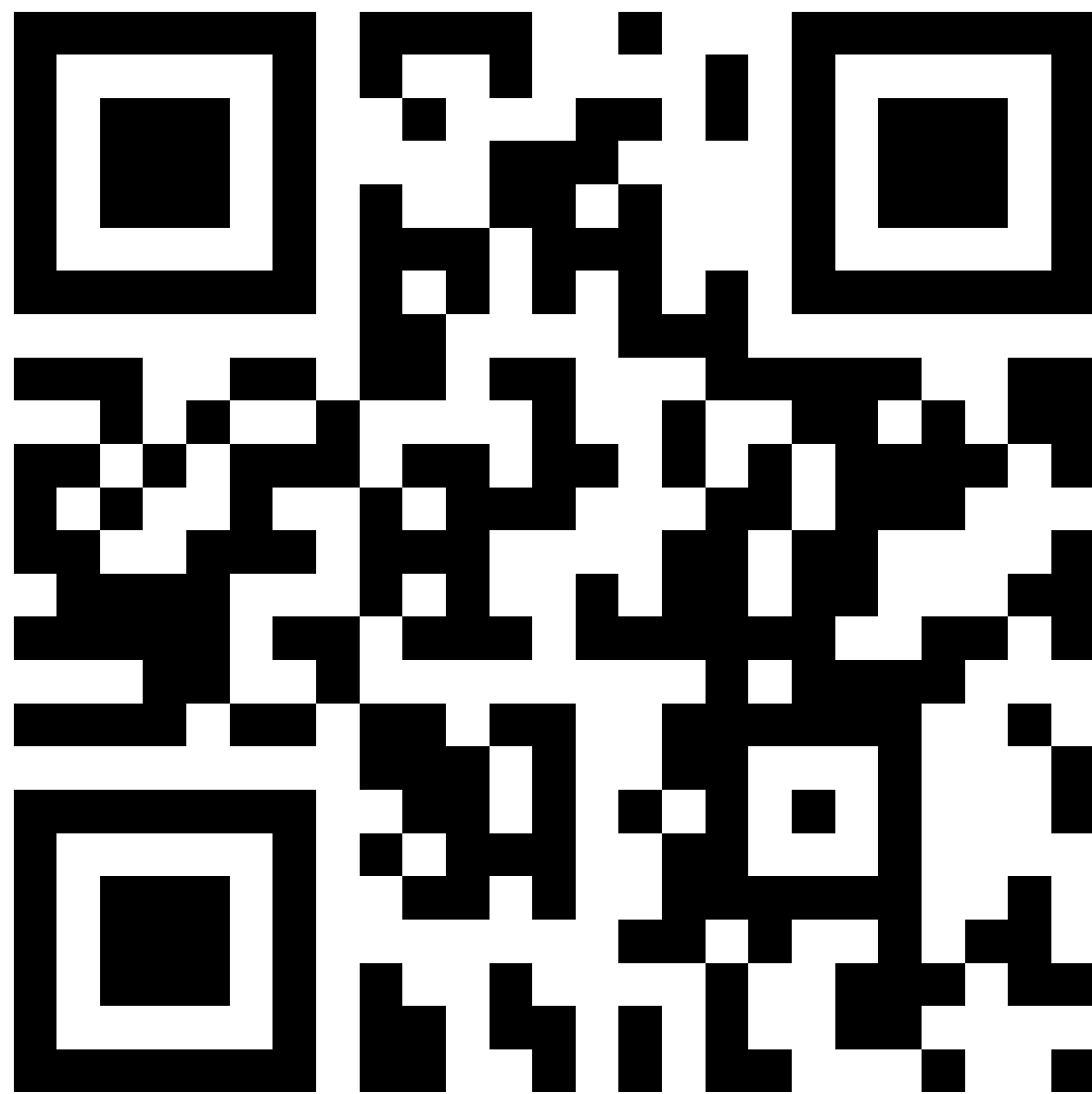


Web Engineering

DHBW Stuttgart · Informatik / Wirtschaftsinformatik

Sommersemester 2026 · Sitzung 5 — Express API, CRUD und Middlewares

<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

Inhalt

1. Node.js – Fun Facts & alternative Runtimes
2. `process` – Signal-Handling, Graceful Shutdown
3. Express – Setup, Routing
4. Middlewares (eigene + built-in)
5. CRUD-Endpunkte (Create, Read, Update, Delete)
6. Error Handling
7. WebSockets

Node.js – Advanced

node.js – Fun Facts

- Entwickelt von Ryan Dahl in 2009
- Erste serverseitige Laufzeitumgebung in JavaScript (C++)
- Inspiriert von Nginx (Non-blocking I/O)

Alternativen heute:

- **Deno** (2020): Rust, Tokio
- **Bun** (2022): Zig

process – Prozess-Steuerung

```
process.on("SIGINT", () => {  
  console.log("Received SIGINT. Closing server...");  
  server.close(() => {  
    console.log("Server closed.");  
    process.exit(0);  
  });  
});  
  
// Signale  
process.exit(0); // Erfolgreich  
process.exit(1); // Fehler
```

node.js Module System

```
// ES Modules (modern, "type": "module")  
import express from "express";  
import { port } from "../src/config/constants.js";  
import { readFileSync } from "node:fs";
```

```
// CommonJS (legacy)  
const express = require("express");  
const fs = require("fs");
```

→ Neue Projekte: ESM. Built-ins immer mit **node:**-Präfix.

