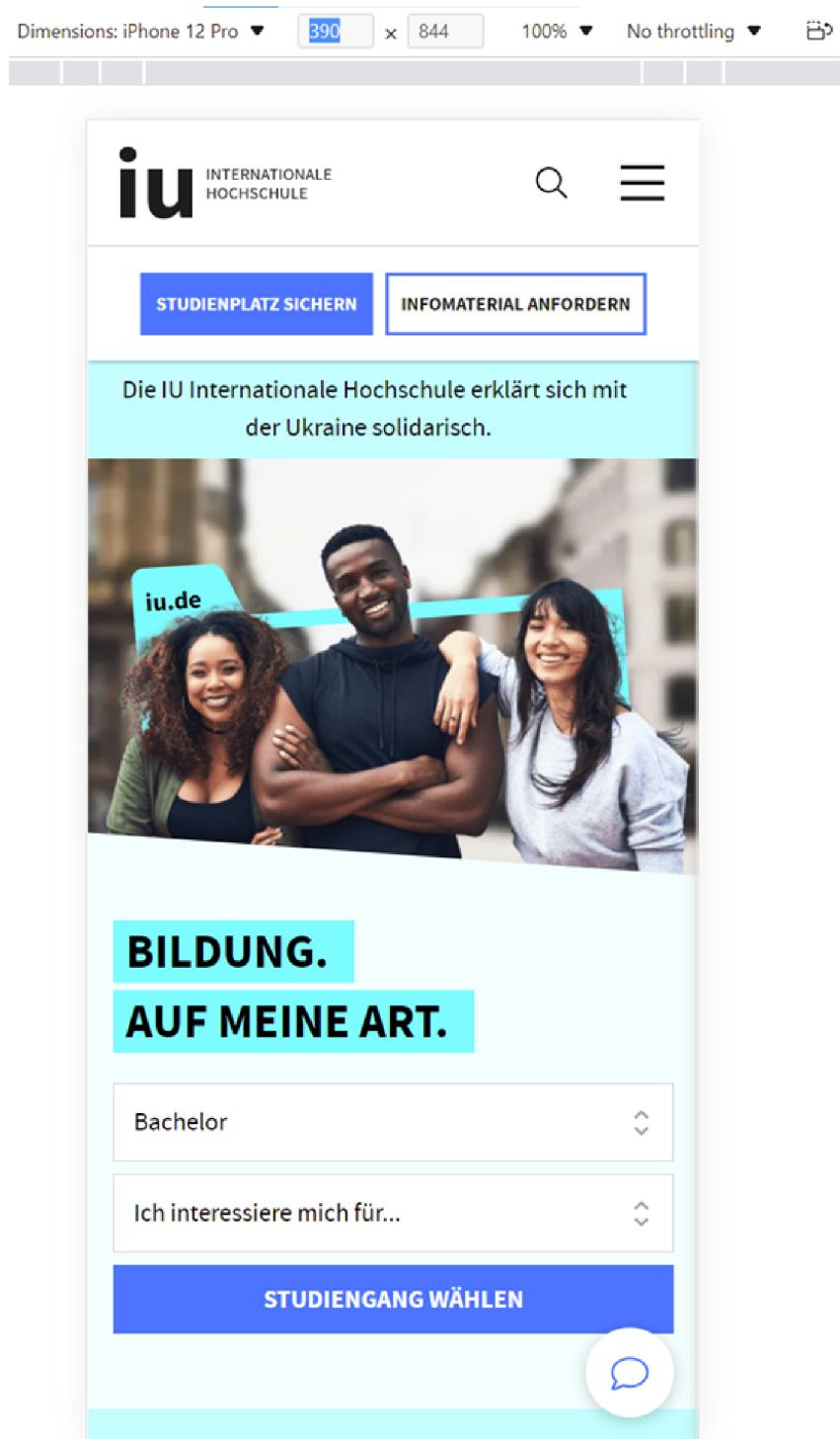
Quelle: Christian Winkler, 2022.

Die Beispielseite aus dem letzten Kapitel ist (noch) nicht responsiv, aber mit der **Homepage der IU** lässt sich die Funktion hervorragend testen.

#### IU-Homepage

Die Homepage der IU ist ein gutes Beispiel für responsives Design. Alle hier genannten Regeln werden dort eingehalten (und sogar noch mehr).

Abbildung 19: IU-Homepage in der iPhone-Emulation durch den Web Inspector



Quelle: Christian Winkler, 2022.

### **Mouseover auf mobilen Geräten**

Viele Websites nutzen Mouseover intensiv. Die dort angezeigten Informationen bleiben auf mobilen Geräten aber verborgen bzw. müssen anders vermittelt werden.

In der Browser-Leiste oben lassen sich die unterschiedlichen Endgeräte auswählen. Die „Emulation“ geht dabei sehr weit und verbirgt sogar den Mauspfeil, der dann durch ein Symbol ersetzt wird, das den Finger darstellen soll. Die bekannten **Mouseover-Effekte** funktionieren damit nicht mehr, sodass sich eine wirklich gute Testmöglichkeit für unterschiedliche Endgeräte ergibt, ohne den Browser verlassen zu müssen. Entwickler:innen spart das enorm viel Zeit.

## **4.2 Seitenlayout**

CSS ermöglicht, neben der Gestaltung der einzelnen Elemente, Einfluss auf das Seitenlayout zu nehmen. Über bestimmte Eigenschaften kann festgelegt werden, welche Elemente wie zueinander **positioniert** werden.

### **Lineares Seitenlayout**

In der ersten Version der Startup-Homepage wurde bereits ein lineares Seitenlayout verwendet. Alle Elemente erscheinen in der Reihenfolge „untereinander“, in der sie auch im HTML definiert wurden. Dafür muss kein separates CSS erstellt werden, dieses Layout wird automatisch vom Browser verwendet.

**Abbildung 20: Lineares Seitenlayout der Homepage**

---

- 
- [Instagram](#)
- [Facebook](#)
- [Kontakt](#)
  
- [Homepage](#)
- [Unsere Leistungen](#)
- [Unsere Produkte](#)
- [Unsere Preise](#)
- [Über uns](#)

## Willkommen auf der Homepage von WebAppStartup

Wir konzipieren und implementieren fantastische Web-Applikationen!

### Unsere Leistungen

Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites.

### Unsere Produkte

Wir haben einen Homepage-Baukasten entwickelt.

### Unsere Preise

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,--
Tagessatz Beratung	€ 600,--
Homepage Baukasten pro Monat	€ 1.000,--

- [Datenschutz](#)
- [Impressum](#)

---

Quelle: Christian Winkler, 2022.

Wie im Screenshot zu sehen, sieht das damit entstehende Layout sehr einfach, ja rudimentär aus. Die Website ist zwar lesbar und passt sich auch den unterschiedlichen Endgeräten an, sieht aber wenig zeitgemäß aus. Heute erwartet man von komplexen Websites, dass sich zumindest eine Navigation auf der linken Seite befindet und der Content davon abgetrennt ist. Auch die obere Navigationsleiste sollte idealerweise fest verankert und damit immer sichtbar sein.

### Floatbasiertes Seitenlayout

Dies kann z. B. über ein „float“-basiertes Layout gelöst werden. „Float“ bedeutet in diesem Fall „Fluss“ – damit lässt sich beeinflussen, welche Elemente neben- oder

### **Layout-Engine**

Damit bestimmt der Browser die Positionierung und das Styling der Elemente. Heute verwendete Layout-Engines sind sich relativ ähnlich. Das war früher anders und sorgte für erheblichen Aufwand beim Testen von Webseiten.

untereinander stehen. Die „float“-Eigenschaft kann für Block-Elemente gesetzt werden und dort unter anderem die Werte „left“ oder „right“ annehmen. So gibt die **Layout-Engine** des Browsers vor, wie die Elemente zueinander angeordnet werden sollen. „Left“ bedeutet, dass die Elemente so weit wie möglich nach links gerückt werden, dabei aber möglichst nebeneinander erscheinen. Um diese Funktionen nutzen zu können, sind Änderungen (fast) nur im head des Dokuments vorzunehmen. Das HTML muss nur minimal verändert werden – lediglich die section-Elemente sind in ein `div id="content"` zu kapseln.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Homepage WebAppStartup</title>
<style type="text/css">
  header { background-color: #afa; }
  header nav ul { text-align: right; }
  header nav ul li { display: inline; }
  main nav { width: 40 %; float: left background-color: #aaf; }
  main div#content { width: 60 %; float: left; background-color: #aff; }
  footer { clear: both; background-color: #faa; }
  footer nav ul { text-align: center; }
  footer nav ul li { display: inline; }
</style>
```

Die Farben („background-color“) sind für die Funktionalität nicht wichtig, sondern dienen lediglich der Illustration der unterschiedlichen Strukturelemente. Das mit diesen Modifikationen entstehende Layout sieht dann so aus wie in der unteren Abbildung. Da der nav-Bereich 40 % der Breite einnimmt und der `div#content` 60 % benötigt, können diese beiden Blöcke nebeneinander dargestellt werden (technisch werden sie links aneinander angelagert). Ändert man die nav-Breite auf 50 %, passt das nicht mehr nebeneinander und wird folgerichtig untereinander dargestellt. Der footer-Bereich hebt das „float“ mit der Regel „clear: both“ auf und benötigt wieder die gesamte Breite.

Abbildung 21: Float-basiertes Layout



Quelle: Christian Winkler, 2022.

Es sind noch einige andere Änderungen integriert. So wird die Top-Navigation als Liste nicht in Block-Form (also mit der gesamten Breite untereinander) dargestellt, sondern „inline“, also nebeneinander und rechtsbündig. Eine ähnliche Regel kommt im footer zum Einsatz, hier wird die Liste zentriert dargestellt.

Im Vergleich zu dem einfachen Layout ist der damit erzielte Fortschritt erheblich. „Reponsiv“ ist die Darstellung damit allerdings noch nicht, wie man mit dem Geräteemulator leicht überprüfen kann. Dazu sind noch weitere Modifikationen erforderlich, die im nächsten Kapitel besprochen werden.

## Flex und Flexbox

Im Vergleich zur Anfangszeit des World Wide Webs, wo Layouts bis etwa 2005 nur mit Tabellen konstruiert werden konnten, waren die **float-basierten Layouts** ein gewaltiger Fortschritt. Allerdings ist der Umgang mit den fließenden Block-Elementen teilweise umständlich und fehleranfällig. Einige Anforderungen können nur mit sehr großem Aufwand gelöst werden (MDN, n. d.-c):

### **Float Layouts**

Im Vergleich zu den früher verwendeten Tabellen waren Float-Layouts viel einfacher. Flexbox vereinfacht das noch weiter und bietet neue Möglichkeiten.

- Es ist sehr schwierig, Inhalte vertikal in dem übergeordneten Element zu zentrieren.
- Einem Container untergeordnete Elemente jeweils gleiche Höhen-/Breitenanteile zu gewährleisten, ist eine große Herausforderung.
- Spaltenbasierte Layouts können nur schwer mit einer einheitlichen Höhe der Spalten dargestellt werden.

Als Ersatz kommt daher eine „flex“-basierte Darstellung zum Einsatz, die diese Schwierigkeiten alle elegant lösen kann. Um Elemente als flexible Boxen anordnen zu können, muss im übergeordneten Element die Eigenschaft „display“ auf den Wert „flex“ gesetzt werden. Im Vergleich zum float-basierten Modell oben sind unabhängig davon keine Änderungen notwendig, dadurch sieht der CSS-Code etwas kompakter aus:

```
<style type="text/css">
  header { background-color: #afa; }
  header nav ul { text-align: right; }
  header nav ul li { display: inline; }
  main { display: flex; }
  main nav { width: 40%; background-color: #aaf; }
  main div#content { width: 60%; background-color: #aff; }
  footer { background-color: #faa; }
  footer nav ul { text-align: center; }
  footer nav ul li { display: inline; }
</style>
```

Das Layout selbst ändert sich dabei im Vergleich zu der vorherigen Variante kaum, lediglich der Abstand zum Footer ist anders, wie man in der unteren Abbildung erkennen kann.

Abbildung 22: Flex-basiertes Layout

---

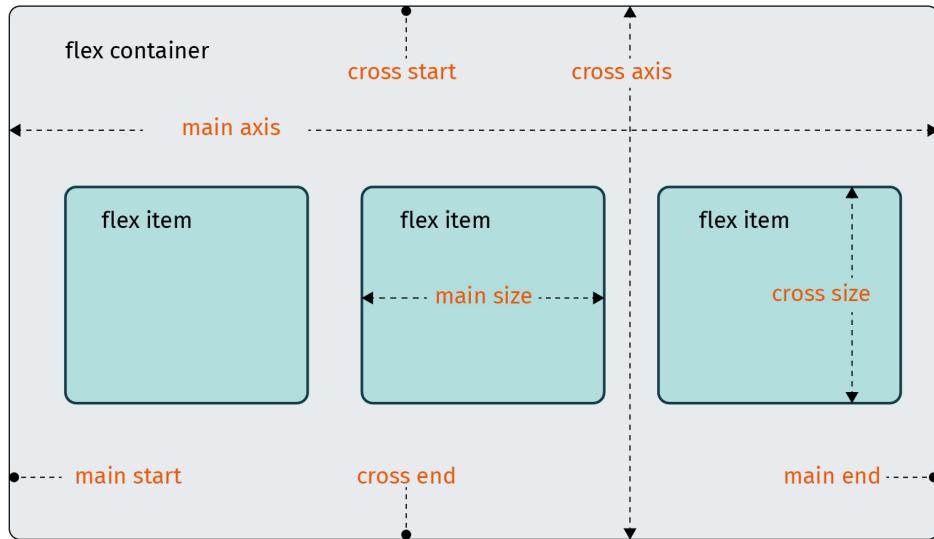


---

Quelle: Christian Winkler, 2022.

Das System kann noch deutlich flexibler eingesetzt werden. Entscheidend sind dafür die beiden Achsen, die „flex“ verwendet und die der folgenden Abbildung zu entnehmen sind.

Abbildung 23: Flex-Achsen



Quelle: MDN, n. d.-c, 2022 [CC-BY-SA 2.5].

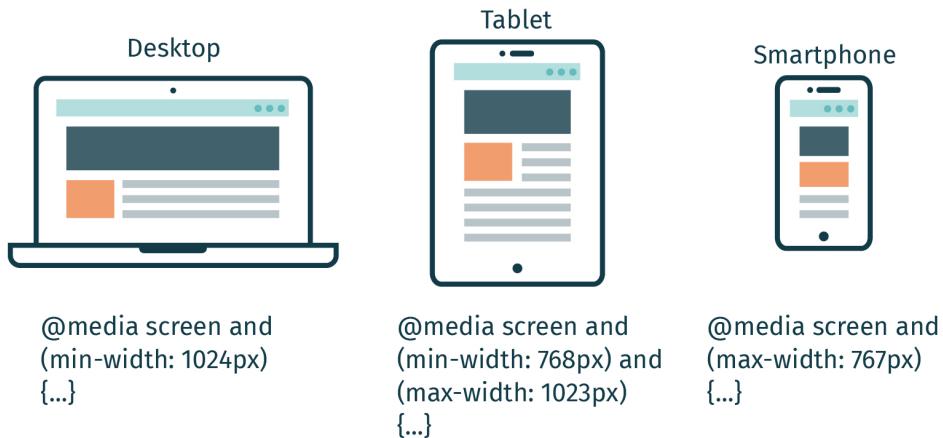
Mithilfe von „flex-direction“ lässt sich die Hauptachse beeinflussen, die standardmäßig auf „row“ gesetzt ist. So kann man mit dieser durch den Wert „row-reverse“ die Richtung ändern und eine Darstellung erreichen, in der die **Navigation rechts** steht.

Flexbox ist ein sehr mächtiges Verfahren, das den Entwickelnden große Freiheiten ermöglicht und dabei viele Probleme vermeidet, die mit float-basierten Layouts nur schwer gelöst werden können. Allerdings konnte auch durch Flexbox noch keine Darstellung erreicht werden, die sich auf unterschiedliche Endgeräte selbstständig optimiert, wie man einfach mit dem Web Inspector überprüfen kann.

## 4.3 Media Queries

Für die geräteabhängig optimierte Darstellung werden die sog. Media Queries (MDN, n. d.-g) benötigt. Es handelt sich dabei um spezielle CSS-Regeln, welche abhängig von Screen-Größen und -Orientierung bestimmte Regeln auswählen. Sie funktionieren zusätzlich zu den Selektoren als übergreifende Anweisungen und lassen auch Bedingungen zu. Als allgemeine Regeln beginnen die Queries mit dem Präfix @media. Media Queries können auch unterschiedliche Medientypen unterscheiden, also z. B. zwischen Bildschirm und Drucker. Das ist sinnvoll, wenn man beim Druck z. B. die Navigation weglassen möchte – in diesem Kurs liegt der Fokus aber auf Geräten mit Bildschirmen.

Abbildung 24: Beispielhafte Media Queries



Quelle: Seobility, n. d. [CC BY-SA 4.0].

Mit einer Media Query kann man z. B. CSS-Regeln so **filtern**, dass sie nur zutreffen, wenn die Bildschirmbreite maximal 767 Pixel beträgt, wie es z. B. bei einem Smartphone der Fall ist:

```
@media screen and (max-width: 767px) {  
    main { display: block; }  
    main nav, main div#content { width: 100 %; }  
}
```

In diesem Fall werden am Handy die Navigation und der Content in der vollen Breite dargestellt. Das lässt sich mit dem Web Inspector gut überprüfen.

#### CSS-Filter

Diese Regeln werden nur dann angewendet, wenn die Media Query zutrifft. Daher sollten sie möglichst separat im CSS stehen, um sie schnell wiederzufinden.

Abbildung 25: Simulierte Darstellung auf dem Handy

---

The image shows a simulated mobile phone screen displaying a website for "WebAppStartup". The top navigation bar is green and contains a search input field labeled "Suche", and social media links for "Instagram", "Facebook", and "Kontakt". Below the navigation is a purple sidebar menu with the following items:

- [Homepage](#)
- [Unsere Leistungen](#)
- [Unsere Produkte](#)
- [Unsere Preise](#)
- [Über uns](#)

The main content area has a light blue background. It features a large heading "Willkommen auf der Homepage von WebAppStartup" and a subtext "Wir konzipieren und implementieren fantastische Web-Applikationen!". Below this, there are three sections with bold titles: "Unsere Leistungen", "Unsere Produkte", and "Unsere Preise". The "Unsere Leistungen" section contains the text "Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites.". The "Unsere Produkte" section contains the text "Wir haben einen Homepage-Baukasten entwickelt.". The "Unsere Preise" section contains a table with the following data:

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,-
Tagessatz Beratung	€ 600,-
Homepage Baukasten pro Monat	€ 1.000,-

At the bottom of the page is a red footer bar with the links "[Datenschutz](#)" and "[Impressum](#)".

---

Quelle: Christian Winkler, 2022.

Damit ist die erstellte Webseite responsiv und zumindest für zwei Gerätetypen optimiert. 767 Pixel sind nicht zufällig gewählt. Ältere Tablets haben im Hochformat manchmal eine Auflösung von **768 Pixel** oder mehr, dafür wird man dann eigene Regeln bauen. Es genügt nicht, die Breiten auf 100 % zu setzen. Die Flexbox ist so flexibel, dass sie die Breiten der beiden Elemente dann gleich groß wählt – also werden beide nebeneinander positioniert. Daher muss die „display: flex“-Regel auch aufgehoben werden.

Über eine Kombination von „min-width“ und „max-width“ können viele unterschiedliche Endgeräte sicher abgefragt und die dafür notwendigen Regeln optimiert werden. Es hat sich als sinnvoll herausgestellt, die allgemeinen Regeln für alle Bildschirmauflösungen separat zu formulieren und nur die Änderungen für die unterschiedlichen Geräteklassen am Ende noch mal spezifisch mithilfe der Media Queries zu optimieren.

Media Queries werden vom W3-Konsortium gerade überarbeitet (W3C, n. d.-b) und unterstützen dann noch weit mehr Möglichkeiten zur Auswahl. Die Browser-Unterstützung ist dort noch sehr dünn – daher sollten diese Möglichkeiten noch nicht in Produktion verwendet werden.

**768px**  
In früheren Zeiten war eine typische Auflösung 1024x768, daher sind 768 Pixel immer noch eine verbreitete Größe zur Unterscheidung von Bildschirmauflösungen.

## 4.4 CSS-Frameworks

Da die Entwicklung eines professionellen Designs mit CSS sehr aufwändig ist und sich immer wieder ähnliche Schwierigkeiten ergeben, lohnt es sich fast immer, auf bereits existierende Frameworks zu setzen und diese den eigenen Anforderungen anzupassen. Man muss sich dann nicht mehr um das **Grid-System** und das Design für die wichtigsten User-Interface-Elemente, wie Menüs, Buttons, Eingabefelder usw., kümmern, weil das vom Framework schon mit geliefert wird. Die meisten Frameworks sind als Open Source Software kostenlos erhältlich und können auch für kommerzielle Zwecke genutzt werden. Die wichtigsten Frameworks werden im Folgenden vorgestellt.

**Grid-System**  
Für die Anordnung von Kacheln im Browser wird fast immer ein Grid-System verwendet.

### Bootstrap

Bootstrap (Bootstrap, n. d.-a) ist mit Sicherheit das bekannteste und am häufigsten verwendete CSS-Framework. Ursprünglich von **Twitter**-Entwickler:innen erdacht, wurde es von der Open Source-Community weiterentwickelt und liegt aktuell in der Version 5 vor. Bootstrap arbeitet mit einem Komponentensystem und ist flexibel konfigurierbar, sodass sich z. B. Farben beliebig anpassen lassen. Bootstrap ist hervorragend dokumentiert und es gibt viele Beispieleseiten (Bootstrap, n. d.-b). Bootstrap arbeitet mit einem Container, worin die Kacheln in Zeilen („row“) angeordnet werden. Die ursprüngliche, semantisch formulierte Homepage muss für die Darstellung mit Bootstrap strukturell modifiziert werden.

**Twitter**  
Der Kurznachrichtendienst hatte schon immer ein Interesse daran, dass seine Inhalte auf möglichst vielen Endgeräten optimal dargestellt und schnell erfasst werden können.

Zunächst muss im Kopfbereich der HTML-Datei ein Verweis auf Bootstrap eingefügt werden.

```

<head>
  <title>Homepage WebAppStartup</title>
  <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.2/dist/css/bootstrap.min.css">
</head>

```

Dafür wird für den Kopfbereich der Seite nun kein separates CSS mehr benötigt, weil alles über die CSS-Klassen von Bootstrap abgedeckt werden kann:

```

<header>
  <nav>
    <ul class="text-end">
      <li class="d-inline">
        <form>
          <input type="text" name="suche" placeholder="Suche">
        </form>
      </li>
      <li class="d-inline"><a
        href="https://www.instagram.com/WebAppStartup">Instagram</a></li>
      <li class="d-inline"><a
        href="https://www.facebook.com/WebAppStartup">Facebook</a></li>
      <li class="d-inline"><a
        href="contact.html">Kontakt</a></li>
    </ul>
  </nav>
</header>

```

„Text-end“ schiebt den Text dabei nach rechts, „d-inline“ sorgt für eine Darstellung der li-Elemente nebeneinander. Der main-Bereich muss auch modifiziert werden:

```

<div class="row">
  <div class="col-lg-4 col-sm-12">
  [...]
  </div>
  <div class="col-lg-8 col-sm-12">
  [...]
  </div>
</div>

```

Dieser Bereich ist besonders interessant. Bootstrap arbeitet mit Zeilen, dabei wird der gesamte Content in eine einzige Zeile gepackt. Innerhalb der Zeilen gibt es 12 Spalten. Für große Bildschirmauflösungen („**col-lg**“) wird eine 4:8-Aufteilung der Spalten gewählt, bei kleinen Auflösungen („col-sm“) wird die gesamte Breite genutzt. So kann sehr elegant ein responsives Layout geschaffen werden. Auch eine Verkleinerung des Browsers bewirkt eine Umstellung der Navigation und des Contents auf die volle Breite.

#### „**col-lg**“

Die Abkürzungen col-lg oder auch col-sm sowie col-md verwendet Bootstrap, um unterschiedliche Bildschirmauflösungen schnell zu referenzieren.

Abbildung 26: Bootstrap-basiertes Layout der Homepage in Desktop-Auflösung

---

Suche  
Instagram Facebook Kontakt

**Willkommen auf der Homepage von  
WebAppStartup**

Wir konzipieren und implementieren fantastische Web-Applikationen!

**Unsere Leistungen**

Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites.

**Unsere Produkte**

Wir haben einen Homepage-Baukasten entwickelt.

**Unsere Preise**

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,-
Tagessatz Beratung	€ 600,-
Homepage Baukasten pro Monat	€ 1.000,-

[Datenschutz](#) [Impressum](#)

---

Quelle: Christian Winkler, 2022.

Sehr gut funktioniert nun die Darstellung in einer reduzierten Auflösung, was sich leicht mithilfe des Web Inspectors überprüfen lässt. Die col-sm-Klassen funktionieren dort wie gewünscht und führen zu einer Darstellung der Hauptelemente untereinander, wie in der nachfolgenden Abbildung zu erkennen ist.

Abbildung 27: Bootstrap-basiertes Layout in reduzierter Auflösung

---

The screenshot shows a website for 'WebAppStartup'. At the top right is a search bar labeled 'Suche' and social media links for 'Instagram', 'Facebook', and 'Kontakt'. Below the header is a navigation menu with the following items:

- [Homepage](#)
- [Unsere Leistungen](#)
- [Unsere Produkte](#)
- [Unsere Preise](#)
- [Über uns](#)

# Willkommen auf der Homepage von WebAppStartup

Wir konzipieren und implementieren fantastische Web-Applikationen!

## Unsere Leistungen

Wir beraten bei der Erstellung von Homepages, konzipieren und implementieren Websites.

## Unsere Produkte

Wir haben einen Homepage-Baukasten entwickelt.

## Unsere Preise

Leistung	Einzelpreis
Stundensatz Beratung	€ 100,--
Tagessatz Beratung	€ 600,--
Homepage Baukasten pro Monat	€ 1.000,--

[Datenschutz](#) [Impressum](#)

---

Quelle: Christian Winkler, 2022.

### CSS-Komplexität

Bei CSS gibt es die Tendenz, immer mehr Regeln, Klassen usw. miteinander zu kombinieren, was zu unübersichtlichem und schwer wartbarem Code führen kann.

Nicht umsonst ist Bootstrap sehr beliebt. Die damit gestalteten Layouts sind kompatibel mit praktisch allen Endgeräten und sehr flexibel anpassbar. Dem steht allerdings eine sehr große **Komplexität** entgegen. Um das Layout wie gewünscht zu gestalten, sind oftmals sehr viele CSS-Klassen notwendig, was zu einer schlechten Lesbarkeit (und damit auch schwierigen Pflegbarkeit) des HTML-Codes führt. Bootstrap erlaubt auch eine Flexbox-basierte Darstellung.

Basierend auf Bootstrap gibt es sehr viele Templates, die Spezialfunktionen implementieren. Besonders hervorzuheben ist z. B. das Dashboard AdminLTE (2022), das wirklich viele Widgets anbietet und sehr professionell gestaltet ist.

### Tailwind (Tailwind CSS, n. d.)

Im Vergleich zu Bootstrap verfolgt Tailwind einen ganz anderen Ansatz. Es betont nicht die Komponenten wie Bootstrap oder fertige Layouts, sondern ist eher als „Werkzeugkasten“ gedacht, mit dem man sich alles selbst so in HTML konfigurieren kann, wie man es

möchte. Daher sind die CSS-Klassen auch sehr universell. Das hat auf der anderen Seite allerdings den Nachteil, dass erst durch die Kombination (sehr) vieler CSS-Klassen das gewünschte Layout entsteht. Dies führt zu sehr unübersichtlichem HTML-Code.

Im Gegensatz zu Bootstrap ist Tailwind außerordentlich gut an Anforderungen anpassbar, ohne dass dafür etwas am Framework selbst verändert werden muss. Auf der anderen Seite bedeutet dies aber auch, dass kein Standard-„Theme“ mit in Tailwind enthalten ist. Diese muss man sich alle selbst bauen. Deswegen ist Tailwind trotz seiner Popularität eigentlich ausschließlich für Profi-Entwickler:innen geeignet. Tailwind lässt sich auch nicht ganz einfach in existierende Applikationen einbauen, weil es die **Philosophie** verfolgt, dass nur Klassen mitgeliefert werden, die auch wirklich verwendet werden. Dies erfordert einen separaten Server, der dies erkennt und die entsprechende Auswahl trifft (inkl. der notwendigen Abhängigkeiten).

### Material Design (Material Design, n. d.)

Das Material Design-Framework orientiert sich an der Design-Philosophie, die Google sich für **Android** ausgedacht hat. Ähnlich wie bei Apps arbeitet Material Design daher sehr stark komponentenorientiert. Viele davon abgeleitete CSS-Frameworks (Materialize CSS, n. d.) nutzen auch die sog. Web Components, die über den Inhalt dieses Kurses weit hinausgehen.

Um die Homepage auf das Material Design-Layout umzustellen, sind keine großen Änderungen notwendig. Im Kopfbereich muss das CSS eingebunden werden, außerdem sind einige CSS-Regeln zu setzen, da sonst die Seitenavigation nicht richtig erscheint:

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css
      /materialize.min.css">
<style type="text/css">
  main nav { height: auto; }
  main nav li { clear: both; }
  main nav li a { color: black; }
</style>
```

Der Rest ist fast identisch zu der Bootstrap-Variante, nur die Spalten werden etwas anders bezeichnet:

```
<div class="col l4 s12">
[...]
</div>
<div class="col l8 s12">
[...]
</div>
```

Trotz vieler Versuche hat es sich als nicht möglich erwiesen, den Footer zentriert zu platzieren.

**Tailwind-Philosophie**  
Tailwind propagiert den sog. „Utility first“-Ansatz, d. h. es sind keine Standard-Layouts enthalten – diese muss man sich erst selbst bauen.

**Material Design auf Android**  
Tatsächlich ist Material Design eher in der App-Welt zu Hause als auf Websites. Es kann aber auch dort verwendet werden.

## Andere Frameworks

Es gibt noch viele andere Frameworks, die sich in bestimmten Fällen evtl. besser für eine Website eignen.

- **Foundation** (Foundation, n. d.)

Foundation gibt es fast schon so lange wie Bootstrap. Am Anfang standen die beiden Frameworks in harter Konkurrenz zueinander. Foundation ist ähnlich umfangreich wie Bootstrap, punktet aber mit etwas einfacherem Markup und weniger Klassen. Der HTML-Code sieht dadurch deutlich semantischer aus. Foundation ist wie Bootstrap sehr gut konfigurierbar und wird von vielen großen Websites eingesetzt.

- **Bulma** (Bulma, n. d.)

Bulma unterscheidet sich in einigen Aspekten von den anderen Frameworks. Es ist sehr klein, braucht kein Javascript und verfügt über wiederverwendbare Komponenten. Da es auf **SASS** und CSS-Variablen basiert, lassen sich auch Farben usw. leicht ändern und individuell anpassen. Ähnlich wie bei Tailwind ist kein Default Theme mit enthalten.

- **Pico.css** (Pico CSS, n. d.)

