(1) HTML

Eine Webseite besteht in erster Linie aus HTML. Damit wird die Struktur der Webseite aufgebaut.

Die Struktur einer Webseite ist das, was man im Browser sieht. Sie umfasst solche Dinge wie Menüleiste, Navigation, Footer, Bilder, Tabellen, Eingabefelder, Schaltflächen, Listen, Links etc.

(2) CSS

Damit eine Webseite auch schön aussieht, wird CSS benutzt, um die HTML-Elemente, aus denen die Struktur der Seite aufgebaut ist, zu färben, mit Animationen auszustatten und schön zu formatieren.

Umrahmung

Farben

Abstände

Textformatierung

Schriftart

Body-Struktur

Kreise

(3) JavaScript

Eine Webseite ist normalerweise statisch. Das bedeutet, einmal erstellt ist sie nicht änderbar.

Doch mit JavaScript kann man eine Webseite im Browser dynamisch machen. Das bedeutet, die Webseite kann gezielt gesteuert werden. Sie kann Menüs aufklappen, Formulare abschicken, zwischen sichtbaren und unsichtbaren Elementen wechseln, Elementinhalte ändern und vieles mehr.

Mit JavaScript kann man einer Webseite LEBEN einhauchen.

(4) React

Eine Webseite besteht normalerweise aus mehreren einzelnen Seiten. Wenn man jedoch eine professionelle Webseite oder eine „Web-Anwendung“ bzw. eine App erstellen möchte, so ist es Ressourcen-Verschwendung, den gesamten Browser-Inhalt bei jedem „Content-Wechsel“ neuzuladen. Das verbraucht viel Traffic, Speicher und Zeit, basierend auf die Größe des Codes und der Ressourcen, die geladen werden müssen.

Um das zu umgehen gibt es zwei Möglichkeiten:

1. Teile der Webseite vom Server bei Bedarf nachladen (Stichworte PHP, Node, Ruby, Python +- DB)
2. Die Webseite aus nur einer Seite zu gestalten und die Änderungen nur mittels JavaScript durchzuführen, ohne die Seite insgesamt neu laden zu müssen

React ist eine JavaScript-Bibliothek, der den zweiten Ansatz verfolgt. Sie erzeugt dynamische Webseiten die nur aus einer Seite bestehen. Kurz um Single-Page-Apps. Um die Arbeit insgesamt sehr einfach zu gestalten verschmelzt React in gewisser weise JavaScript mit HTML, in dem es JSX unterstützt. Dadurch wird es leichter, Änderungen an der Seite zu machen, ohne viel JavaScript schreiben zu müssen. Ein weiteres Ziel von React ist es, die komplexen Änderungen an der einen Webseite mit Hilfe von Komponenten in eine sehr einfache und vor allem übersichtliche Konzept zu packen, sodass jederzeit eine schnelle Einarbeitung und rascher Umgang mit dem Code gewährleistet ist.

(5) Node

Normalerweise ist JavaScript nur für den Browser vorgesehen. Doch Node bringt JavaScript auch auf den Server.

Das hat den Vorteil, das man für die Server-Programmierung keine weitere Sprache lernen muss und einfach sowohl für den Client (Browser) als auch für den Server in derselben Sprache (JavaScript) programmiert.

Ein weiterer Vorteil ist, dass JavaScript auf beiden Seiten besser aufeinander abgestimmt werden kann. Es ist möglich JavaScript-Codes zu schreiben, die auf beiden Seiten (Client und Server) gleichermaßen verwendet werden können.

DAS Zusammenspiel zwischen CLIENT und SERVER

Die Webseite selbst wird im Browser erzeugt und erscheint auf dem Bildschirm des Surfers. Aber eine Webseite besteht aus Elementen, die vom Server kommen. Dazu gehören der CODE (HTML, CSS, JavaScript) und die Ressourcen (Bilder, Videos, Audiodateien, etc.)

Eine richtige dynamische Webseite kommt nicht drum herum weg, bestimmte Dinge bei Bedarf aus dem Server nachzuladen. Dabei ist es insgesamt egal, ob es eine Single-Page-App ist oder eine gewöhnliche Webseite wie zum Beispiel Amazon oder eBay.

