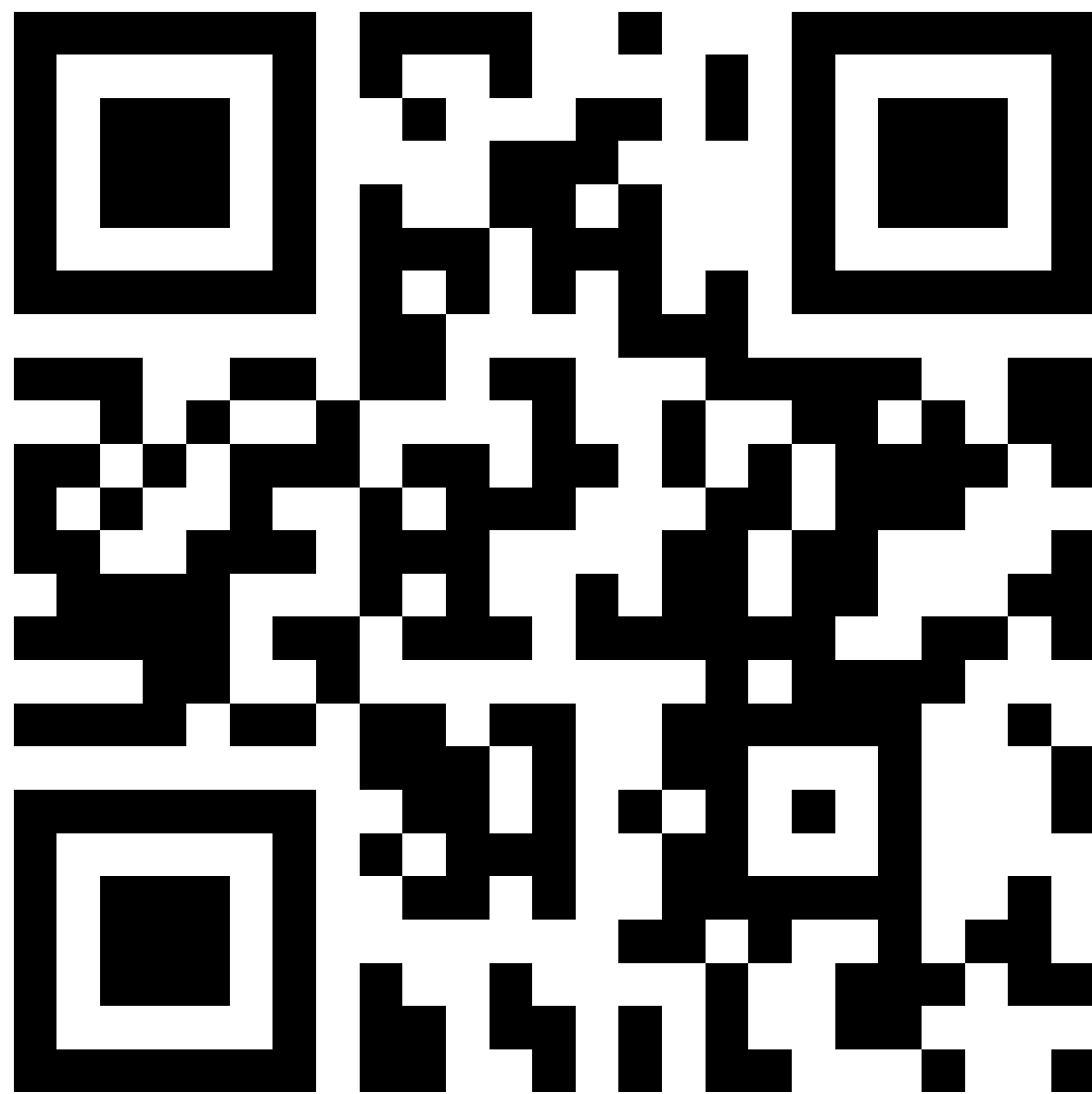


Web Engineering

DHBW Stuttgart · Informatik / Wirtschaftsinformatik

Sommersemester 2026 · Sitzung 3 — JS, Frameworks & Build Tools

<https://librete.ch/dhbw/>



Ressourcen zum Selbstlernen

- **CODE CRISPIES** — codecrispi.es
- **Online Code-Editor** — codepen.io
- **MDN** (Mozilla Developer Network) — developer.mozilla.org/de/
- **Flexbox-Spiel** — flexboxfroggy.com
- **Grid-Spiel** — cssgridgarden.com