Pico.css ist ein Sonderfall, weil es sich um ein sog. Microframework handelt. Es kann immer dann sinnvoll eingesetzt werden, wenn man dem Layout weniger Beachtung schenkt und sich stattdessen auf das HTML konzentriert. Es sind keine Klassen notwendig, sondern die Formatierung geschieht größtenteils über die HTML-Tags. Oftmals ist es besser, pico.css statt überhaupt kein Framework zu verwenden.

### **SASS**

Hierbei handelt es sich um einen Präprozessor für CSS, womit Regeln übersichtlicher formuliert und programmiert werden können. Im Build-Prozess entsteht daraus wieder CSS.

## Kommerzielle Templates

Der Markt an kommerziellen Anbietern von Templates ist unüberschaubar groß. Fast alle Anbieter setzen dabei auf eines der oben genannten Open Source-Frameworks als Basis. Dennoch kann es oft sinnvoll sein, eines der kommerziellen Produkte zu verwenden, weil diese auch einige Vorteile bieten:

- **Professionelles Design**

Oftmals werden solche Templates von Designer:innen erstellt, die sich auch mit Typographie usw. auskennen. Dadurch entsteht ein ganz anderer Eindruck einer Website.

- **Viele Seitentypen**

Häufig sind in fertigen Templates 20 bis 60 unterschiedliche Seitentypen enthalten, die von Homepages, Landingpages, über Produktübersichten bis hin zu Blog-Beiträgen reichen.

- **Website schnell fertig**

Mithilfe derartiger Templates ist eine Website sehr schnell aufgebaut, was z. B. für Start-ups wichtig sein kann (wenn sie sich nicht gerade selbst mit der Entwicklung von Websites beschäftigen).



## ZUSAMMENFASSUNG

Wie Browser aus einer HTML-Seite und dem dazugehörigen CSS ein Layout berechnen, ist ein komplexer Prozess. Grundsätzlich unterscheiden die Layout-Engines zunächst zwischen den Block- und Inline-Elementen, die anderen Darstellungsregeln unterworfen werden. Allein damit ließe sich aber kein Layout erzeugen, was den Ansprüchen der Benutzenden an Bedienbarkeit und Ästhetik entspricht.

Mit der „float“-Eigenschaft steht eine Möglichkeit zur Verfügung, die Reihenfolge der Darstellung von Block-Elementen zu beeinflussen. Diese können je nach Platz dann nebeneinander oder übereinander dargestellt werden. Lange Zeit war das auch die bevorzugte Möglichkeit, komplexe Web-Layouts zu realisieren. Allerdings kämpft das Verfahren mit einigen Unzulänglichkeiten, die durch die Flexbox behoben wurden. Viele moderne Websites oder Frameworks setzen daher heute auf das Flexbox-Verfahren.

Neben dem Desktop-Browser gibt es sehr viele andere Endgeräte, wie etwa Smartphones, Tablets oder auch Fernseher. Aus Aufwandsgründen kann nicht für jedes Endgerät eine eigene Website erstellt werden. Mit Hilfe der CSS Media Queries können spezielle Regeln für diese Endgeräte angegeben werden, mit deren Hilfe sich eine optimierte Darstellung umsetzen lässt. So kann z. B. auf dem Smartphone die gesamte Breite für den Content genutzt werden, während sich auf dem Desktop links daneben noch eine Navigation befindet.

CSS-Frameworks nehmen den Entwickelnden viel dieser Handarbeit ab und arbeiten meistens mit einem Gitter/Raster, auf dem sich die Elemente anordnen lassen. Die Breite kann dann endgeräteabhängig gewählt werden, sodass sich eine optimierte Darstellung ergibt. Das beliebteste Framework ist Bootstrap, wobei auch weitere, wie Tailwind, Foundation und Bulma, existieren. Für die Entwicklung von Web-Apps finden sich spezielle Frameworks, wie Material Design, die der Web-App ein (nahezu) natives Aussehen verleihen.



# LEKTION 5

## WEBSEITENENTWICKLUNG MIT JAVASCRIPT

### LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- einfache Programme in Javascript zu schreiben.
- Javascript in Webseiten einzubinden, um diese dynamisch zu machen.
- JavaScript-Bibliotheken in Webseiten zu verwenden.
- JSON-Daten einzulesen und zu verarbeiten.

## 5. WEBSITE-NENTWICKLUNG MIT JAVASCRIPT

### Einführung

Zu Beginn des World Wide Web waren die meisten Seiten statisch. (Private) Homepages dominierten das neue Medium. Bereits am Anfang gab es Interaktionsmöglichkeiten, die über das reine Anklicken von Links hinausgingen. SPIRES (SPIRES, n. d.) war eine der ersten solcher dynamischen Webseiten und ließ die Suche nach wissenschaftlichen Publikationen zu. Sobald man einen Suchbegriff eingegeben hatte, wurde dieser zum Server geschickt. Als Antwort erhielt man dann eine Seite mit den Suchergebnissen.

Für viele Anwendungen ist dieser sog. Request-Response-Cycle gut geeignet. Allerdings musste dazu immer eine neue Webseite aufgebaut werden. Das dauert eine ganze Weile und ist nicht praktisch, wenn man nur kleinere Operationen an einer bereits geladenen Seite vornehmen möchte, wie z. B. das Ausklappen einer Navigation. Idealerweise sollte das gleich der Browser selbst übernehmen. Anfangs war das nicht möglich, weil die Browser nicht programmierbar waren.

Die Notwendigkeit der Programmierbarkeit erklärte 1995 zuerst der damals größte Browserhersteller Netscape und stellte in seinem Browser die erste Version einer Programmiersprache zur Verfügung, die heute unter dem Namen JavaScript weltweit bekannt ist. Damals wurde sie recht hastig implementiert. Sogar noch heute muss man mit vielen Unzulänglichkeiten der Sprache kämpfen, die damals nicht hinreichend bedacht wurden. Heute ist JavaScript trotz allem aus der Mehrzahl der modernen Webanwendungen nicht mehr wegzudenken – es ist die am weitesten verbreitete Programmiersprache überhaupt.

### 5.1 JavaScript-Geschichte, ES5/ES6

**Interpreter-Sprachen**  
Solche Programmiersprachen werden zur Laufzeit übersetzt. Diese Programme laufen etwas langsamer, aber das Ergebnis ist im Gegensatz zu Compiler-Sprachen sofort sichtbar.

JavaScript ist eine **Interpreter-Sprache**, die 1995 vom Browser-Hersteller unter dem Namen „Livescript“ (Web Archive, 1995) erfunden wurde, um Webseiten interaktiver zu machen. JavaScript hat nichts mit der Programmiersprache Java zu tun, es wurde nur aus Marketingzwecken so umbenannt. Zu Beginn wurde JavaScript hauptsächlich dazu verwendet, um Grafiken zu animieren und Formulare zu validieren. Im Laufe der Zeit wuchsen die Fähigkeiten. Seit 2009, mit dem Erscheinen von „Node.js“, steht JavaScript auch als serverseitige Programmiersprache zur Verfügung und wird in vielen Projekten verwendet. Viele Web-Anwendungen laufen heute mittels eines **JavaScript-Frameworks** im Browser. In diesem Fall dient der Server oft nur noch zur Datenspeicherung und Authentifizierung.

**Tabelle 1: Geschichte von JavaScript**

Jahr	Ereignis	Quelle	JavaScript-Framework
1995	JavaScript wird vom Browser-Hersteller „Netscape“ auf den Markt gebracht. Es soll Webseiten interaktiver machen und eine Alternative zu Java-Applets bieten.	(Web Archive, 1995)	Ein JS-Framework ist eine Code-Sammlung, die das Entwickeln von Web-Applikationen vereinfachen soll. Es behebt die Inkompatibilität zwischen verschiedenen Browsern und vereinfacht die Programmierung.
1997	JavaScript wird unter dem Namen „ECMAScript“ standardisiert.		
1999	ECMAScript Version 3 erscheint und bringt als Neuerung „reguläre Ausdrücke“ sowie eine Fehlerbehandlung mit try/catch.	(MDN, n. d.-d)	
2005	Der Internet Explorer von Microsoft erhält die Möglichkeit eines XMLHttpRequests. Obwohl nicht dafür gedacht, werden dadurch völlig neue Web-Applikationen möglich, die nur einen Teil der Seite neu laden. Die Technologie wird AJAX (Asynchronous JavaScript And XML) genannt und ist bis heute sehr gebräuchlich.	(Microsoft, n. d.) (adaptive path, n. d.)	
2006	Das JavaScript-Framework „jQuery“ erscheint und erleichtert die Entwicklung von Web-Applikationen drastisch.	(Resig, n. d.)	
2009	ECMAScript 5/ES5 verbessert die Code-Qualität durch den „strict mode“, führt „Getter & Setter“ ein, bringt nützliche Funktionen für Arrays und Unterstützung für „JSON“.	(ECMAScript, 2011, S. 262)	
2009	Node.js erscheint und ermöglicht serverseitige Programmierung mit JavaScript. Mittlerweile hat es sich zu einem großen Player bei den Server-Sprachen entwickelt.	(Node JS, n. d.)	
2010	Das JavaScript-Framework „Angular.js“ von Google erscheint und ermöglicht umfangreiche, komplexe Web-Applikationen, die über die Möglichkeiten von „jQuery“ hinausgehen. Es ist relativ schwer, zu nutzen und erfordert eine genaue Planung.	(Angular, n. d.-b)	
2012	Microsoft bringt mit Typescript eine statisch typisierte Version von JavaScript auf den Markt. Angular.js wird fortan in Typescript entwickelt.	(Turner, 2014)	
2013	Das JavaScript-Framework „React.js“ von Facebook erscheint und verbindet JavaScript und HTML in sog. Komponenten.	(JSConf, 2013)	
2014	Das JavaScript-Framework „Vue.js“ erscheint und vereint die gute Strukturierung von „Angular“ mit der Einfachheit von „React“.	(egghead.io, n. d.)	
2015	Nach sechs Jahren erscheint ECMAScript 2015/ES6 und bildet die JavaScript-Grundlage der heutigen Browser.	(ECMAScript, 2015)	

Jahr	Ereignis	Quelle
2016	ECMAScript 7 bzw. 2016 bringt Verbesserungen bei asynchronen Serveraufrufen mit „async/await“.	(ECMAScript, 2016)
2019	Der Standard für WebAssembly wird offiziell. WebAssembly erlaubt auch die Programmierung von Browsern, ist darin jedoch wesentlich schneller als Javascript und erlaubt keine Modifikationen oder Kopien des Codes.	(Web Assembly, n. d.)
2021 ff.	Neuere Versionen von ECMAScript spielen für dieses Skript vorerst keine Rolle.	

Quelle: Christian Winkler, 2022, in Anlehnung an die in der Quellenspalte genannten Quellen.

## 5.2 JavaScript-Grundlagen

Die ursprüngliche Aufgabe von JavaScript war es, HTML-Seiten dynamisch und interaktiv zu machen. Dieser Aufgabe wird sie auch heute noch gerecht, auch wenn die Sprache über die Jahre gewachsen ist und viel mehr Anforderungen abdecken kann. Da jeder **Webbrowser** JavaScript versteht, ist es eine nahezu universelle Programmiersprache, die überall ablauffähig ist.

**Webbrowser und JavaScript**  
Da jeder Webbrowser JavaScript ausführen kann, ist JavaScript die Sprache, die auf den meisten Geräten der Welt genutzt werden kann.

Technisch gesprochen ist JavaScript eine dynamisch typisierte, interpretierte Sprache. Man muss Programme nicht umfangreich vorbereiten, sondern kann einfach loslegen. Leider hat das zur Folge, dass Programme auch fehlerhaft sein können und man die Fehler erst spät entdeckt. Daher spielt das Testen bei JavaScript eine überaus bedeutsame Rolle.

Normalerweise werden JavaScript-Programme mit dem `script`-Tag in HTML eingebunden. Das eigentliche Programm kann dann innerhalb des Elements stehen, oder über das „src“-Attribut vom Server geladen werden. Besonders bei größeren Programmen ist letztere Variante viel praktischer. JavaScript-Programme können sehr umfangreich werden. Wenn dann auf jeder Seite das gesamte Programm steht, werden auch die HTML-Seiten groß – wird hingegen nur darauf verwiesen, kann der Browser sich das Programm merken (cachen).

**C**  
Die Sprache C ist eine Hochsprache, die in den 1970er-Jahren erfunden wurde, um das Betriebssystem Unix zu programmieren. C war wegweisend – viele heutige Sprachen ähneln C.

Die Syntax von JavaScript ist an die Sprache „C“ (Ritchie, 1993) angelehnt, es gibt aber auch einige Unterschiede. Insgesamt kann man den Umgang mit JavaScript schnell erlernen, es gibt allerdings sehr viele Subtilitäten, denen man immer wieder begegnet.

### Ausprobieren!

JavaScript lässt sich sehr einfach ausprobieren, weil es im Browser integriert ist. Auch hier leistet der Web-Inspector hervorragende Dienste. In diesem Fall sollte die „Console“ aufgerufen werden, in der sich direkt Befehle eingeben lassen. Das ist sehr praktisch, weil dort immer der Wert des letzten Ausdrucks ausgegeben wird. JavaScript beherrscht die elementaren Rechenoperationen:

```
> 3+5
< 8
> 3*5
< 15
> 3**5
< 243
> Math.cos(0)
< 1
```

Das ist nützlich für einen Browser, aber eigentlich sollen ja damit die HTML-Dokumente interaktiver werden. Um das zu erreichen, muss JavaScript mit dem Browser interagieren – dazu dient das **DOM** (Document Object Model). Am besten funktioniert das mit einer der bereits erzeugten Webseiten. Aus den vorangegangenen Lektionen sollte bekannt sein, dass HTML-Dokumente hierarchisch aufgebaut sind und eine Art Baumstruktur darstellen: Das äußerste Element (die Wurzel) ist immer `html`. Es gibt mehrere Möglichkeiten, bestimmte Elemente im HTML-Dokument zu selektieren, z. B. nach dem Tagnamen:

```
> document.getElementsByTagName("h1")
< HTMLCollection { 0: h1, length: 1 }
```

Tagnamen sind nicht eindeutig, deswegen liefert die Funktion eine „HTMLCollection“ zurück, d. h. eine Liste von Ergebnissen. In diesem Fall besteht die Liste nur aus einem einzigen Element, das sich mit dem Index [0] referenzieren lässt.

```
> document.getElementsByTagName("h1")[0]
< <h1>
```

Auch der Inhalt des Elements lässt sich auswählen, über die **innerHTML**-Eigenschaft:

```
> document.getElementsByTagName("h1")[0].innerHTML
< "Willkommen auf der Homepage von WebAppStartup"
```

JavaScript ist eine interaktive Programmiersprache, demnach lassen sich auch Werte damit überschreiben:

```
> document.getElementsByTagName("h1")[0].innerHTML = "neuer Text"
< "neuer Text"
```

Die bereits geladene Webseite ändert sich damit. Das geht äußerst schnell – ein großer Vorteil von JavaScript, der dazu geführt hat, dass sich viele Web-Applikationen heute wie „normale“, native Programme anfühlen.

Um JavaScript wirklich verstehen zu können, muss man sich der Sprache systematisch nähern. Darum soll es in den folgenden Punkten gehen.

**DOM**  
Mit DOM ist eine Baumstruktur gemeint, die der Browser für jede HTML-Seite aufbaut.

**innerHTML**  
Damit kann der HTML-Inhalt eines Elements gesetzt werden.

## Grammatik

### Unicode

Ein Unicode ist ein Zeichensatz, mit dem sehr viele unterschiedliche Zeichen dargestellt werden können, auch die von exotischen Sprachen.

JavaScript beachtet Groß- und Kleinschreibung und nutzt **Unicode** (Unicode Home, n. d.).

