(1) HTML

A web page consists primarily of HTML. This creates the structure of the website.

The structure of a website is what you see in the browser. It includes such things as menu bar, navigation, footer, images, tables,
Input fields, buttons, lists, links etc.

(2) CSS

To make a web page look nice, CSS is used to color, animate and style nicely the HTML elements that make up the structure of the page.

Surrounding

Colors

Margins

Text formating

Font style

Body structure

Circles

(3) JavaScript

A web page is usually static. This means that once created, it cannot be changed.

But with JavaScript you can make a website dynamic in the browser. This means that the website can be controlled in a targeted manner. It can expand menus, submit forms, switch between visible and invisible elements, change element content and much more.

With JavaScript you can breathe LIFE into a website.

(4) React

A website usually consists of several individual pages. However, if you want to create a professional website or a "web application" or an app, it is a waste of resources to reload the entire browser content with each "content change". This consumes a lot of traffic, memory and time based on the size of the code and the resources that need to be loaded. There are two ways to circumvent this:

1. Reload parts of the website from the server if necessary (Keywords PHP, Node, Ruby, Python +- DB)
2. Designing the website from just one page and making the changes using only JavaScript without having to reload the entire page

React is a JavaScript library that takes the second approach. It creates dynamic websites that consist of only one page. In short -> single page apps. To make the overall work very easy, React melts JavaScript with HTML by dowing support of JSX. This makes it easier to make changes to the page without having to write a lot of JavaScript. Another goal of React is to pack the complex changes to a website into a very simple and, above all, clear concept with the help of components, so that quick familiarization and quick handling of the code is guaranteed at all times.

(5) Node

Normally, JavaScript is only intended for the browser. But Node also brings JavaScript to the server.

This has the advantage that you do not have to learn any other language for server programming and simply program both for the client (browser) and for the server in the same language (JavaScript).

Another advantage is that JavaScript can be better coordinated on both sides. It is possible to write JavaScript codes that can be used equally on both sides (client and server).

The interaction between CLIENT and SERVER

The website itself is generated in the browser and appears on the surfer's screen. But a website consists of elements that come from the server. This includes the CODE (HTML, CSS, JavaScript) and resources (images, videos, audio files, etc.)

A real dynamic website cannot avoid loading certain things from the server as needed. Overall, it doesn't matter whether it's a single-page app or an ordinary website such as Amazon or eBay.

This "reloading" affects, for example, the search results on Google, listing the items on eBay/Amazon, switching to the next item page on Amazon/eBay when a new notification pops up on YouTube, etc.

This is achieved by the client (JavaScript) sending a request to the server (e.g. Node) and then receiving an answer and building it into the website accordingly.

(6) Database

A database is necessary to store and provide large amounts of data.

Normally you could also use text files, but a database can do more. You can search data, filter, adjust at any time, remove data that is no longer needed, add new data.

A database usually resides on the server side and provides data when needed. In the meantime, databases are also available in a slimmed-down form directly on the client (browser).

SQL

SQL (Structured Query Language) is intended solely for managing massive amounts of data. Therefore, SQL databases are primarily on servers.

You can use it for example with NodeJS, PHP, Ruby, Python, JSP, ASPX etc.

On the client (browser), on the other hand, you can use a mini variant of SQL, the SQLite, or a no-SQL database such as MongoDB.

Databases are used to manage customer data, item data, location data, personalized settings, login data and so on.

SQL vs. No-SQL

SQL databases are relational databases. They can be controlled via a request (the so-called queries). A query is usually formulated as text and passed to the database engine. It interprets the request, generates the requested output and sends it back.