JavaScript

ECMAScript

- **Ecma International** – European Computer Manufacturers Association
- Seit **1997** standardisiert als **ECMA-262**
- Implementierungen: **JavaScript** (Netscape, später Mozilla), **JScript** (Microsoft), **ActionScript** (Adobe Flash) etc.
- Aktuelle Version: **ECMAScript 2024 (ES15)**, jährliche Releases

Browser-Kompatibilität:

- Faustregel: **ES2017+** läuft in allen modernen Browsern
- Ältere Targets via Transpiler (Babel, esbuild, swc, tsc)
- Kompatibilitätstabelle: kangax.github.io/compat-table

JavaScript Grundlagen

```
// Variablen  
let name = "Welt";  
const alter = 25;  
  
// Funktionen  
function gruss(name) {  
    return `Hallo, ${name}!`;  
}  
  
// Arrow Functions  
const add = (a, b) => a + b;  
  
// Arrays  
const arr = [1, 2, 3];  
arr.map(x => x * 2); // [2, 4, 6]
```

DOM Manipulation

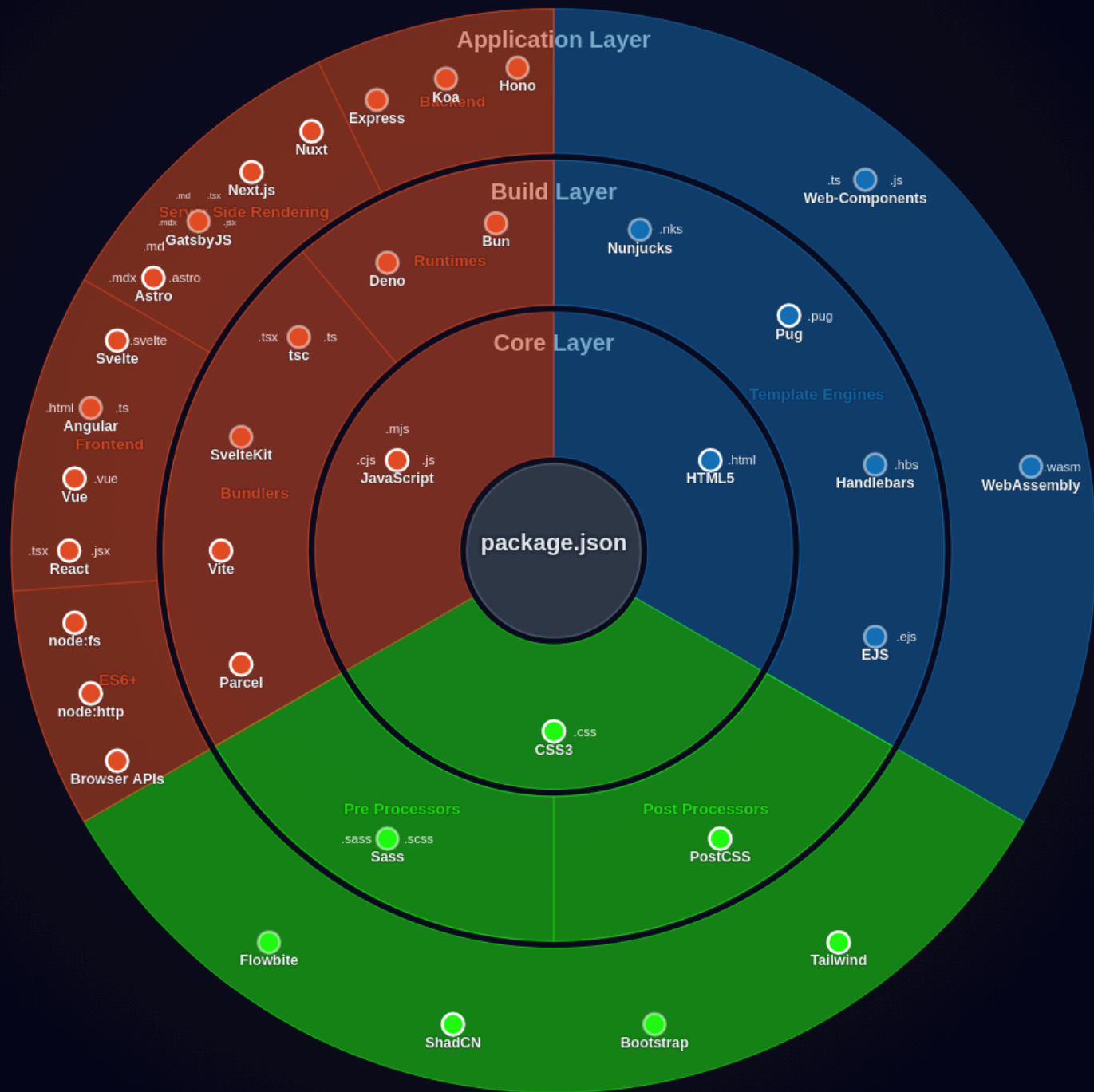
```
// Element auswählen  
const title = document.querySelector('h1');  
  
// Inhalt ändern  
title.textContent = 'Neue Überschrift';  
  
// Event Listener  
button.addEventListener('click', () => {  
    alert('Geklickt!');  
});
```