Damit sind auch folgende Statements möglich:

```
const möglich = true;  
const laughing = "😂";
```

Es gibt bestimmte Konstruktionsregeln für Variablen, diese müssen mit einem Buchstaben, \_ oder \$ beginnen. Die Variablen „Möglich“ oder „Laughing“ wären wegen der Groß-/Kleinschreibung also andere Variablen als die beiden oben abgebildeten. Wie bereits zu sehen, werden Befehle in JavaScript mit einem Semikolon beendet. Dieser kann theoretisch entfallen, wenn der nächste Befehl in einer neuen Zeile beginnt – das sieht aber unschön aus und sollte vermieden werden. Neuere JavaScript-Versionen haben noch mehr Regeln, wann der Strichpunkt entfallen kann – wenn man ihn setzt, ist man auf der sicheren Seite.

Der Quellcode von JavaScript wird von links nach rechts ausgewertet. Leerzeichen können fast immer durch Tabulatoren oder Zeilenumbrüche ersetzt werden. Ein doppelter Schrägstrich „//“ leitet einen Kommentar ein. Der Rest der Zeile wird dann ignoriert.

## Variablen

### Deklaration

Darunter versteht man in Programmiersprachen das Bekanntmachen von Programmelementen, wie bspw. dem Namen, bevor diese Elemente in anderen Programmteilen verwendet werden können.

Variablen sind ein zentrales Element fast jeder Programmiersprache. Dort werden Werte gespeichert, die (teilweise) auch verändert werden können. Um Variablen nutzen zu können, müssen sie zunächst **deklariert** werden. Dazu gibt es in JavaScript unterschiedliche Möglichkeiten, die im Anschluss vorgestellt werden.

### const

Damit wird eine Konstante definiert. Dieser Wert steht ab der Deklaration zur Verfügung und kann dann nicht mehr verändert werden. Beispiel:

```
> const pi = 3.14159;
```

### let

Mit „let“ wird eine Variable deklariert, die in dem aktuellen Block gilt und verändert werden darf. Vereinfacht gesprochen ist ein Block die innerste Ebene, die durch die geschweiften Klammern {} definiert wird. Die Variable steht ab der Deklaration zur Verfügung:

```
> let visible = true;
```

## var

Ähnlich wie mit „let“ wird auch mit „var“ eine Variable definiert, die verändert werden kann. Allerdings gilt diese nicht nur im Block, sondern in der aktuellen Funktion. Etwas irritierend ist, dass sie auch schon vor der Deklaration verwendet werden kann, auch wenn ihr Wert dann noch nicht definiert ist. Nicht nur aus diesem Grund sollte man „var“ heute nicht mehr verwenden:

```
> console.log(a); var a = 5;  
< undefined
```

Grundsätzlich müssen in JavaScript keine Datentypen für Variablen festgelegt werden. Als dynamisch typisierte Sprache bestimmt JavaScript die richtigen Typen zur Ablaufzeit des Programms. Das ist für die Programmierung sehr komfortabel, kann aber auch Fehler hervorrufen, die über Tests rechtzeitig abgefangen werden müssen.

## Datentypen

Variablen gibt es in unterschiedlichen Typen. Zahlen müssen anders dargestellt werden als Zeichenketten. JavaScript kennt relativ wenige Datentypen. Man muss sie, im Gegensatz zu anderen Programmiersprachen, auch nicht mit einem eigenen Datentyp kennzeichnen, sondern sie ergeben sich aus dem zugewiesenen Wert zur Laufzeit (**dynamische Typisierung**). Die folgenden Variabtentypen sind in JavaScript verfügbar (siehe Tabelle).

Tabelle 2: Datentypen in Javascript (Auszug)

Variabtentyp	Beschreibung	Beispiel-Code
Number	Damit werden Zahlen (Ganz- und Dezimalzahlen) gespeichert. Wichtig: Dezimalzahlen werden amerikanisch mit Punkt „.“ und nicht mit Komma „,“ getrennt.	let preis = 800.0;
BigInt	Damit können beliebig lange Ganzahlen abgespeichert werden (wie hier die 9. Mersenne-Primzahl).	let p9 = 2305843009213693951;
String	Damit werden Wörter und Zeichenketten gespeichert. Der eigentliche Text steht zwischen zwei Anführungszeichen.	let firma = "WebAppStartup";
Boolean	Dieser Wahrheitstyp kann nur zwei Zustände annehmen: true oder false.	let online = true;
Object	Dies ist der komplexeste und am häufigsten verwendete Datentyp von JavaScript. Man kann sich Objekte wie „Container“ vorstellen, in denen mehrere unterschiedliche Variablen mit Schlüsseln gespeichert werden. Sie können hierarchisch gruppiert werden. Im Gegensatz zu den anderen Variabtentypen werden hier nur Referenzen gespeichert, d. h. Änderungen betreffen alle Instanzen.	let firma = { name: "Startup", online: false };

### Typisierung

Man unterscheidet zwischen dynamischer und statischer Typisierung. Letztere wird fast nur in übersetzten Sprachen verwendet.

Quelle: Christian Winkler, 2022, in Anlehnung an (MDN, n. d.-e).

Es gibt noch mehr Variabtentypen ( `null`, `undefined`, `symbol` [Wikipedia, 2022]), die für diesen Kurs nicht relevant sind. In der Tabelle oben scheint eine Liste („Array“) als Datentyp zu fehlen, diese wird in JavaScript auch durch Object dargestellt.

## Objekte und Arrays

Objekte sind die zentralen Datenstrukturen in JavaScript und werden in nahezu allen Programmen sehr intensiv verwendet. In einem Objekt werden Eigenschaft-Wert-Paare gespeichert. Man kann sich ein Objekt als Sammlung („Collection“) von Eigenschaften vorstellen. Über die Eigenschaften kann man deren Werte auslesen oder setzen.

```
> let firma = {  
    name: "Startup",  
    online: false  
};  
firma  
< {name: 'Startup', online: false}  
> firma["online"]  
< false
```

Statt die Eigenschaft in Klammern zu schreiben, kann man sie auch mit einem Punkt dahinter setzen. `firma["online"]` entspricht also `firma.online`. Das geht allerdings nur für Eigenschaften, die keine speziellen Zeichen enthalten.

Unbedingt beachtet werden muss bei der Arbeit mit Objekten, dass eine Zuweisung keine Kopie erzeugt, sondern nur eine **Referenz**.

### Referenz

Eine Referenz ist ein Verweis auf ein Objekt.

```
> let startup = firma;  
startup["online"] = true;  
firma["online"]  
< true
```

Obwohl scheinbar nur die Eigenschaft von `startup` modifiziert wurde, hat sich die Eigenschaft von `Firma` ebenfalls geändert. Das liegt daran, dass `firma` und `startup` nur Referenzen (unterschiedliche Namen) für das gleiche Objekt sind.

Arrays sind eine spezielle Art von Objekten, die man sich wie eine Liste vorstellen kann. Im einfachsten Fall enthalten sie z. B. Zahlen (siehe Beispiel unten). Sie können aber auch andere Objekte enthalten. Arrays lassen sich direkt in JavaScript definieren und haben einige sehr nützliche Funktionen.

```
> let list = [0, 1, 2]; list.length  
< 3  
> list[0]  
< 0
```

```
> list.push(3)
< 4
> list
< Array(4) [ 0, 1, 2, 3 ]
```

Arrays werden mit eckigen Klammern definiert. Über „length“ lässt sich die Anzahl der Werte ermitteln. Die eckigen Klammern dienen (wie bei Objekten) dem Auswählen der einzelnen Elemente. Dazu muss man nur den Index des gewünschten Elements angeben. Der Index stellt eine Art Adresse dar, die jedes Element im Array besitzt. Das erste Element besitzt immer den Index 0, das zweite den Index 1 usw. `list[2]` liefert also das dritte Element. Arrays sind veränderlich, so kann über „push“ ein Wert angehängt werden. Nicht alle Werte in Arrays müssen den gleichen Typ haben (dafür gibt es mit `ArrayBuffer` andere Objekte, die dies forcieren).

## Fallunterscheidungen/Alternativen

Programme sind dann besonders spannend, wenn sie verschiedene Fälle unterscheiden und z. B. abhängig vom Wert einer Variablen unterschiedliche Aktionen durchführen können. Für solche Fallunterscheidungen gibt es in JavaScript (wie in sehr vielen anderen Programmiersprachen auch) die if-Anweisung. Eine if-Anweisung besteht aus einer Bedingung und einem Teil, der ausgeführt wird, wenn die Bedingung zutrifft sowie aus einem Part, der ausgeführt wird, wenn die Bedingung nicht zutrifft. Die jeweiligen Teile werden in geschweiften Klammern geschrieben und bilden einen sog. **Block**:

```
if (bedingung) {
  //wird ausgeführt, falls die Bedingung zutrifft
} else {
  //wird sonst ausgeführt
}
```

In den meisten Programmiersprachen muss die Bedingung ein logischer Ausdruck sein, der entweder `true` oder `false` ist. In JavaScript ist das nicht so. Die Bedingung trifft hier nicht zu, wenn sie `false`, `null`, `0` oder `undefined` ist. Alles andere führt dazu, dass er erste Code-Block ausgeführt wird. Das ist im ersten Augenblick ungewohnt, stellt sich aber als sehr komfortabel heraus.

Um Fallunterscheidungen wirklich nutzen zu können, muss es unterschiedliche Fälle geben. Das ist meist dann der Fall, wenn Benutzerinteraktionen stattfinden. Dies ist mit der JavaScript-Konsole allein nicht möglich, sondern erfolgt am besten direkt in der Webseite. Nun ist Interaktion gefordert. Des schöneren Layouts wegen lohnt es sich, die Bootstrap-Variante zu nutzen und den Code (vor `</body>`) zu ergänzen (siehe folgendes Beispiel).

```
<script type="text/javascript">
  document.querySelector("form").onsubmit = function() {
    //Wert des Suche-Felds bestimmen
    let suche = document.querySelector("input").value;
```

### Block

Dieser Begriff bezeichnet mehrere Code-Zeilen, welche in sich abgeschlossen sind. Sie beginnen mit „{“, gefolgt von den Code-Zeilen und enden mit „}“. Die mit darin deklarierten Variablen sind nur innerhalb dieses Blockes gültig, nicht mehr außerhalb.

```

if (suche.length < 3) {
    alert("zu kurzer Suchbegriff");
} else {
    //style ist ein Object, daher einfach eine Eigenschaft setzen
    let h1 = document.querySelector("h1");
    h1.innerHTML = "Noch nicht unterstützt";
    h1.style["background-color"] = "green";
}
return false;
}
</script>

```

Im Gegensatz zum bisherigen Code wird hier immer die Methode „document.querySelector“ aufgerufen. Hier können CSS-Selektoren verwendet werden und die Methode liefert nur den ersten passenden Eintrag – sehr praktisch! Das Formular erhält über **onsubmit** eine Funktion, die aufgerufen wird, wenn es (mit Return) abgeschickt wird. Normalerweise öffnet sich in einem solchen Fall eine neue Seite, was durch das „return false“ ganz am Ende verhindert wird. Stattdessen wird der eingegebene Suchbegriff ermittelt und bei einer zu kleinen Länge eine Fehlermeldung (mit „alert“) angezeigt. Eine Suche selbst durchzuführen ist schwierig, stattdessen wird einfach die Überschrift geändert und mit grüner Farbe hinterlegt. Das ist in echten Programmen nicht sinnvoll, zeigt aber viel von der dort auch verwendeten Logik, dem Auslesen von Daten und dem Setzen von Eigenschaften.

### Onsubmit

Bevor ein Formular abgeschickt wird, wird die onsubmit-Methode aufgerufen. Damit kann z. B. die Vollständigkeit der Eingabe überprüft werden.

### JavaScript-Konsole

Dieses Ein-/Auszugfenster erlaubt den Einblick in die „internen Zustände“ der JavaScript-Maschine. Es kann im Browser angezeigt und über Befehle wie „console.debug()“ gesteuert werden.

## Vergleichs-Operatoren

Oben war bereits ein Vergleich mit „<“ zu finden. JavaScript unterstützt aber noch weitere Vergleichs-Operatoren, mit deren Hilfe sich Variablen miteinander (oder mit Konstanten) vergleichen lassen. Vergleichs-Operatoren ergeben immer einen Wahrheitswert, also true oder false. Mithilfe von Bedingungen ermöglichen sie dadurch das Programmieren von Regeln. Die Operatoren lassen sich hervorragend in der **JavaScript-Konsole** testen.

**Tabelle 3: Vergleichs-Operatoren**

Operator	Beschreibung	Beispiel-Code
==	prüft, ob die beiden Seiten gleich sind	suche == "wert"
===	prüft, ob die beiden Seiten identisch sind	suche === "wert"
!=	prüft, ob die beiden Seiten ungleich sind	suche != "wert"
!==	prüft, ob die beiden Seiten nicht identisch sind	suche !== "wert"
<	prüft, ob die linke Seite kleiner ist als die rechte	3 < 5
<=	prüft, ob die linke Seite kleiner oder gleich der rechten ist	5 <= 4
>	prüft, ob die linke Seite größer ist als die rechte	3 > 5

Operator	Beschreibung	Beispiel-Code
<code>&gt;=</code>	prüft, ob die linke Seite größer oder gleich der rechten ist	<code>5 &gt;= 4</code>

Quelle: Christian Winkler, 2022.

Es gibt hier einige Besonderheiten zu beachten.

- Die Größer-/Kleiner-Operatoren können auch für Zeichenketten verwendet werden. Es sind sogar **Zeichenketten mit Zahlen** vergleichbar. Dabei werden dann die Zeichenketten in Zahlen gewandelt. Die Regeln dafür sind sehr komplex – daher sollte man dies möglichst vermeiden.
- JavaScript unterscheidet zwischen Gleichheit und Identität. In den meisten Fällen spielt das keine Rolle, es gibt aber subtile Unterschiede. So sendet der Vergleich „`null == undefined`“ den Wert „`true`“, während „`null === undefined`“ den Wert „`false`“ liefert. Im Allgemeinen ist es daher immer sicherer, mit den Identitäten zu arbeiten, auch wenn das von den meisten JavaScript-Programmen leider nicht eingehalten wird.

#### Zeichenketten mit Zahlen

Die dynamische Typisierung macht es hier manchmal schwierig. Was passiert, wenn man zu einer Zahl ein Zeichen dazu zählt oder umgekehrt? Am besten gilt es, solche Fälle zu vermeiden.

## Logische Verknüpfungen

Häufig reicht eine einzige Bedingung nicht, um die gewünschten Abfragen durchführen zu können. Selbstverständlich lassen sich die if-Anweisungen auch schachteln, aber das kann schnell unübersichtlich werden. Es ist oft einfacher, die Bedingungen miteinander zu **verknüpfen**. Dazu bietet JavaScript unterschiedliche Operatoren an. Die folgende Tabelle gibt einen Überblick.

Tabelle 4: Logische Verknüpfungen

#### Verknüpfungen

Diese Operatoren werden auch als Bool'sche Verknüpfungen bezeichnet. Ähnlich kommen sie auch in der Mengenlehre vor.

Verknüpfung	Beschreibung	Beispiel-Code
<code>!</code>	NICHT: Die Bedingung nach dem „ <code>!</code> “ darf nicht zutreffen.	<code>! (suche === "wert")</code>
<code>&amp;&amp;</code>	UND: Beide Teilbedingungen müssen zutreffen, damit die Gesamtbedingung zutrifft.	<code>suche === "wert" &amp;&amp; online === true</code>
<code>  </code>	ODER: Es genügt, wenn eine der beiden Bedingungen zutrifft, damit die Gesamtbedingung zutrifft.	<code>suche === "leer"    suche.length &lt; 3</code>

Quelle: Christian Winkler, 2022.