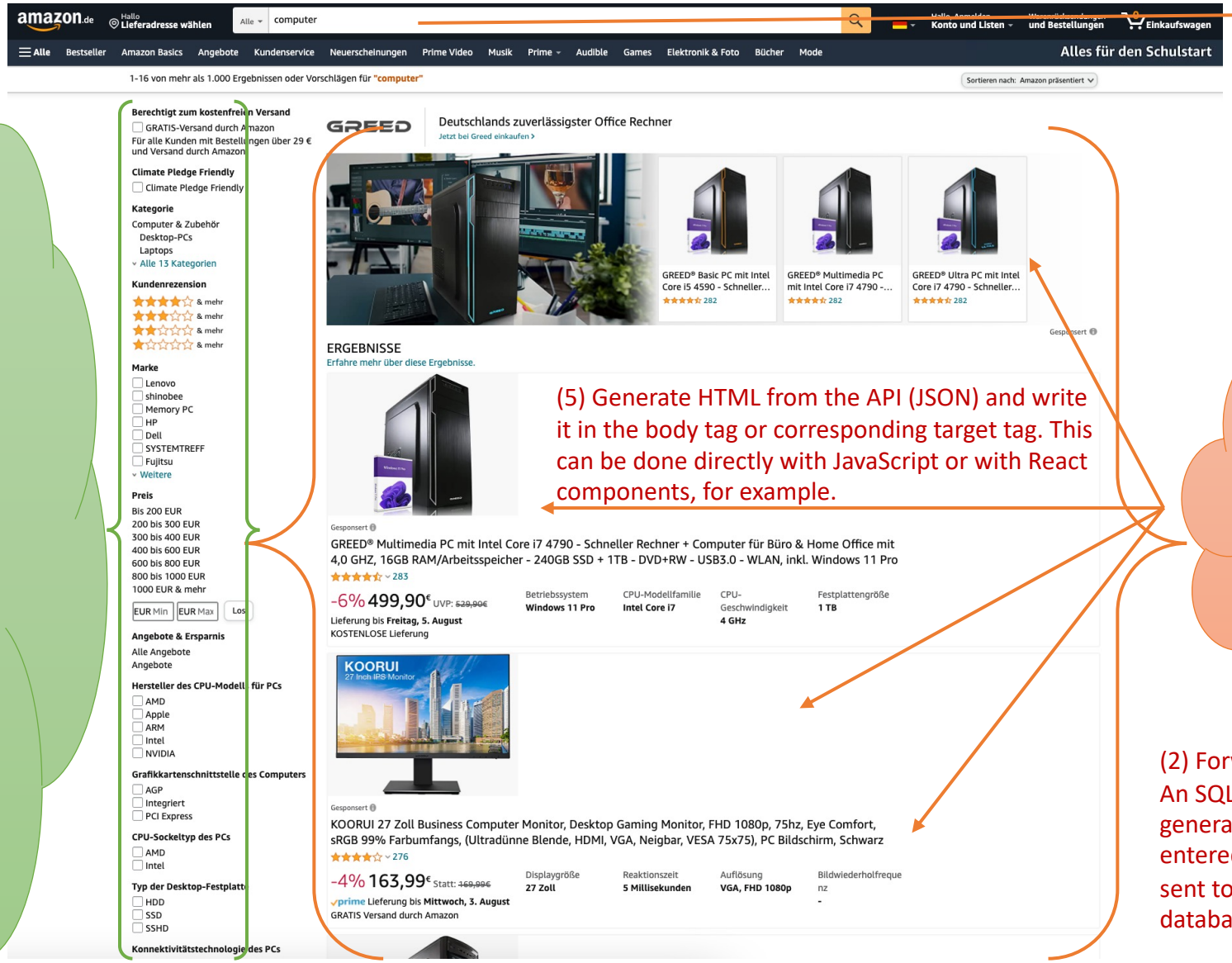
A No-SQL database, on the other hand, works differently. At this point there is no request as text, but for each type of communication there is a corresponding function of the engine, which is simply called.

Therefore a SQL database is much more powerful than a no-SQL database. On the other hand, a no-SQL database is often faster than a SQL database.

A No-SQL database is not intended for large amounts of data.

When a filter option is checked, the same thing happens:

1. A request is sent to the server
1. The server forwards it to a database
1. The database responds to the server
2. The server generates an API output from this and sends it on to the client
3. From this, the client produces a new article overview



(5) Generate HTML from the API (JSON) and write it in the body tag or corresponding target tag. This can be done directly with JavaScript or with React components, for example.

(1) When searching, a request is sent to the server

(1) Request with fetch or axios

(2) The server forwards the request to a database, fetches the results, generates a corresponding API code from it and sends it back to the client, which generates the output from the API text stream

(2) Forwarding. An SQL query is generated from the entered text and sent to the database

(4) API (JSON) will be created

Will be processed

(3) Answer from Database

Database

Client (Browser)

HTML *(Structure of the page)* +

CSS *(Design of the Seite)* +

JavaScript *(Activity and usability of the site)*

Necessary to create a website.
Libraries like React build on this.

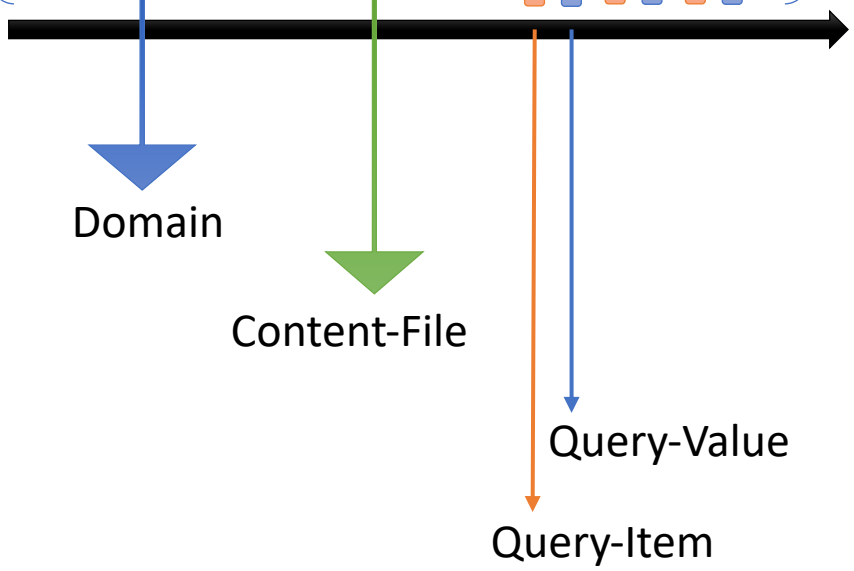
React

Component used to compose the content of the app

Route or Path

Request via fetch or axios to the server...