Dieses „Nachladen“ betrifft zum Beispiel die Suchergebnisse bei Google, das Auflisten der Artikel bei eBay/Amazon, das Wechseln zu der nächsten Artikelseite bei Amazon/eBay, wenn bei Youtube eine neue Benachrichtigung aufklappt etc.

Das wird erreicht, in dem der Client (JavaScript) eine Anfrage an den Server schickt (zB Node) und dann eine Antwort erhält und es entsprechend in die Webseite einbaut.

(6) Datenbank

Eine Datenbank ist notwendig, um große Menge an Daten zu speichern und bereitzustellen.

Normalerweise könnte man auch Textdateien nutzen, aber eine Datenbank kann mehr. Sie kann Daten durchsuchen, filtern, jederzeit anpassen, nicht mehr benötigte Daten entfernen, neue Daten hinzufügen.

Eine Datenbank liegt in der Regel auf der Serverseite und liefert Daten beim Bedarf. Mittlerweile sind Datenbanken auch in abgespeckter Form direkt auf dem Client (Browser) verfügbar.

SQL

SQL (Structured Query Language) ist ausschließlich dafür vorgesehen, riesige Mengen Daten zu verwalten. Daher sind SQL-Datenbanken primär auf Servern.

Nutzen kann man sie zum Beispiel mit NodeJS, PHP, Ruby, Python, JSP, ASPX etc.

Auf dem Client (Browser) kann man dagegen eine mini Variante von SQL, die SQLite nutzen oder eine No-SQL Datenbank wie zum Beispiel MongoDB.

Datenbanken dienen zum Verwalten von Kundendaten, Artikeldaten, Standortdaten, personalisierte Einstellungen, Logindaten und so weiter.

SQL vs. No-SQL

SQL Datenbanken sind relationale Datenbanken. Sie können über eine Anfrage (die sogenannten Queries) gesteuert werden. Eine Anfrage wird in der Regel als Text formuliert und an den Datenbank-Engine übergeben. Er interpretiert die Anfrage, erzeugt die angeforderte Ausgabe und schickt diesen wieder zurück.