## Schleifen

Schleifen in Programmiersprachen sind „Wiederholungen“ – ein bestimmter Befehls-Block wird wiederholt. Sie werden hauptsächlich mit Arrays und **Listen** verwendet, um bestimmte Befehle mit deren einzelnen Elementen auszuführen. Schleifen sind eines der wesentlichen Strukturelemente von Programmiersprachen und essenziell für die Automatisierung von Aufgaben. JavaScript unterstützt unterschiedliche Arten von Schleifen:

#### Liste

Dieser Begriff kann in JavaScript synonym zu Array verwendet werden.

### • Klassische for-Schleife

Diese Schleife ist in fast allen C-ähnlichen Programmiersprachen vorhanden. Dabei wird eine sog. Index-Variable über einen bestimmten Bereich gezählt. Eine Bedingung entscheidet, wann die Schleife abgebrochen wird (wenn die Bedingung schon zu Beginn nicht erfüllt ist, wird die Schleife gar nicht ausgeführt). Die Syntax für die Schleife sieht folgendermaßen aus:

```
for (initialisierung; bedingung; änderung) { ... }
```

Die Schleife wird so lange ausgeführt, wie die Bedingung erfüllt ist. Fast immer wird in „änderung“ die Schleifenvariable selbst modifiziert und in „bedingung“ abgefragt – das ist aber kein Muss. Beispiel (darin ist auch „console.log“ eingefügt):

```
for (let i = 0; i < 10; i++) { console.log(i) }
```

### • for-of-Schleife

Oft ist man eher an den Elementen eines Arrays interessiert, für die z. B. bestimmte Operationen durchgeführt werden sollen. Das ist mit der klassischen Schleife auch möglich, allerdings etwas umständlich. Daher gibt es in (neueren Versionen von) JavaScript noch eine weitere Form der Schleife, die das in einem Rutsch erledigt. Beispiel:

```
for (e of [1, 4, 9, 16, 25]) { console.log(Math.sqrt(e)) }
```

### • while-Schleife

Diese Schleife enthält nur eine Bedingung, die schon beim ersten Durchlauf überprüft wird. Die Schleife wird so lange ausgeführt, bis die Bedingung nicht mehr zutrifft. Beispiel:

```
let i = 1; while (i < 1000) { console.log(i); i = i *2}
```

## Funktionen

Funktionen werden in Programmiersprachen eingesetzt, um Code-Wiederholungen zu vermeiden. Warum sollte man das tun? Oft werden Programmfragmente lediglich mit leichten Abweichungen verwendet. Anstatt den Code zu wiederholen, ist es besser, eine Funktion zu schreiben und die Abweichungen als **Parameter** zu übergeben. Das hat einige Vorteile:

- Der Funktions-Code liegt zentral vor und muss bei Änderungen lediglich an einer einzigen Stelle angepasst werden.
- Wenn der gleiche Code zehnmal verwendet wird, muss er nicht zehnmal im Programm stehen.
- Ein großes, komplexes Programm kann in viele, einfache Funktionen zerlegt werden.

Genau wie mathematische Funktionen können auch solche in JavaScript Werte zurückliefern (mit `return`). Eine Funktion kann wie folgt definiert werden:

```
function name(parameter1, parameter2) {  
    //hier stehen die Befehle  
    //...  
    //wert wird zurückgegeben, nur mit „return“ keine Rückgabe  
    return wert;  
}
```

Diese **Funktion** kann jetzt jederzeit durch `name(parameter1, parameter2)` aufgerufen werden. Es sieht fast so aus, als ob sie den Sprachumfang von JavaScript damit ergänzt. Funktionen können auch ohne Namen definiert werden, wie im obigen Beispiel, als das Abschicken des Formulars abgefangen wurde.

## Klassen

Häufig möchte man Daten und Funktionen enger miteinander verknüpfen: Funktionen sollen auf Daten zugreifen können, die ihnen nicht extra übergeben werden müssen, sondern die schon in einem „Objekt“ zusammen mit ihnen gekapselt vorliegen. Das versucht die **objektorientierte Programmierung** zu lösen. Sie unterscheidet dabei traditionell zwischen Klassen – das sind die Baupläne – und den konkreten Objekten – den sog. Instanzen. Das ist in JavaScript zwischenzeitlich (scheinbar) auch so.

Klassen sind also Baupläne, um daraus Objekte zu erzeugen. Neben den Funktionen enthalten sie auch die Daten, mit denen die Funktionen arbeiten. Seit EcmaScript 5 können Klassen direkt in JavaScript verwendet werden, ohne sich mit den eigentlich darunter liegenden Prototypen beschäftigen zu müssen. Beispiel:

```
class Rechteck {  
    Höhe = 0;  
    Breite = 0;  
    constructor(h, b) {  
        this.Höhe = h;  
        this.Breite = b;  
    }  
  
    get Fläche() {  
        return this.Höhe * this.Breite;  
    }  
  
    berechneFläche() {  
        return this.Höhe * this.Breite;  
    }  
}
```

Im Web Inspector (oder direkt im JavaScript-Programm) lässt sich das nun nutzen durch:

```
> let r = new Rechteck(5, 4); r.Fläche  
< 20  
> r.berechneFläche()  
< 20
```

## Funktionaler Programmierung

Dies bezeichnet ein Code-Paradigma, welches Funktionen ohne interne Zustände in den Vordergrund stellt.

## Objektorientierte Programmierung

Damit ist ein Code-Paradigma gemeint, welches Objekte mit ihren internen Zuständen in den Vordergrund stellt. Es eignet sich gut für mittlere bis komplexe Programme, die leicht zu ändern sein sollen und wiederwendbare Programmteile enthalten.

### Klassen und Objekte

Klassen und Objekte sind die wesentlichen Merkmale der objektorientierten Programmierung. Eine Klasse ist der „Bauplan“, aus dem Objekte mit „new“ erzeugt werden können.

Ein **Objekt** der **Klasse** wird durch den „new“-Operator aus einer Klasse erzeugt (instanziert). Dabei müssen die Parameter übergeben werden, die im „constructor“ der Klasse genannt sind. Die mit „get“ definierte Funktion (ein sog. „Getter“) kann direkt am Objekt ohne Klammern aufgerufen werden, bei der ohne „get“ definierten Funktion ist das allerdings nicht möglich.

Ein besonders wichtiges Konzept der objektorientierten Programmierung ist die Vererbung. Dabei kann die aus den Klassen vorhandene Funktionalität übernommen und wieder verwendet werden. Hierbei ist z. B. eine Spezialisierung möglich:

```
class Quadrat extends Rechteck {  
    constructor(l) {  
        super(l, l);  
    }  
}  
  
> let q = new Quadrat(3); q.fläche  
< 9
```

## 5.3 Verwendung von JSON

In realen Web-App-Projekten werden Webanwendungen oft an eine Datenbank oder REST-Services angebunden, um Daten zu laden oder sie dauerhaft zu speichern. Zum Austausch dieser Daten wird das JSON-Datenformat verwendet.

### Aufbau

JSON ist die Abkürzung für „JavaScript Object Notation“ (JSON, n. d.). Jede JSON-Datei beginnt mit geschweiften Klammern „{...}“, danach folgen die Schlüssel-Wert-Paare, welche mit Doppelpunkt „:“ voneinander getrennt werden. Die jeweiligen Einträge werden mit Komma „,“ aneinander angeschlossen, ähnlich wie bei Arrays. Die Schlüssel müssen im Unterschied zur Objektnotation bei JSON in Anführungszeichen stehen und die Werte dürfen ausschließlich aus den folgenden Datentypen bestehen:

Tabelle 5: JSON Datentypen

Datentyp	Beschreibung	Beispiel-Code
Null-Wert	„Leere“ Werte werden mit null initialisiert.	"id": null,
Boolean	kann nur true oder false enthalten	"visible": false,
Number	kann eine Ganz- oder Kommazahl sein	"pi": 3.14159265359,
String	repräsentiert Zeichenketten	"name": "Startup",
Array	Die Array-Elemente können aus den anderen Datentypen bestehen.	"keywords": ["webapp", "fast"],

Datentyp	Beschreibung	Beispiel-Code
Object	Damit sind JSON-Hierarchien möglich. Jedes „Unter-Objekt“ steht wiederum in geschweiften Klammern „{...}“.	"navigation": { "id": 5, "name": "Service", "visible": false },

Quelle: Christian Winkler, 2022.

## Einsatz

Da moderne Web-Applikationen zumindest teilweise auch im Browser laufen, brauchen sie eine Möglichkeit, Daten auf dem Server zu speichern und auszutauschen. Weiterhin fragen immer mehr Maschinen und Programme Daten selbstständig über das Internet ab und kommunizieren miteinander („Internet of Things“). Oft geschieht dies mithilfe von **REST**-basierten **Web Services**, die JSON als Transport-Datenformat verwenden. Viele große Web-Portale, wie Facebook, Twitter, Amazon, Microsoft und Google, bieten REST-Schnittstellen an, um ihre Dienste zu nutzen. Dafür muss man sich per Passwort oder Sicherheits-Token authentifizieren.

Ein öffentliches, frei zugängliches Beispiel für REST Web Services befindet sich auf der Website von Predic8 (predic8, n. d.). Dort kann man z. B. im Browser bestimmte URLs aufrufen und erhält als Antwort Testdaten im JSON-Format zurück.

## Verwendung in JavaScript

JavaScript kann nativ JSON-Daten lesen und schreiben. Dafür gibt es folgende Befehle:

- `JSON.stringify(Objekt)` – übersetzt die Variable `Objekt` nach JSON und gibt das Ergebnis als String zurück.
- `JSON.parse(jsonData)` – übersetzt die String-Variable `jsonData` in ein JavaScript-Objekt und gibt es zurück. Typischerweise kommen diese JSON-Daten von einem Server per REST-URL-Aufruf oder aus einer statischen Datei.
- `fetch(url)` – lädt Server-Daten von der URL, die in der Variable `url` stehen. Das müssen nicht zwingend JSON-Daten sein, sondern können auch Bilder, PDFs, XML oder sonstige Daten sein. Da es nicht klar ist, wie lange ein entsprechender Aufruf dauert, ist dies eine sog. asynchrone Funktion. In neueren JavaScript-Versionen kann man damit fast umgehen wie mit einer normalen Funktion, wenn man das Präfix „`await`“ voranstellt.

Folgendes JSON-File stellt Attribute einer Person dar und ist von Wikipedia (2022) entlehnen:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10010"
  }
}
```

### REST

Representational State Transfer ist eine Architektur, die Web Services über HTTP anbietet. Per GET-Befehl werden Daten abgerufen, per POST-Befehl Daten hochgeladen. REST Ist heutzutage die verbreitetste Form der Client-Server-Kommunikation.

### Web Services

Damit sind Programmierschnittstellen gemeint, die über das Internet abrufbar sind. Sie werden wie Browser-URLs aufgerufen und liefern als Antwort JSON- oder XML-Daten.

```

    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}

```

Das folgende Code-Beispiel zeigt, wie man eine JSON-Datei vom Server lädt, die Daten in ein JavaScript-Objekt umwandelt und anschließend ausgibt:

```

> const response = await fetch("person.json")
> const json = await response.json()
> json
< Object { firstName: "John", lastName: "Smith", isAlive: true, age: 27,
address: {...}, phoneNumbers: (2) [...], children: (3) [...], spouse:
null }

```

Das Objekt steht nun in der Variable `json` zur Verfügung und könnte z. B. mit einer `innerHTML` in die aktuell **dargestellte Webseite** integriert werden. Da sich darin auch einige Arrays befinden, würden sich Schleifen anbieten, um alle Werte einzubinden.

**Einmalige Darstellung?**  
Manche Applikationen haben nur eine einzige Seite, die immer wieder über JavaScript aktualisiert wird. JSON wird dabei oft zum Datenaustausch mit dem Server verwendet.

## 5.4 Gängige JavaScript-Frameworks

JavaScript liefert lediglich einen „Grundwortschatz“ an Befehlen. Um reale, vollständige Web-Applikationen damit zu programmieren, bedarf es vieler hunderter Zeilen Code. Deshalb fingen Entwickler:innen damit an, **Programm-Bibliotheken** („Libraries“) zu schreiben, in denen sie die wichtigsten Funktionen bündelten. Später veröffentlichten sie diese Libraries und erleichterten damit anderen Entwickler:innen die Vorarbeit erheblich.

Im Laufe der Zeit entstand die Nachfrage nach kompletten „Programm-Gerüsten“/**Frameworks**, welche große Code-Teile bereits vorgeben und nur noch angepasst werden müssen. Der Unterschied zwischen Framework und Bibliothek wird im Artikel „Framework vs Bibliothek“ von Patrick Böllhoff (2022) sehr detailliert erklärt und ist im Literaturverzeichnis verlinkt.

Die bekanntesten JavaScript-Frameworks werden nachfolgend kurz vorgestellt. Sie sind im Lernzyklus 5.1 „JavaScript-Geschichte“ chronologisch aufgelistet.

### **Jquery (2005) (js.foundation, n. d.)**

Vor etwa 15 Jahren war die Programmierung in JavaScript äußerst mühsam und auch stark abhängig davon, welche Features der jeweilige Browser gerade unterstützte. Der damals moderne Firefox war viel weiter als der deutlich weiter verbreitete Internet Explorer. Um dennoch mit möglichst einheitlichem JavaScript-Code arbeiten zu können, entstanden mehrere Bibliotheken, von denen sich Jquery am besten durchsetzen konnte. Die Funktionen waren komfortabel, da die API durchdacht war und die Programmierung damit Spaß machte.

Jquery gibt es immer noch und es wird auch nach wie vor von vielen Websites eingesetzt. Allerdings handelt es sich dabei eher um eine Art „Werkzeugkasten“, der die Programmierung an einigen Stellen angenehmer gestaltet. Die meisten Funktionen, die ursprünglich den Charme von Jquery ausmachten, werden heute direkt von den Browsern angeboten. Daher lohnt es sich bei neuen Projekten heute nicht, noch Jquery einzusetzen.

### **Angular.js (2010) (Angular, n. d.-a)**

Angular.js wurde 2010 von Google veröffentlicht, um das Entwickeln von sog. **Single-Page-Applications** zu vereinfachen. In den ersten fünf Jahren war es eines der beliebtesten Web-Frameworks, weil es das Programmieren von wiederverwendbaren Bausteinen (Komponenten) erleichterte. Die meisten Angular-Programme werden mittlerweile in „TypeScript“ (Turner, 2014) geschrieben, einer statisch typisierten Variante von JavaScript. Angular hat das **MVVM-Muster** bekannt gemacht. Heutzutage hat seine Popularität nachgelassen, da neue Frameworks, wie Vue.js etc. von Programmier:innen bevorzugt werden.

Im Anschluss ist ein schematischer Beispiel-Code, der „Hello World“ im Browser ausgibt, dargestellt.

HTML-Code in „index.html“:

```
<html>
<body>
<app-root></app-root>
</body>
</html>
```

JavaScript-Code in der Datei „app.root.ts“:

**Programm-Bibliothek**  
Die Programm-Bibliothek ist eine Sammlung von Klassen und Funktionen für einen bestimmten Einsatzzweck, welche unabhängig voneinander verwendet werden können.

**Framework**  
Ein Framework ist ein fertiges Gerüst für bestimmte Arten von Programmen. Es schreibt genau vor, welcher Code geschrieben werden muss, um es zu benutzen.

**Single-Page-Application/SPA**  
In modernen Web-Apps werden pro Klick keine neuen HTML-Seiten mehr vom Server geladen, stattdessen läuft die gesamte Bedienlogik im Browser. Das heißt, dass beim App-Start der gesamte Code geladen wird, danach nicht mehr. Dadurch fühlt sich die Web-App auf Smartphones wie eine native App an.

### **MVVM-Muster**

Diese Abkürzung steht für „Model-View-View-Model“ und ist ein Entwurfsmuster zum Trennen von Darstellung und Logik der UI-Schnittstelle. Dadurch kann das Design (CSS) und die Benutzungslogik (JavaScript) parallel von zwei verschiedenen Personen entwickelt werden, da sie unabhängig voneinander sind.