Fetch API

```
fetch('/api/products')  
  .then(res => res.json())  
  .then(data => {  
    console.log(data);  
  })  
  .catch(err => console.error(err));
```

Async/Await

```
async function loadProducts() {  
  try {  
    const res = await fetch('/api/products');  
    const data = await res.json();  
    return data;  
  } catch (err) {  
    console.error('Fehler:', err);  
  }  
}
```



Web Engineering Mandala

Interaktive version: nextlevelshit.github.io/web-engineering-mandala

3 Schichten:

- **Core Layer** — JS, HTML, CSS (vanilla)
- **Build Layer** — bundlers, runtimes, template engines, pre/post processors
- **Application Layer** — frontend, backend, frameworks, SSR, CSS-utilities

3 Segmente: HTML (blau) · JavaScript (rot) · CSS (grün) · Mittelpunkt: `package.json`

JavaScript Frameworks – Kategorien

Kategorie	Beispiele
CSS Frameworks	Tailwind, Bootstrap, shadcn/ui
Frontend Frameworks	React, Vue, Svelte, Astro
Rendering / Meta	Next.js, Nuxt, Gatsby
Build Tools / Bundler	Webpack, Vite, Parcel, esbuild
Backend Frameworks	Express, Fastify, NestJS

Vanilla JS – Counter

```
<button id="myButton">Clicked 0 times</button>
<script>
  let count = 0;
  const btn = document.getElementById('myButton');
  btn.addEventListener('click', () => {
    count++;
    btn.textContent = `Clicked ${count} times`;
  });
</script>
```

Pro: hohe Kompatibilität, simpel

Contra: State-Management und Re-Rendering werden schnell komplex

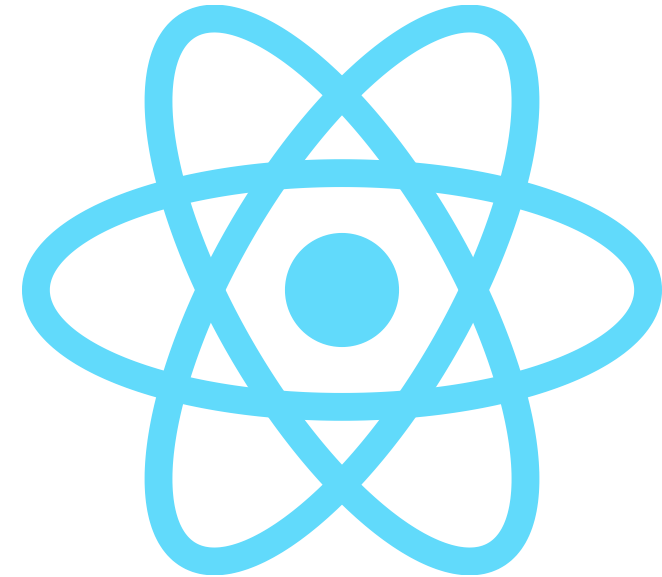
React – Counter (JSX)

```
import { useState } from "react";

function MyButton() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Clicked {count} times
    </button>
  );
}
```

- **Component-Based** Architecture
- **Virtual DOM**, JSX, Hooks
- großes Ökosystem (React Router, Redux, ...)