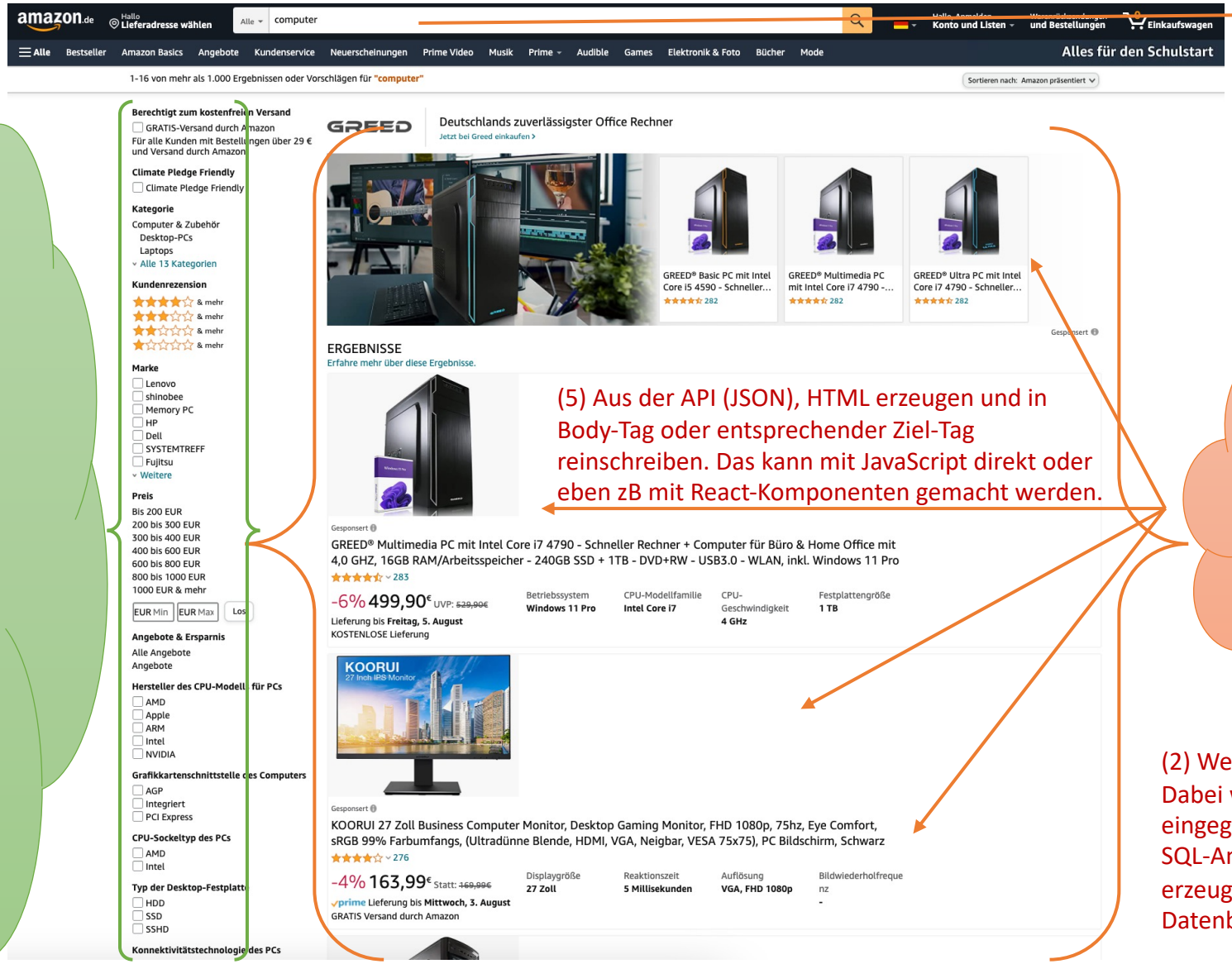
Eine No-SQL Datenbank dagegen funktioniert anders. An dieser Stelle kommt keine Anfrage als Text, sondern für jede Kommunikationsart gibt es eine entsprechende Funktion des Engines, der einfach aufgerufen wird.

Daher eine SQL-Datenbank ist sehr viel mächtiger, als eine No-SQL-Datenbank. Dafür ist ein No-SQL-Datenbank oft schneller als eine SQL-Datenbank.

Eine No-SQL Datenbank ist nicht für große Datenmengen vorgesehen.

Wenn eine Filteroption markiert wird, passiert dasselbe:

1. Eine Anfrage wird an den Server gesendet
2. Der Server leitet es weiter an eine Datenbank
3. Der Datenbank antwortet dem Server
4. Der Server erzeugt daraus eine API-Ausgabe und schickt es weiter an den Client
5. Der Client produziert daraus eine neue Artikelübersicht



(5) Aus der API (JSON), HTML erzeugen und in Body-Tag oder entsprechender Ziel-Tag reinschreiben. Das kann mit JavaScript direkt oder eben zB mit React-Komponenten gemacht werden.

(1) Beim Suchen wird eine Anfrage an den Server gesendet

(1) Anfrage mit **fetch** oder **axios**

(2) Der Server gibt die Anfrage weiter an eine Datenbank, holt die Ergebnisse, erzeugt daraus einen entsprechenden API-Code und schickt es zurück zum Client, der aus dem API-Textstrom, die Ausgabe erzeugt

(2) Weiterleitung. Dabei wird aus dem eingegebenen Text SQL-Anfrage erzeugt und an die Datenbank geleitet

(3) Antwort vom Datenbank

(4) API (JSON) wird erstellt

Wird weiter-verarbeitet

Datenbank

Client (Browser)

HTML *(Struktur der Seite)* +

CSS *(Design der Seite)* +

JavaScript *(Aktivität und Bedienbarkeit der Seite)*

Zum Erzeugen einer Webseite notwendig. Darauf bauen Bibliotheken wie React auf.