```
import { Component } from '@angular/core';
@Component ({
  selector: 'app-root',
  template: '<h1>Hello {{name}}</h1>',
})
export class AppComponent { name = 'World'; }
```

### **React (2013) (React, n. d.-a)**

React wurde 2013 von Facebook veröffentlicht. Ein Großteil der Funktionalitäten auf den Seiten von Facebook-, Instagram- und WhatsApp-Webseiten sind damit programmiert, z. B. die Übersicht mit der Timeline oder der Nachrichten-Historie. Es ermöglichte es als eines der ersten Frameworks, mit Komponenten zu arbeiten, die die Integration von HTML, CSS und JavaScript erlauben. Heutzutage ist React eines der beliebtesten clientseitigen Web-Frameworks. Eine Einführung in React findet sich auf der Webseite (React, n. d.-b).

Im Anschluss ist ein schematischer Beispiel-Code, der „Hello World“ im Browser ausgibt, dargestellt.

HTML- und JavaScript-Code (React, n. d.-b):

```
<div id="root"></div>
<script type="text/babel">
  const root = ReactDOM.createRoot(document.getElementById('root'))
  root.render(<h1>Hello, world!</h1>)
</script>
```

### **Vue.js (2014) (Vue.js, n. d.)**

Vue.js wurde 2014 vom ehemaligen Google-Mitarbeiter Ewan You entwickelt, weil ihm React zu unstrukturiert und Angular zu kompliziert war. Es vereint die flache Lernkurve von React mit dem **Two-Way-Binding** von Angular. Dies ist heutzutage ein sehr beliebtes Web-Framework und viele namhafte Firmen verwenden es: Facebook, Netflix, Adobe, GitHub, Apple, Google uvm.

Anbei ein schematisches Vue.js Code-Beispiel, welches ein Eingabefeld mit der Modell-Variable `title` verknüpft, sodass beide synchron sind:

HTML- und JavaScript-Code:

```
<div id="app">
  <input type="text" v-model="title"/>
  <p> {{ title }} </p>
</div>

<script>
```

```
var app = new Vue({
  el: "#app",
  data: function(){
    return {
      title: "Hello World!",
    }
  }
})
</script>
```

### Svelte (2016) (Svelte, n. d.)

Seit 2016 erfreut sich das Web-Framework „Svelte“ zunehmender Beliebtheit, weil es noch einfacher als Vue.js zu erlernen ist und deutlich weniger Code benötigt als React.

Das folgende Code-Beispiel erlaubt die Eingabe zweier Zahlen, deren Summe ausgegeben wird. Auch wenn das Beispiel an sich sehr einfach erscheint, passiert im Hintergrund sehr viel – das ist bei vielen Frameworks so.

JavaScript- und HTML- Code:

```
<script>
let a = 1;
let b = 2;
</script>

<input type="number" bind:value={a}>
<input type="number" bind:value={b}>
<p>{a} + {b} = {a + b}</p>
```



#### ZUSAMMENFASSUNG

JavaScript ist eine objektorientierte, dynamisch typisierte, C-ähnliche Programmiersprache, die in allen Browsern zur Verfügung steht und dazu verwendet werden kann, Webseiten clientseitig dynamisch zu gestalten. In seiner modernen Form wurde JavaScript als ECMAScript standardisiert. Die Version 7 kann von allen modernen Browsern verarbeitet werden.

Wie viele andere Programmiersprachen hat auch JavaScript if-Anweisungen und bietet unterschiedliche Varianten für Schleifen an. Ein entscheidendes Merkmal von JavaScript sind die Funktionen – teilweise ist es an die funktionale Programmierung angelehnt und ermöglicht damit sehr elegante Konstrukte. Die zentrale Datenstruktur in JavaScript ist das Object, das aus Schlüssel-Wert-Paaren besteht und verschachtelt

werden kann. In neueren Sprachversionen hat JavaScript auch den Umgang mit Klassen gelernt und so die objektorientierte Programmierung im Vergleich zu den vorher notwendigen Prototypen deutlich vereinfacht.

Mehr und mehr hat sich JSON als Datenformat etabliert. Es dient sowohl zum Datenaustausch als auch zur Speicherung von Daten. Der große Vorteil ist dabei die einfache Integrierbarkeit in bestehende Applikationen, weil das Format direkt von JavaScript interpretiert und in Objekte gewandelt werden kann.

Für komplexe clientseitige Web-Applikationen gibt es viele Frameworks, die die Arbeit mit JavaScript strukturieren und Routineaufgaben erleichtern. Als Vorreiter ist diesbezüglich Jquery anzuführen, das aber in modernen Anwendungen nicht mehr verwendet werden sollte. Heute sind React.js und Vue.js die populärsten Frameworks, die von vielen großen Websites eingesetzt werden.

# **LEKTION 6**

## **TESTEN UND SICHERHEIT VON WEBANWENDUNGEN**

### **LERNZIELE**

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- Konzepte zum Testen von Webanwendungen zu verstehen.
- typische Sicherheitsprobleme von Webseiten zu erkennen und zu wissen, wie sie überwunden werden können.

## 6. TESTEN UND SICHERHEIT VON WEBANWENDUNGEN

### Einführung

Web-Applikationen bestehen aus unterschiedlichen Komponenten – HTML-Seiten, Stylesheets in Form von CSS und JavaScript-Dateien, die für ein interaktives Erlebnis auf den Seiten sorgen. Diese Teile haben starke Abhängigkeiten zueinander und müssen daher zusammenpassen, damit keine Fehler auftreten und die Benutzenden ein optimales Ergebnis erhalten.

Häufig arbeitet ein ganzes Team an einer Web-Applikation. Dann sind die Verantwortlichkeiten für die einzelnen Komponenten verteilt – ein Teil kümmert sich um HTML, ein anderer um CSS und wieder andere um JavaScript. Schließlich muss alles integriert werden und dabei kann es leicht zu Problemen und Inkompatibilitäten kommen. Um diese möglichst zeitnah zu erkennen, können (automatisierte) Tests auf verschiedenen Ebenen durchgeführt werden. Besonders wichtig sind diese bei der agilen Software-Entwicklung, in der es sehr viel mehr Releases gibt, die gar nicht mehr manuell überprüft werden können. Automatisiertes Testen wird daher zu einer Kernkompetenz.

Web-Applikationen werden nicht nur zur Bereitstellung von Informationen eingesetzt, sondern auch in Online-Shops, beim Banking oder beim E-Business, dort speziell bei Anwendungen, wie Saslesforce und Workday. Weil in solchen Fällen vertrauliche Informationen verarbeitet werden, spielt die Sicherheit von Web-Applikationen eine immer größere Rolle. Wie wichtig das Thema ist, zeigt sich auch darin, dass es Berichte über Datenlecks oder Attacken regelmäßig in die Tagespresse schaffen.

### 6.1 Testen von Webanwendungen

Software-Bugs können unglaublich teuer werden (Raygun Blog, 2022). Daher ist es wichtig, solche Fehler schon im Ansatz zu vermeiden. In der Software-Entwicklung wurde das schon vor langer Zeit erkannt und es hat sich eine Test-Kultur etabliert, die zunehmend auf automatisierten Prozessen beruht. Wie andere Software-Programme werden auch Web-Applikationen in vielen Fällen geschäftskritisch und durchdringen immer mehr Bereiche des Lebens. Umso wichtiger ist ihre Stabilität. Um diese trotz ständiger Veränderung zu gewährleisten, bedient man sich auch hier automatisierter Tests.

#### Release

Häufig wird Software kontinuierlich weiterentwickelt, aber nur bestimmte Stände werden an Nutzende weitergegeben – diese werden als Release bezeichnet.

Web-Applikationen werden in jedem **Release** verändert, optimiert und erweitert – im Rahmen der agilen Entwicklung oft auch in Sprints. Um teure Bugs zu vermeiden und die Stabilität zu gewährleisten, verwendet man automatisierte Tests.

## **Arten von Tests**

Tests können auf ganz unterschiedlichen Ebenen durchgeführt werden:

a) **Funktionstest**

Darin wird die ordnungsgemäße Funktion der Anwendung gemäß den Anforderungen getestet.

b) **Regressionstest**

Dies ist ein Test, der überprüft, ob die Weiterentwicklung einen neuen Fehler erzeugt hat, nachdem eine neue Funktion hinzugefügt oder ein früherer Fehler behoben wurde.

c) **Usability-Test**

In einer Usability-Testsitzung bitten Moderierende Teilnehmende, Aufgaben mit der App auszuführen. Während der Ausführung beobachtet der:die Moderator:in das Verhalten der teilnehmenden Person und notiert sich das Feedback.

d) **Kompatibilitätstest**

Dieser Test prüft, ob die Web-Applikation auf den verbreitetsten Ziel-Endgeräten läuft

e) **Performance-Test (lokal)**

Darin wird die Leistung der Web-Applikation (Reaktionsfähigkeit, Robustheit, Stabilität) geprüft.

f) **Last-Test (Netzwerk)**

Dieser Test fokussiert auf die Web-Applikations-Performance unter großer Server-Belastung.

g) **Unterbrechungsprüfung**

Dieser Test überprüft, wie die Anwendung auf Unterbrechungen wie eingehende Anrufe, SMS, Batterieschwäche, Netzwerkausfall, Anschließen externer Geräte usw. reagiert

h) **User-Interface-Test**

Darin wird getestet, ob die Web-Applikation eine nahtlose Benutzeroberfläche bereitstellt.

i) **Lokalisierungstest (Übersetzung)**

Es wird prüft, ob die Übersetzungen in andere Sprachen korrekt sind („native speaker“).

j) **Sicherheitstest**

Dieser Tests prüft den Schutz vor Sicherheitsbedrohungen und -verletzungen, dazu gibt es ein eigenes Projekt OWASP (OWASP, n. d.-a), siehe Lernzyklus 6.2.

Besonders wichtig sind die Funktionstests, die im Folgenden genauer betrachtet werden.

## **Unit Tests**

Die kleinste Variante der Funktionstests sind die sog. Unit-Tests. Moderne Web-Applikationen sind modular aufgebaut, d. h. sie benutzen unabhängige, wiederverwendbare Funktionsseinheiten („Units“). Dementsprechend kann man sie unabhängig und parallel testen, da sie keine Abhängigkeiten zu anderen „Units“ haben (sollen). Eventuelle Schnittstellen zu anderen Applikations-Modulen werden simuliert („**gemockt**“). Führt man nun Ände-

### **Mocking**

Oftmals müssen Schnittstellen für Tests simuliert werden, weil sie noch gar nicht zur Verfügung stehen bzw. für Tests ungeeignet sind. Das wird als „Mock“ bezeichnet.

ungen an der Web-Applikation durch, so gewährleisten die Unit Tests die Rückwärtskompatibilität und Stabilität der Anwendung. Dadurch wird sichergestellt, dass sich andere Teile der Applikationen auf ein Modul verlassen können.

JavaScript unterstützt selbst keine Unit Tests. Es gibt aber Frameworks, die das übernehmen können. Eine sehr kleine Unit, die man testen kann, ist die etwas erweiterte Klasse Rechteck:

```
class Rechteck {  
    Höhe = 0;  
    Breite = 0;  
    constructor(h, b) {  
        this.Höhe = h;  
        this.Breite = b;  
    }  
  
    get Höhe () { return this.Höhe; }  
    get Breite () { return this.Breite; }  
  
    get Fläche() {  
        return this.Höhe * this.Breite;  
    }  
  
    get Umfang() {  
        return this.Höhe*2 + this.Breite*2;  
    }  
}
```

Testframeworks unterstützen fast immer die Funktion assert, mit der überprüft wird, ob eine bestimmte Bedingung eingehalten wird. In JavaScript ist diese Funktion (noch) nicht integriert, allerdings über die Konsole des Web-Inspectors verfügbar:

```
let assert = console.assert;  
  
let r = new Rechteck(7, 8);  
assert(r.Breite === 8);  
assert(r.Höhe === 7);  
assert(r.Fläche === 56);  
assert(r.Umfang == 30);
```

Die Tests sind hier so konzipiert, dass alle erfolgreich durchgeführt werden. Was bei einer Code-Änderung jetzt leicht passieren könnte, ist das ein:e neue:r Entwickler:in erkennt, dass bei Rechtecken normalerweise zuerst die Breite und anschließend die Höhe übergeben wird. Wenn man diese Änderung im Code durchführt, nämlich einfach r und h miteinander vertauscht, wird der Test nicht mehr erfüllt. Fläche und Umfang stimmen zwar

noch, nicht aber Höhe und Breite. Eine entsprechende Änderung wäre also **inkompatibel** mit der bisherigen Schnittstelle und würde die Entwickelnden darauf hinweisen, dass dies nicht einfach durchgeführt werden kann.

**Inkompatibilität**  
Unit-Tests dienen häufig dazu, Inkompatibilitäten früh zu entdecken und vor einem Release zu beheben.

## Test-Frameworks

Ein wichtiger Prozessschritt ist das automatisierte Testen. Die beliebtesten JavaScript-Test-Frameworks (BrowserStack, n. d.) dafür sind:

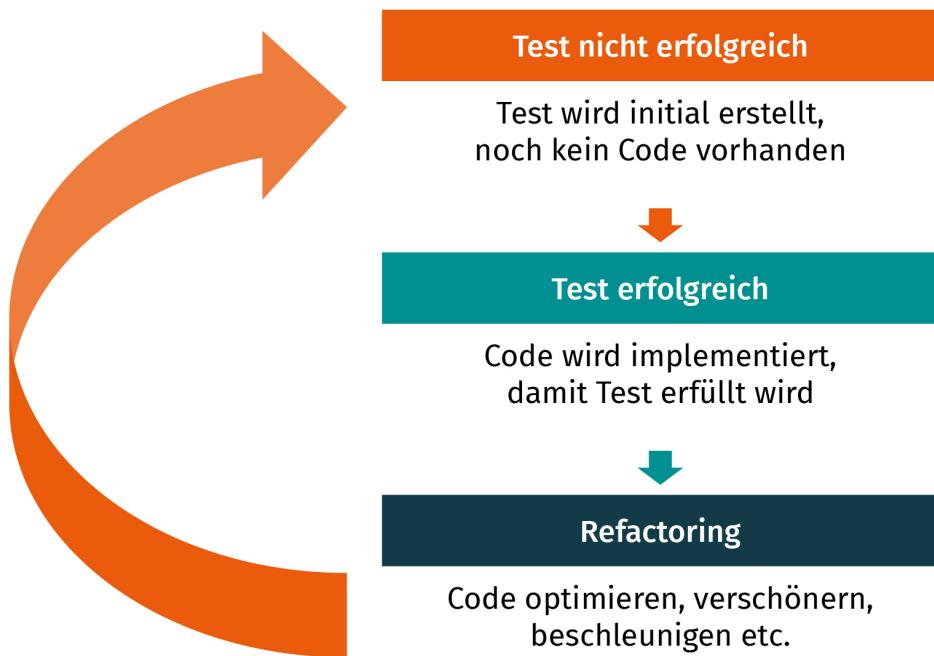
1. Mocha (Mocha, n. d.) – der Marktführer,
2. JEST (Jest, n. d.) von Facebook,
3. Jasmine (Jasmine, n. d.) – speziell für das Frontend, erlaubt „Behavior Driven Development“ sowie
4. Selenium (Selenium, n. d.) – ist eine Fernsteuerung für Browser, mit der man User-Eingaben simulieren kann. Selenium lässt sich sehr leicht als Browser-Add-On installieren. Anschließend können Aktionen damit aufgezeichnet und wieder abgespielt werden. Für das Testen von Frontends ist Selenium damit sicher am einfachsten zugänglich. Die aus der Aufzeichnung entstehenden Programme können auch in eine Testautomatisierung integriert werden.

## Testgetriebene Entwicklung

Bei allen möglichen Tests zeigt es sich, dass Fehler erst am Ende erkannt werden und eine bereits durchgeführte Entwicklung überflüssig war. Das **Paradigma** der testgetriebenen Entwicklung versucht dies zu vermeiden, indem die Tests zuerst vorgegeben werden. Der eigentliche Programmcode muss dann so entwickelt werden, dass er den Tests genügt.

**Paradigma**  
Unterschiedliche Entwicklungsmethoden werden häufig als „Paradigma“ bezeichnet.

Abbildung 28: Testgetriebene Entwicklung



Quelle: Christian Winkler, 2022.

Dieser Prozess hat einige Vorteile:

- Über die Anzahl der fehlgeschlagenen/erfolgreichen Tests kann ein grober Fertigstellungsgrad der Software ermittelt werden.
- Es wird kein Code entwickelt, der den Tests nicht genügt.
- Das Refactoring kann gefahrlos erfolgen, was im Endeffekt zu wesentlich besser lesbarem Code führt.

#### Setup-Phase

Darunter versteht man den Teil eines Projekts, der noch keinen Code produziert.

Allerdings dürfen die Schwierigkeiten auch nicht verschwiegen werden. So dauert insbesondere die **Setup-Phase** eines Projekts viel länger, weil zuerst alle Tests entwickelt werden müssen. Entwickelnde werden außerdem dazu verführt, den Code zuerst zu schreiben, der die meisten Tests besteht. Diese „Belohnung“ entspricht aber nicht automatisch dem, was Nutzende der Applikation erwarten, die lieber mit besonders wichtigen Funktionalitäten starten würden.

## Verhaltensgetriebene Entwicklung

Fehler können nicht nur durch „Bugs“ im eigentlichen Sinne entstehen. Häufig kommt es auch zu Missverständnissen zwischen den fachlichen Anforderungen und dem, wie die Entwickler:innen diese umsetzen oder verstehen. Die testgetriebene Entwicklung hilft dabei nur bedingt, weil die Personen, die die fachlichen Anforderungen stellen, die Tests (Code) im Allgemeinen nicht lesen oder verstehen können. Die verhaltensgetriebene Ent-

wicklung („Behavior-driven development“) schafft hier Abhilfe. In einer leicht verständlichen, aber formalen Sprache werden Anwendungsfälle so definiert, dass diese Personen sie verstehen können. Gleichzeitig ist diese Sprache in Tests realisierbar, sodass gegen die fachlichen Anwendungsfälle getestet werden kann. Eine Beispiel-Implementierung von BDD findet sich z. B. bei Cucumber (n. d.). Ein Anwendungsfall („Szenario“) sieht in der dort verwendeten Sprache „Gherkin“ wie folgt aus:

```
Feature: Is it Friday yet?  
  Everybody wants to know when it's Friday  
  
Scenario: Sunday isn't Friday  
  Given today is Sunday  
  When I ask whether it's Friday yet  
  Then I should be told "Nope"
```

Die Sprache klingt ungewohnt, ist aber verständlich. Daraus erzeugt Cucumber automatisierte Tests, die dann durch die implementierte Software erfüllt werden müssen.

## 6.2 Grundlegende Sicherheitskonzepte und -prinzipien

2020 betrug der wirtschaftliche Schaden durch Cyberangriffe in Deutschland 223 Milliarden Euro (Holthusen, 2021). Immer wieder werden durch Sicherheitslücken Benutzerdaten gestohlen und verkauft. Die 15 größten Datenschutzskandale des 21. Jahrhunderts beschreibt CSO Online in seinem Artikel (CSO Online, 2021). Besonders betroffen sind Web-Applikationen, weil diese häufig öffentlich zugänglich sind und so Begehrlichkeiten wecken. Umso wichtiger sind deshalb Maßnahmen, diese vor solchen Angriffen zu schützen.

### Open Web Application Security Project

Die erste Anlaufstelle für Websicherheit ist das „**Open Web Application Security Project**“ (OWASP, n. d.-a).

Der letzte englischsprachige Bericht stammt aus dem Jahr 2021 und liefert eine Top-10-Liste (OWASP, n. d.-b) der kritischsten Lücken:

#### 1. Defekte Zugangskontrolle

Nicht jede Ressource darf von jedem Benutzenden einer Website gesehen oder gar verändert werden. Das muss individuell überprüft werden, wird aber häufig vergessen. In einem Online-Shop kann das dazu führen, dass es angemeldeten Benutzenden gelingt, die Bestellungen von allen anderen zu sehen.

#### 2. Kryptografische Fehler

#### Open Web Application Security Project

Diese Community überprüft regelmäßig die verbreitetsten Websites und Web-Apps auf Sicherheitslücken und veröffentlicht Reports und Empfehlungen dazu.

**Manipulierte Eingabe**  
Nicht erwünschte Eingaben sind einer der häufigsten Angriffsvektoren auf Web-Applikationen. Eingaben von Benutzern darf man daher niemals vertrauen.

**Sicherheit von Komponenten**  
Eine fehlerhafte Komponente kann die Sicherheit einer ganzen Applikation gefährden.

Bis vor kurzem wurde dieser Fehler noch als „Informations-Exposition“ gelistet. Damit ist gemeint, dass eine Website zu viele Informationen über sich selbst preisgibt. Sicherer ist es, wenn sie möglichst „schweigsam“ bleibt. Je weniger potenzielle Angreifende wissen, desto schwieriger ist es, den Angriff zu planen. Zu dieser Fehlerklasse gehört auch eine unzureichende Verschlüsselung der (transportierten) Daten. Das lässt sich leicht über HTTPS als Protokoll erreichen.

### 3. **Code-Injektion**

Websites sollten nur genau den (JavaScript-)Code ausführen, der von den Entwicklern vorgesehen ist. Durch **manipulierte** Benutzereingaben kann es aber passieren, dass versehentlich auch Code ausgeführt werden kann, den ein:e Angreifer:in z. B. in ein Suchfeld einträgt. Diese sog. Cross-Site-Scripting-Lücken (XSS) sind sehr gefährlich, schwer zu vermeiden und noch immer auf vielen Websites vorhanden.

### 4. **Unsicheres Design**

Bereits bei der Architektur einer komplexen Web-Applikation müssen Sicherheitsfragen gestellt werden. So können z. B. Komponenten voneinander getrennt werden, die interne und externe Daten verarbeiten. Unterbleibt dies, ist es im Laufe der Entwicklung schwer zu beheben und stellt einen permanenten Unsicherheitsfaktor dar.

### 5. **Sicherheitsfehlkonfiguration**

Viele Komponenten einer Web-Applikation (Web-Server, DNS-Server, CDN usw.) ermöglichen individuelle Konfigurationen. Viele davon sind auch sicherheitsrelevant. Dadurch gibt es sehr viele Stellen, an denen etwas schiefgehen kann. Sicherheitshalber sollten solche Konfigurationen daher nur von erfahrenen Administratoren vorgenommen werden. Möchte man nur eine kleine Website hosten, ist die vom Hoster vorgegebene Konfiguration meistens schon hinreichend sicher – auch wenn ein zusätzlicher Blick natürlich nie schaden kann.

### 6. **Anfällige und veraltete Komponenten**

Gerade bei JavaScript-Frameworks gibt es eine große Kaskade von Abhängigkeiten und auch immer wieder neue Versionen der Frameworks selbst. Häufig erfolgen diese Releases, weil sicherheitsrelevante Fehler in den **Teilkomponenten** festgestellt und korrigiert wurden. Hat man selbst einen Build-Prozess implementiert, müssen auch hier regelmäßig neue Versionen der Bibliotheken installiert werden, um nicht Angreifen ausgesetzt zu werden, die spezifische Fehler darin ausnutzen.

### 7. **Identifikations- und Authentifizierungsfehler**

Viele Websites sind passwortgeschützt oder erfordern eine andere Form der Authentifizierung bzw. Autorisierung. Hierbei können viele Fehler passieren. Keinesfalls darf z. B. ein Passwort abgespeichert werden, weil es dann bei einem evtl. kompromittierten Server an die Öffentlichkeit gelangen könnte.

### 8. **Software- und Datenintegritätsfehler**

Wenn Software installiert wird, muss sichergestellt werden, dass diese nicht manipuliert wurde. Dafür können z. B. Hash-Funktionen oder Signaturen eingesetzt werden. Gleicher gilt auch für Daten, die eventuell aus dritten Quellen verwendet werden. Auch hier sollte die Integrität überprüft werden.

### 9. **Sicherheitsprotokollierungs- und Überwachungsfehler**

Ein wesentlicher Schutz vor Angriffen ist die Protokollierung. Wenn man erkennt, dass eine bestimmte Komponente wiederholt angegriffen wird, kann man eventuell noch rechtzeitig reagieren. Allerdings ist es mit der Protokollierung allein nicht getan, die Daten müssen auch überprüft werden. Dazu kann leistungsfähige Monitoring-Software eingesetzt werden, um Sicherheitsvorfälle möglichst zeitnah zu erkennen.

## 10. Serverseitige Anfragefälschung

Einige Funktionen können auch moderne Web-Applikationen nicht eigenständig durchführen, dazu gehört z. B. die **Zahlungsabwicklung** in einem Online-Shop. Für solche und ähnliche Funktionen verwenden die Web-Applikationen Schnittstellen zu anderen Diensten, die häufig über URLs aufgerufen werden. Gelingt es Angreifenden, diesen URL-Aufruf umzuleiten, so hat er:sie Zugriff auf sehr viele vertrauliche Informationen.

**Zahlungsabwicklung**  
Besonders Kreditkartennummern sind bei Dieben extrem beliebt und daher Ziel von Angriffen.

### Beispiel für XSS

Eine der am weitesten verbreiteten Attacken ist das Cross-Site-Scripting, daher soll diese hier genauer betrachtet werden. Basis für dieses spezifische Beispiel ist die Webseite mit der (nicht) implementierten Suche, die hier jetzt noch etwas nutzungsfreundlicher werden soll. Dazu ändert sich das JavaScript wie folgt:

```
let user = { username: "joe", password: "dei9thuSi9eereel" };