Vue 3 – Composition API

```
<script setup>
import { ref } from "vue";
const count = ref(0);
</script>

<template>
  <button @click="count++">
    Clicked {{ count }} times
  </button>
</template>
```

- **Reactive Data Binding**
- Single-File Components (`.vue`)
- Vue Router, Pinia (State)



Svelte – Counter

```
<script>
  let count = 0;
</script>

<button on:click={() => count += 1}>
  Clicked {count} {count === 1 ? 'time' : 'times'}
</button>
```

- **Compiler-Based** – kein Runtime-Framework
- Reactive Assignments (**let**)
- Built-in Animationen



Astro – Content-First

```
---  
const greeting = "Hallo DHBW";  
---  
<html>  
  <body>  
    <h1>{greeting}</h1>  
    <MyReactButton client:load />  
  </body>  
</html>
```

- **Islands Architecture** – Komponenten mehrerer Frameworks mischen
- Default: kein JS im Output (nur statisches HTML)
- Hydration on demand (`client:load`, `client:visible`)



Rendering Frameworks

	Frontend	SSR	SSG	File Routing
Next.js	React	ja	ja	<code>/app/page.js</code>
Nuxt	Vue	ja	ja	<code>/pages/*.vue</code>
Gatsby	React	nein	ja	GraphQL Data

Build Tools – Webpack

- Module Bundling (JS, CSS, Images)
- Module Loaders + großes Plugin-Ökosystem
- Code Splitting, HMR
- Konfigurations-lastig (`webpack.config.js`)

→ Standard für viele Legacy-Setups.



Build Tools – Vite

```
// vite.config.js
import { defineConfig } from "vite";

export default defineConfig({
  server: {
    proxy: {
      "/api": {
        target: "http://localhost:4567",
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, "")
      }
    }
  }
});
```

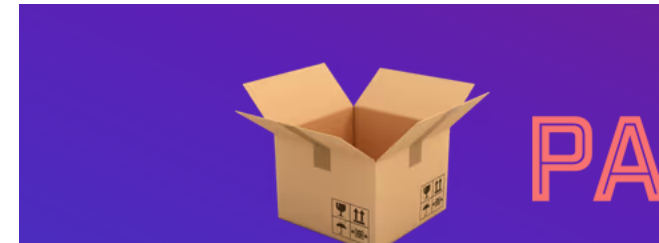


- Rollup unter der Haube, **HMR mit ES Modules**
- Tree-Shaking, Code Splitting, React + Vue out-of-the-box

Build Tools – Parcel

```
{
  "name": "my-project",
  "source": "src/index.html",
  "browserslist": "> 0.5%, last 2 versions, not dead",
  "scripts": {
    "start": "parcel",
    "build": "parcel build"
  },
  "devDependencies": {
    "parcel": "latest"
  }
}
```

- **Zero-Config**
- Auto-Resolution, HMR
- TS / CSS-Preprocessors out-of-the-box



Microfrontends

- Mehrere unabhängige Frontend-Apps in einer Page
- Module Federation (Webpack 5)
- **single-spa**: <https://single-spa.js.org/>
- Anwendungsfall: große Teams, unterschiedliche Tech-Stacks pro Domain

→ Komplexitätskosten: nur bei echtem Skalierungsbedarf.

Demo-Repos

- <https://github.com/nextlevelshit/dhbw-client-js> (Counter, multi-framework)
- <https://github.com/nextlevelshit/node-cache-api> (API + Tests)
- <https://github.com/nextlevelshit/dhbw-docker> (Compose-Stack)