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



API-Response as JSON-Object

response
{
item: value, …
}

Server

(zB **NodeJS** or PHP)
The server serves to deliver data from which the page is composed.

Database

(MySQL, SQL-Server, PostGreSQL, SQLite)

1. Response as an array
2. Converting into a JSON-onject
3. Forwarding

Crash-Course : HTML

Basic framework

```
<!doctype html>

<html>

        <head>
            <title> The page </title>
            <meta charset = ´utf-8´ />
        </head>

        <body>
            <!- - Content - ->
        </body>

</html>
```

An HTML element that can enclose other elements looks like this :

<element> … </element>

There are also HTML elements that stand alone and do not enclose anything :

<element />

An element can have attributes :

<element attribut='value'> or <element attribut='value' />

It is possible for an element to have multiple attributes. Typical attributes are: id, name, class, width, height. Addressing an HTML element via JavaScript is only possible in the client (browser) (not in Node) and looks like this:

```
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-Course : CSS

Basic framework

```
<style>

    selector [, …]
    {
        attribute: value;
        …
    }

</style>
```

A CSS can also be in a separate *.css file. Then <style></style> is omitted and it must be included in the HTML file below <head> as follows:

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

CSS can also be written directly into an HTML element as an attribute: <DIV style = 'attribut1:wert1; attribute2: value2; …´>

A selector can be:

- An HTML-element like DIV, P, MAIN, SECTION, BR, TABLE, TR, CITE etc.
- A unique id for an HTML element
- A class assigned to multiple HTML elements
- A chain of consecutive selectors from all three variants

Sample:

DIV {} DIV P {} DIV P A {}

#ID {} #ID1 #ID2 DIV #ID1 P #ID2 {}

.class {} .main .sub {} DIV . class #ID {}

. class1, . class2, DIV P, #id1, #id2, #id3 {}

Crash-Course : JavaScript

Basic framework

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

Integrate the files in JavaScript

```
import LABEL from „Source“;

import { a, b, c } from „Source“;

const LABEL = require(„Source“);
```

Integrate the files using HTML

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

Creating a variable that is mutable :

```
let x;                  (Variable)
let x = [];             (Array)
let x = { … };          (Object)
let x = () => {};       (Arrow Function)
let x = function() {}   (Anonymous Function)
```

Creating a variable that is read-only :

```
const x;                  (Variable)
const x = [];             (Array)
const x = { … };          (Object)
const x = () => {};       (Arrow Function)
const x = function() {}   (Anonymous Function)
```

A Function:

```
function Bez ()
{
   …
}
```

With parameters:

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

Mit object-parameter:

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

A loop:

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

Controlling-structure:

```
if ( x == y )
{
      …
}
```

Crash-Course : Node

Route or Path

Routes are provided in Node-JS for two things:

1. Deliver direct content
2. Accept requests and deliver targeted content

Basic framework express

```
/* express is required to simplify communication between client and server */
const express = require('express‘);
/* communication object */
const app = express();

/* domain and port number */
const host = 'localhost‘;
const port = 9020;

/* Used to identify and respond to incoming requests */
app.listen(port, () => {
    // …
});

/* If the route is determined, content is automatically loaded */
app.get('/', (req, res) => {
  // …
});
```

Crash-Course : SQL

Database

An SQL database is controlled with queries, i.e. request texts.

A query is always directed at a table.

Get the contents of a table :

SELECT * FROM sheet;

Get the contents of a table based on a condition :

SELECT * FROM sheet WHERE column = wert;

Retrieve the contents of a table based on an approximate condition :

SELECT * FROM sheet WHERE column LIKE “%...“ or “…%“or “…%...“;

Retrieve the contents of a table based on multiple conditions :

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

Sorting the retrieved content by a specific column :

SELECT * FROM sheet WHERE …. ORDER BY column;
or
SELECT * FROM sheet WHERE …. ORDER BY column1, column2, column3;

Descending :
SELECT … ORDER BY column DESC;

Ascending :
SELECT … ORDER BY column ASC;

Add an entry :

INSERT INTO tabelle (col1, col2, …) VALUES (value1, valu2, ...);

Update an entry :

UPDATE sheet SET col1 = value, col2 = value WHERE col3 = x AND col4 = y;

Delete an entry :

DELETE FROM sheet WHERE …;

Create table (details vary by engine):

CREATE TABLE sheet …;

Delete a table :

DROP TABLE sheet;