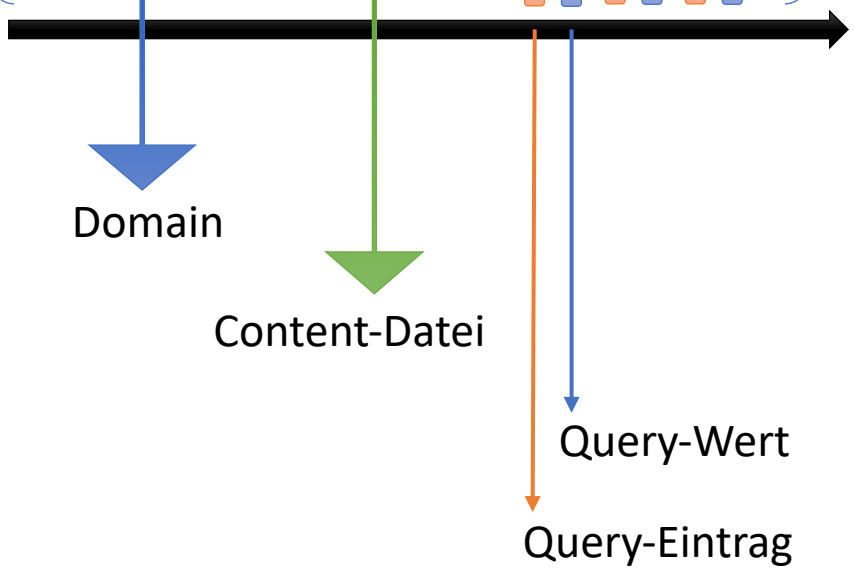
React

Komponente, mit denen der Inhalt des Apps Zusammengesetzt werden

Route bzw. Pfad

*Anfrage mittels **fetch** oder **axios** an den **Server**…*

(https://www.site.com/file.html?a=1&b=2&c=3)



API-Antwort als JSON-Objekt

antwort
{
eintrag: wert, …
}

Server

(zB **NodeJS** oder PHP)

Der Server dient dazu, um Daten zu liefern, aus denen die Seite zusammengesetzt wird.

Datenbank

(MySQL, SQL-Server, PostGreSQL, SQLite)

1. Antwort als Array
2. Umwandlung in ein JSON-Objekt
3. Weiterleitung

Crash-Kurs : HTML

Grundgerüst

```
<!doctype html>

<html>

        <head>
            <title> Die Seite </title>
            <meta charset = ´utf-8´ />
        </head>

        <body>
            <!- - Inhalt - ->
        </body>

</html>
```

Ein HTML-Element der andere Elemente einschließen kann sieht so aus:

<element> … </element>

Es gibt auch HTML-Elemente, die allein sind und nichts einschließen:

<element />

Ein Element kann Attribute haben:

<element attribut='wert'> oder <element attribut='wert' />

Es ist möglich, das ein Element mehrere Attribute erhält. Typische Attribute sind: id, name, class, width, height. Das Ansprechen eines HTML-Elements über JavaScript ist nur im Client (Browser) möglich (nicht in Node) und sieht wie folgt aus:

```
const x = document.getElementById(id)
const x = document.getElementsByName(name)
const x = document.getElemenTagName(`div`)
const x = document.getElementsByClassName(class)
const x = document.querySelector(…)
```

Crash-Kurs : CSS

Grundgerüst

```
<style>

    selektor [, …]
    {
        attribut: wert;
        …
    }

</style>
```

Eine CSS kann auch in eine eigene *.css-Datei liegen. Dann entfällt <style></style> und in der HTML-Datei muss es unterhalb von <head> wie folgt eingebunden werden:

```
<link rel=´stylesheet´ href=´datei.css´>
```

CSS kann auch direkt in ein HTML-Element als Attribut geschrieben werden: <DIV style = ´attribut1:wert1; attribut2: wert2; …´>

Ein Selektor kann sein:

- Ein HTML-Element wie DIV, P, MAIN, SECTION, BR, TABLE, TR, CITE etc.
- Eine eindeutige ID zu einem HTML-Element
- Eine Klasse, dass mehreren HTML-Elementen zugewiesen ist
- Eine Kette aus aufeinander folgenden Selektoren aus allen drei Varianten

Beispiele:

DIV {}	DIV P {}	DIV P A {}
#ID {}	#ID1 #ID2	DIV #ID1 P #ID2 {}
.Klasse {}	.main .sub {}	DIV .klasse #ID {}

.klasse1, .klasse2, DIV P, #id1, #id2, #id3 {}

Crash-Kurs : JavaScript

Grundgerüst

```
<script>
        // Code…
</script>
```

Einbinden der Dateien in JavaScript

```
import Bezeichner from „Quelle“;

import { a, b, c } from „Quelle“;

const Bezeichner = require(„Quelle“);
```

Einbinden der Dateien mittels HTML

```
<script src = „datei.js“></script>
<link rel=„stylesheet“ href=„datei.css“>
```

Erstellen einer Variable, die veränderbar ist:

```
let x;          let x = [];        let x = { … };      let x = () => {};      let x = function() {}
(Variable)      (Array)            (Objekt)            (Arrow Function)       (Anonyme Funktion)
```

Erstellen einer Variable, die schreibgeschützt ist:

```
const x;        const x = [];      const x = { … };    const x = () => {};    const x = function() {}
(Variable)      (Array)            (Objekt)            (Arrow Function)       (Anonyme Funktion)
```

Eine Funktion:

```
function Bez ()
{
   …
}
```

Mit Parametern:

```
function Bez (x, y)
{
      …
}
```

Mit Objektparametern:

```
function Bez ( { a, b, c } )
{
      …
      return x; // Rückgabe
}
```

Eine Schleife:

```
for ( let x = VON; x < BIS; x++ )
{
   …
}
```

Kontrollstruktur If:

```
if ( x == y )
{
      …
}
```

Crash-Kurs : Node

Route bzw. Pfad

Routen werden in Node-JS für zwei Dinge bereitgestellt:

1. Direktes Content liefern
2. Anfragen annehmen und gezieltes Content liefern

Grundgerüst express

```
/* express wird benötigt, um die Kommunikation zwischen Client und Server zu vereinfachen */
const express = require('express‘);
/* Kommunikationsobjekt */
const app = express();

/* Domain und Portnummer */
const host = 'localhost‘;
const port = 9020;

/* Dient dazu, um eingehende Anfragen zu ermitteln und darauf zu reagieren */
app.listen(port, () => {
    // …
});

/* Wird die Route ermittelt, wird automatisch ein Content geladen*/
app.get('/', (req, res) => {
  // …
});
```

Crash-Kurs : SQL

Datenbank

Eine SQL-Datenbank wird mit Queries, also Anfragetexten gesteuert.

Eine Anfrage richtet sich immer an eine Tabelle.

Den Inhalt einer Tabelle abrufen:

SELECT * FROM tabelle;

Den Inhalt einer Tabelle basierend auf eine Bedingung abrufen:

SELECT * FROM tabelle WHERE spalte = wert;

Den Inhalt einer Tabelle basierend auf eine ungefähre Bedingung abrufen:

SELECT * FROM tabelle WHERE spalte LIKE “%...“ oder “…%“ oder “…%...“;

Den Inhalt einer Tabelle basierend auf mehrere Bedingungen abrufen:

SELECT * FROM tabelle WHERE x = a AND y = b AND z = c;

Den abrufenden Inhalt nach einer bestimmten Spalte sortierend:

SELECT * FROM tabelle WHERE …. ORDER BY spalte;
oder
SELECT * FROM tabelle WHERE …. ORDER BY spalte1, spalte2, spalte3;

Absteigend:
SELECT … ORDER BY spalte DESC;

Aufsteigend:
SELECT … ORDER BY spalte ASC;

Einen Eintrag hinzufügen:

INSERT INTO tabelle (spalte1, spalte2, …)
VALUES (wert1, wert2, ...);

Einen Eintrag aktualisieren:

UPDATE tabelle SET spalte1 = wert, spalte2 = wert WHERE spalte3 = x AND spalte4 = y;

Einen Eintrag löschen:

DELETE FROM tabelle WHERE …;

Tabelle erstellen (Details variieren je nach Engine):

CREATE TABLE tabelle …;

Löschen einer Tabelle:

DROP TABLE tabelle;