document.querySelector("form").onsubmit = function() {
    //Wert des Suche-Felds bestimmen
    let suche = document.querySelector("input").value;
    if (suche.length < 3) {
        alert("zu kurzer Suchbegriff");
    } else {
        let h1 = document.querySelector("h1");
        h1.innerHTML = "Suche nach \"" + suche + "\" noch nicht
implementiert";
        h1.style["background-color"] = "green";
    }
    return false;
}
```

Viel hat sich nicht getan – es gibt ein Objekt mit dem gerade angemeldeten User. Außerdem meldet die Website, dass die Suche noch nicht durchgeführt werden kann und gibt dazu noch mal den Suchbegriff aus. Eine Suche nach „hallo“ bestätigt, dass das gut funktioniert. Ein bisschen mehr passiert, wenn man nach „<b>hallo</b>“ sucht. Plötzlich ist „hallo“ in der Fehlermeldung fett geschrieben. Dabei ist etwas geschehen, was man vermeiden sollte: Dem Nutzenden ist es gelungen, eigenen Code in die HTML einfließen zu lassen (zu „**injizieren**“). Das ist an sich noch nicht schlimm, allerdings können Angreifende über den gleichen Mechanismus auch JavaScript in die Seite einfließen lassen. <script>-Tags werden in innerHTML zwar nicht ausgeführt, aber mit folgendem Suchbegriff erscheint eine Dialogbox: . Somit wurde aktiver Code, der von den Benutzenden eingegeben wurde, im Kontext der

**Injektions-Attacke**  
Manipulierte Anfragen so zu formulieren, dass Code im Kontext der Webseite ausgeführt wird, ist eine besonders beliebte Methode, um an sensible Daten zu gelangen.

Webseite ausgeführt. Das ist eine schwere Sicherheitslücke (und leider noch immer sehr weit verbreitet). Verändert man die Suche leicht, kann man auch auf das Passwort der Benutzenden zugreifen:  Professionelle Angreifende nutzen das, um ganze Skripte von dritten Websites z. B. über präparierte URLs einzuschleusen. Damit lassen sich Passwörter eingeloggter Nutzer:innen auslesen oder auch Bestellungen in deren Namen tätigen.

Wie können solche Fehler verhindert werden? Das Kernproblem ist die naive Verarbeitung der eingegebenen Informationen. Alles, was von Nutzenden eingegeben wird, ist prinzipiell unsicher („tainted“) und darf niemals direkt wieder ausgegeben werden. Stattdessen müssen die geeigneten Quoting-Funktionen verwendet werden, die die Steuerzeichen (wie < oder >) ersetzen und damit die Eingabe von Befehlen unmöglich machen. In dem obigen Beispiel lässt sich das sehr einfach lösen, indem man `innerHTML` durch `innerText` ersetzt – dann wird kein HTML mehr interpretiert und solche „aktiven Inhalte“ lassen sich nicht mehr einschmuggeln.

## Best Practices

- **Korrekte Konfiguration des Webservers**

Webserver bieten heute sehr viele Konfigurationsoptionen, beginnend bei der Authentisierung bis zu möglichen Weiterleitungen usw. Nachdem der erste Kontakt der Nutzenden immer mit dem Webserver stattfindet, ist das auch der erste Punkt, an dem sich eine eventuelle Sicherheitslücke ergibt.

- **Nutzung einer Web Application Firewall**

Viele Hoster haben solche speziellen Firewalls in ihrem Programm. Diese können viele Attacken abwehren, sind aber eher auf serverseitige Anwendungen mit Datenbanken etc. spezialisiert – dort funktionieren sie allerdings sehr effizient und können Standard-Attacken abwehren.

- **HTTPS statt HTTP**

Eigentlich sollte das unverschlüsselte HTTP-Protokoll gar nicht mehr zum Einsatz kommen, weil dadurch auch alle Passwörter etc. im Klartext übertragen werden. HTTPS-Zertifikate gibt es von vielen Anbietern kostenlos und daher spricht wenig dafür, auf HTTPS zu verzichten.

- **DNS absichern**

Das Domain Name System (DNS) ist eine Art Adressbuch für das Internet und gibt an, wo welcher Webserver zu finden ist. Angreifende können sich auch des DNS bemächtigen und eine Website einfach umleiten. Über verschiedene Security-Mechanismen (wie DNSSEC [Heise Newsticker, n. d.]) kann man das verhindern.



### ZUSAMMENFASSUNG

Moderne Software wird immer komplexer, gleichzeitig aber auch geschäftskritischer. Dieser Trend ist auch bei Web-Applikationen zu beobachten. Weil sich Fehler in der Software, z. B. in einem Online-Shop,

direkt durch Umsatzverluste bemerkbar machen, ist das Testen der Software absolut entscheidend. Damit können Fehler schon im Vorfeld erkannt und behoben werden.

Software kann auf unterschiedlichen Ebenen getestet werden. Sehr populär sind Unit-Tests, die die einzelnen Funktionseinheiten unabhängig voneinander testen. So können Teams parallel an der Softwareentwicklung arbeiten und dabei sicherstellen, dass ihre Änderungen nicht die Stabilität des Gesamtsystems beeinflussen. Um Tests strukturiert durchzuführen, gibt es unterschiedliche Frameworks, die auch in einen automatischen Test integriert werden können.

Eine neue Entwicklung ist die testgetriebene Entwicklung. Dabei werden zuerst Tests implementiert und danach die eigentliche Funktionalität. Da es auch schon vorher zu Missverständnissen zwischen Anforderungen und Implementierung kommen kann, kann dies durch verhaltensgetriebene Entwicklung ergänzt werden, in der die fachlichen Anforderungen so formalisiert werden, dass sie von den Anfordernden verstanden, aber auch automatisiert in Tests gewandelt werden können.

Häufig werden sensible Daten in Webanwendungen verarbeitet. Hier muss besonderes Augenmerk auf die Sicherheit gelegt werden. OWASP stellt eine Liste von häufigen Sicherheitsproblemen zusammen, die besonders oft für Attacken verwendet werden. Eine besonders gern ausgenutzte Sicherheitslücke sind die Cross-Site-Scripting-Attacken, mit denen Angreifende interne Informationen auslesen und für sich verwenden können. Sorgfalt bei der Programmierung, ausführliche Tests und sichere Protokolle können dabei helfen, die meisten Sicherheitsprobleme zu verhindern.



# **ANHANG**

# LITERATURVERZEICHNIS

adaptive path. (n. d.). *Ajax: A new approach to web applications.* <https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php>

AdminLTE. (2022). *ColorlibHQ/AdminLTE.* <https://github.com/ColorlibHQ/AdminLTE>

Allotropia. (n. d.). *Soffice.* <https://lab.allotropia.de/wasm/>

Angular. (n. d.-a). *Angular.* <https://angular.io/>

Angular. (n. d.-b). *angular/angular.js.* <https://github.com/angular/angular.js/releases/tag/v0.9.0>

Apache Software Foundation. (n. d.). *Welcome! - The Apache HTTP Server Project.* <https://httpd.apache.org/>

Apache Subversion. (n. d.). *Apache Subversion.* <https://subversion.apache.org/>

BabelJS. (n. d.). *Babel. The compiler for next generation JavaScript.* <https://babeljs.io/>

Berners-Lee, T., Fielding, R. T., & Masinter, L. M. (2005). *Uniform Resource Identifier (URI): Generic Syntax.* <https://doi.org/10.17487/RFC3986>

Böllhoff, P. (23. Februar 2022). *Framework vs Bibliothek: Was ist der Unterschied?* [https://kru schecompany.com/de/framework-vs-bibliothek/](https://kruschecompany.com/de/framework-vs-bibliothek/)

Bootstrap. (n. d.-a). *Bootstrap.* <https://getbootstrap.com/>

Bootstrap. (n. d.-b). *Bootstrap Examples.* <https://getbootstrap.com/docs/5.2/examples/>

BrowserStack. (n. d.). *Top Javascript Testing Frameworks.* <https://www.browserstack.com/guide/top-javascript-testing-frameworks>

Bulma. (n. d.). *Bulma: Free, open source, and modern CSS framework based on Flexbox.* <https://bulma.io>

Cerf, V., & Kahn, R. (1974). A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5), 637–648. <https://doi.org/10.1109/TCOM.1974.1092259>

Computer History. (n. d.). *The Apple II – CHM Revolution.* <https://www.computerhistory.org/revolution/personal-computers/17/300>

computerhistory.org. (n. d.). *Commodore 64 – CHM Revolution*. <https://www.computerhistory.org/revolution/personal-computers/17/298/1179>

CSO Online. (16. Juli 2021). *The 15 biggest data breaches of the 21st century*. <https://www.csosonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>

css Zen Garden. (n. d.). *css Zen Garden: The Beauty of CSS Design*. <http://dv.csszengarden.com/>

Cucumber. (n. d.). *Cucumber Open—Get Started with BDD Today*. <https://cucumber.io/tools/cucumber-open/>

CVS. (n. d.). *CVS - Open Source Version Control*. <http://cvs.nongnu.org/>

ECMAScript. (2015). *Language Specification*.

ECMAScript. (2016). *Language Specification*.

ECMAScript. (2011). *Language Specification—ECMA-262 Edition 5.1*. <https://262.ecma-international.org/5.1/>

egghead.io. (n. d.). *Evan You, creator of Vue.js*. <https://egghead.io/podcasts/evan-you-creator-of-vue-js>

Foundation. (n. d.). *The most advanced responsive front-end framework in the world*. <https://get.foundation/>

git SCM. (n. d.). *git*. <https://git-scm.com/>

Github. (n. d.-a). *Features. GitHub Actions*. <https://github.com/features/actions>

Github. (n. d.-b). *GitHub Pages*. <https://pages.github.com/>

Github. (n. d.-c). *GitHub: Where the world builds software*. <https://github.com/>

GNU. (n. d.). *RCS - GNU Project — Free Software Foundation*. <https://www.gnu.org/software/rcc/>

gulp.js. (n. d.). *Gulp.js*. <https://gulpjs.com/>

Heise Newsticker. (n. d.). *DNSSEC in der DNS-Rootzone gestartet*. <https://www.heise.de/newsticker/meldung/DNSSEC-in-der-DNS-Rootzone-gestartet-1039401.html>

Holthusen, L. (13. August 2021). *223 Milliarden Euro Schaden durch Cyberangriffe*. <https://www.datenschutz-notizen.de/223-milliarden-euro-schaden-durch-cyberangriffe-3030768/>

- Hostadvice. (2022). *Global Web Server Market Share August 2022*. <https://hostadvice.com/marketshare/server/>
- IEEE. (18. November 1997). IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. *IEEE Std 802.11-1997*. 1–445. <https://doi.org/10.1109/IEEESTD.1997.85951>
- IETF. (1983). *Telnet Protocol Specification*. <https://doi.org/10.17487/RFC0854>
- IETF. (1985). *File Transfer Protocol*. <https://doi.org/10.17487/RFC0959>
- ISO. (1984). *ISO 7498:1984*. <https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/01/42/14252.html>
- ITU. (1983). *ITU-T Recommendation database*. <https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=10753&lang=en>
- Jasmine. (n. d.). *Jasmine Documentation*. <https://jasmine.github.io/>
- Jenkins. (n. d.). *Jenkins*. <https://www.jenkins.io/>
- Jest. (n. d.). *Jest*. <https://jestjs.io/>
- JetBrains. (n. d.). *WebStorm: The Smartest JavaScript IDE*, by JetBrains. <https://www.jetbrains.com/webstorm/>
- JSConf. (5. August 2013). [JSConfUS 2013] Tom Occhino and Jordan Walke: JS Apps at Facebook. <https://www.youtube.com/watch?v=GW0rj4sNH2w>
- js.foundation, J. F. (n. d.). *JQuery*. <https://jquery.com/>
- JSON. (n. d.). *JSON*. <https://www.json.org/json-en.html>
- MARC. (n. d.). (fwd) *Greetings from the Safari team at Apple Computer*. <https://marc.info/?m=104197092318639>
- Material Design. (n. d.). *Material Design*. <https://material.io/design>
- Materialize CSS. (n. d.). *Documentation – Materialize*. <https://materializecss.com/>
- MDN. (n. d.-a). *CSS: Cascading Style Sheets*. <https://developer.mozilla.org/en-US/docs/Web/CSS>
- MDN. (n. d.-b). *CSS reference*. <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>
- MDN. (n. d.-c). *Flexbox*. [https://developer.mozilla.org/de/docs/Learn/CSS/CSS\\_layout/Flexbox](https://developer.mozilla.org/de/docs/Learn/CSS/CSS_layout/Flexbox)

- MDN. (n. d.-d). *JavaScript*. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- MDN. (n. d.-e). *Learn to style HTML using CSS*. <https://developer.mozilla.org/en-US/docs/Learn/CSS>
- MDN. (n. d.-f). *position*. <https://developer.mozilla.org/en-US/docs/Web/CSS/position>
- MDN. (n. d.-g). *Using media queries*. [https://developer.mozilla.org/en-US/docs/Web/CSS/Media\\_Queries/Using\\_media\\_queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries)
- Microsoft. (n. d.). *IXMLHTTPRequest Members*. [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms760305\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms760305(v=vs.85))
- Mocha. (n. d.). *Mocha – The fun, simple, flexible JavaScript test framework*. <https://mochajs.org/>
- MySQL. (n. d.). *MySQL*. <https://www.mysql.com/>
- nginx. (n. d.). *Nginx*. <https://nginx.org/en/>
- Node JS. (n. d.). *Tags*. *nodejs/node-v0.x-archive*. <https://github.com/nodejs/node-v0.x-archive>
- npm. (n. d.). *Npm*. <https://www.npmjs.com/>
- Oracle. (n. d.). *Java*. <https://www.java.com/de/>
- OWASP. (n. d.-a). *OWASP Foundation, the Open Source Foundation for Application Security*. <https://owasp.org/>
- OWASP. (n. d.-b). *OWASP Top 10 - 2021*. <https://owasp.org/Top10/>
- Parcel. (n. d.). *Parcel – The zero configuration build tool for the web*. <https://parceljs.org>
- PHP. (n. d.). *PHP: Hypertext Preprocessor*. <https://www.php.net/>
- Pico CSS. (n. d.). *Pico.css. Minimal CSS Framework for semantic HTML*. <https://picocss.com>
- predic8. (n. d.). *Online REST Web Service Demo*. <http://www.predic8.com/rest-demo.htm>
- Procházka, T. (2012). *Responsive web design*. [CC0]. [https://en.wikipedia.org/wiki/Responsive\\_web\\_design#/media/File:Complete.png](https://en.wikipedia.org/wiki/Responsive_web_design#/media/File:Complete.png)
- Python. (n. d.). *Welcome to Python.org*. <https://www.python.org/>
- Raygun Blog. (26. Januar 2022). *11 of the most costly software errors in history*. <https://raygun.com/blog/costly-software-errors-history/>

- React. (n. d.-a). *React – A JavaScript library for building user interfaces*. <https://reactjs.org/>
- React. (n. d.-b). *Tutorial: Intro to React – React*. <https://reactjs.org/tutorial/tutorial.html>
- Resig, J. (n. d.). *BarCampNYC Wrap-up*. <https://johnresig.com/blog/barcampnyc-wrap-up/>
- Ritchie, D. M. (1993). The development of the C language. *ACM SIGPLAN Notices*, 28(3), 201–208. <https://doi.org/10.1145/155360.155580>
- Roberts, G. (14. Dezember 2006). *The History of Ethernet*. <https://www.youtube.com/watch?v=g5MezxMcRmk>
- Selenium. (n. d.). *Selenium*. <https://www.selenium.dev/>
- Seobility. (n. d.). *What are Media Queries and how do they work? - Seobility Wiki*. [https://www.seobility.net/en/wiki/Media\\_Questions](https://www.seobility.net/en/wiki/Media_Questions)
- Servo. (n. d.). *Servo*. <https://servo.org/>
- Siegmund, G. (1992). *Grundlagen der Vermittlungstechnik*. R. v. Decker.
- SPIRES. (n. d.). *About SPIRES*. <https://www.slac.stanford.edu/spires/about/>
- Submit Express. (n. d.). *Raging search: A new search engine by AltaVista*. [https://www.submitexpress.com/newsletters/may\\_15\\_00.html](https://www.submitexpress.com/newsletters/may_15_00.html)
- Svelte. (n. d.). *Svelte. Cybernetically enhanced web apps*. <https://svelte.dev/>
- Tailwind CSS. (n. d.). *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. <https://tailwindcss.com/>
- Turner, J. (2. April 2014). *Announcing TypeScript 1.0*. <https://devblogs.microsoft.com/typescript/announcing-typescript-1-0/>
- Unicode Home. (n. d.). *Unicode*. <https://home.unicode.org/>
- Visual Studio. (n. d.). *Live Server – Visual Studio Marketplace*. <https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer>
- Visual Studio Code. (n. d.-a). *Visual Studio Code for the Web*. <https://vscode.dev/>
- Visual Studio Code. (n. d.-b). *Code Editing. Redefined*. <https://code.visualstudio.com/>
- VSCodium. (2022). *VSCodium*. <https://github.com/VSCodium/vscodium>
- Vue.js. (n. d.). *The Progressive JavaScript Framework*. <https://vuejs.org/>

W3C. (n. d.-a). *Hypertext Transfer Protocol—HTTP/1.0*. <https://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA>

W3C. (n. d.-b). *Media Queries Level 4*. <https://www.w3.org/TR/mediaqueries-4/>

Web Archive. (n. d.-a). *ETSI - Long Term Evolution*. <https://web.archive.org/web/20150303230348/http://www.etsi.org/technologies-clusters/technologies/mobile/long-term-evolution>

Web Archive. (n. d.-b). *HTML & CSS*. <https://web.archive.org/web/20101129081921/https://www.w3.org/standards/webdesign/htmlcss#whatcss>

Web Archive. (1995). *Press Release JavaScript*. <https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html>

Web Assembly. (n. d.). *Release wg-1.0. WebAssembly/spec*. <https://github.com/WebAssembly/spec/releases/tag/wg-1.0>

webpack. (n. d.). *Webpack*. <https://webpack.js.org/>

Wikipedia. (2022). *JSON*. <https://en.wikipedia.org/w/index.php?title=JSON&oldid=1106610160>

WordPress. (n. d.). *Blog Tool, Publishing Platform, and CMS*. <https://wordpress.org/>

Yarn. (n. d.). *Yarn. Safe, stable, reproducible projects*. <https://yarnpkg.com/>

# ABBILDUNGSVERZEICHNIS

Abbildung 1: TCP/IP-Referenzmodell .....	16
Abbildung 2: Struktur einer URL .....	17
Abbildung 3: Interaktion zwischen Browser und Server .....	18
Abbildung 4: Model-View-Controller Muster .....	20
Abbildung 5: Anlegen einer neuen Datei bei Visual Studio Code .....	35
Abbildung 6: Darstellung der ersten HTML-Datei im Browser .....	36
Abbildung 7: Grafik zur Seitenstruktur des Startups .....	38
Abbildung 8: HTML-Seite mit hervorgehobener Inhaltsstruktur .....	39
Abbildung 9: Darstellung der Seite ohne Layout .....	42
Abbildung 10: Einchecken in git .....	43
Abbildung 11: HTML wird mithilfe von CSS gestylt .....	44
Abbildung 12: CSS Box Model .....	47
Abbildung 13: Ergebnis des CSS-Experiments .....	48
Abbildung 14: Element-Selektor .....	49
Abbildung 15: Web Inspector mit CSS-Regeln für <h1> .....	50
Abbildung 16: Web Inspector mit berechnetem Layout für <h1> .....	51
Abbildung 17: Anordnung von Kacheln auf unterschiedlichen Endgeräten .....	55
Abbildung 18: Geräteauswahl im Browser .....	56
Abbildung 19: IU-Homepage in der iPhone-Emulation durch den Web Inspector .....	57
Abbildung 20: Lineares Seitenlayout der Homepage .....	59
Abbildung 21: Float-basiertes Layout .....	61

Abbildung 22: Flex-basiertes Layout .....	63
Abbildung 23: Flex-Achsen .....	64
Abbildung 24: Beispielhafte Media Queries .....	65
Abbildung 25: Simulierte Darstellung auf dem Handy .....	66
Abbildung 26: Bootstrap-basiertes Layout der Homepage in Desktop-Auflösung .....	69
Abbildung 27: Boostrap-basiertes Layout in reduzierter Auflösung .....	70
Tabelle 1: Geschichte von JavaScript .....	77
Tabelle 2: Datentypen in Javascript (Auszug) .....	81
Tabelle 3: Vergleichs-Operatoren .....	84
Tabelle 4: Logische Verknüpfungen .....	85
Tabelle 5: JSON Datentypen .....	88
Abbildung 28: Testgetriebene Entwicklung .....	100





 **IU Internationale Hochschule GmbH**  
**IU International University of Applied Sciences**  
Juri-Gagarin-Ring 152  
D-99084 Erfurt

 **Postanschrift**  
Albert-Proeller-Straße 15-19  
D-86675 Buchdorf

 [media@iu.org](mailto:media@iu.org)  
[www.iu.org](http://www.iu.org)

 **Hilfe & Kontakt (FAQ)**  
Antworten rund um Dein Studium findest  
Du jederzeit auf myCampus.