HTTP Request mit node-fetch

```
// Node.js 18+ hat fetch eingebaut  
const res = await fetch('https://api.example.com/data');  
const data = await res.json();  
  
// Ältere Versionen: node-fetch  
import fetch from 'node-fetch';
```

Web Server, Builder, Bundler

```
# Boilerplates erstellen  
npm create vite@latest my-app -- --template react  
npm create svelte@latest  
npm create astro@latest  
  
# nuxt  
npm install nuxt  
npm run dev
```

JavaScript Frameworks – Übersicht

Kategorien des modernen JS-Ökosystems

CSS FRAMEWORKS

- **Tailwind CSS**
Utility-first
- **Bootstrap**
Komponenten
- **Pico CSS**
Classless
- **UnoCSS**
Atomic

FRONTEND FRAMEWORKS

- **React**
Meta / Virtual DOM
- **Svelte**
Compiler-based
- **Vue**
Progressive
- **Angular**
Google / Full

RENDERING FRAMEWORKS

- **Next.js**
React / SSR
- **SvelteKit**
Svelte / SSR
- **Nuxt**
Vue / SSR
- **Astro**
Islands Arch.
- **Gatsby**
React / SSG

BUILD TOOLS & BUNDLER

- **Vite**
Dev-Server / HMR
- **Webpack**
Bundler
- **esbuild**
Go / ultra-fast
- **tsc**
TypeScript
- **Rollup**
Libraries

BACKEND FRAMEWORKS

- **Express**
Minimal / de-facto
- **Fastify**
Performant
- **AdonisJS**
Full-stack
- **NestJS**
Angular-style
- **Hono**
Edge-ready

Express – Grundlagen

```
import express from 'express';  
const app = express();  
  
app.listen(port, () => {  
  console.log(`Server läuft auf Port ${port}`);  
});
```

Express – Middleware

```
app.use(express.json());

app.use('/api', (req, res, next) => {
  console.log('Time:', Date.now());
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 2');
  next();
});
```

Express – CRUD Routes

```
app.get('/api/:key', async (req, res) => {  
  const context = cache.get(req.params.key);  
  res.json({ context });  
});  
  
app.post('/api/:key', async (req, res) => {  
  const payload = req.body;  
  res.json({ created: cache.set(req.params.key, payload) });  
});  
  
app.put('/api/:key', async (req, res) => {  
  const payload = req.body;  
  res.json({ updated: cache.update(req.params.key, payload) });  
});  
  
app.delete('/api/:key', async (req, res) => {  
  cache.remove(req.params.key);  
  res.json({ deleted: req.params.key });  
});
```

Express – server.listen

```
const server = app.listen(port, () => {  
  console.log(`Server läuft auf Port ${port}`);  
});  
  
server.on('error', (err) => {  
  console.error('Server error:', err);  
});  
  
server.close(() => {  
  console.log('Server geschlossen');  
});
```

WebSockets

```
import { WebSocketServer } from 'ws';

const wss = new WebSocketServer({ port: 8080 });

wss.on('connection', (ws) => {
  console.log('Client verbunden');

  ws.on('message', (data) => {
    console.log('Empfangen:', data.toString());
    ws.send('Pong');
  });
});
```

TypeScript → JavaScript Compilation

Der TypeScript-Compiler entfernt alle Typ-Annotationen — nur JavaScript bleibt

TYPESCRIPT-DATEI

```
type Result = "pass" | "fail";

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed");
  } else {
    console.log("Failed");
  }
}
```

↓
tsc

TYPEN WERDEN ENTFERNT

```
type Result = "pass" | "fail";

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed");
  } else {
    console.log("Failed");
  }
}
```

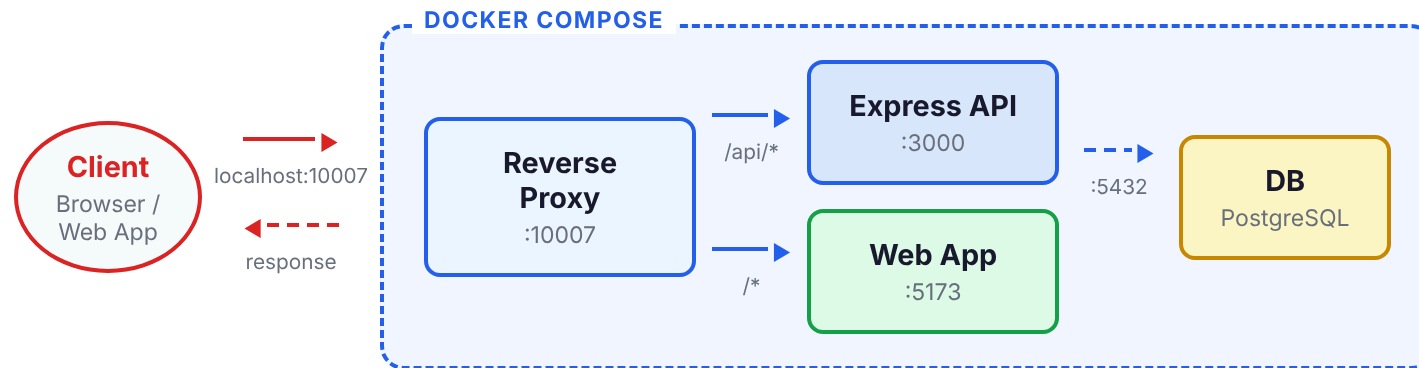
↓
output

JAVASCRIPT-DATEI

```
function verify(result) {
  if (result === "pass") {
    console.log("Passed");
  } else {
    console.log("Failed");
  }
}
```

Docker: Service Orchestration

Client kommuniziert nur mit dem Reverse Proxy — interne Services bleiben isoliert



Test Automation Trophy

Schicht-Modell nach Kent C. Dodds — Breite = empfohlener Anteil

