

Todo App

Eine Lern-Anwendung für Softwarearchitektur

Vom Quick & Dirty zur Clean Architecture

Robert Bretz

10. Mai 2026

Inhaltsverzeichnis

1	Portfolio-Seite: Von Null zum Live-Deployment	3
1.1	Projektstruktur (Monorepo mit pnpm)	3
1.2	Schritt 1: Monorepo initialisieren	3
1.3	Schritt 2: React + Vite + TypeScript einrichten	4
1.4	Schritt 3: Tailwind CSS einrichten	4
1.5	Schritt 4: Erste App-Komponente	5
1.6	Schritt 5: Git initialisieren	5
1.7	Schritt 6: Gitea-Repository anlegen und pushen	6
1.8	Schritt 7: Dockerfile für Produktion	6
1.9	Schritt 8: .npmrc für Build-Scripts	7
1.10	Schritt 9: CI/CD-Pipeline mit Gitea Actions	7
1.11	Schritt 10: Firewall öffnen und deployen	8
1.12	Aufgetretene Fehler und ihre Lösungen	8
1.12.1	Fehler 1: pnpm-workspace.yaml not found	8
1.12.2	Fehler 2: pnpm-lock.yaml not found	8
1.12.3	Fehler 3: ERR_PNPM_IGNORED_BUILDS	9
1.13	Die Seite erreichen	9
1.14	Vollständiger Code: App.tsx mit Tailwind	9
1.15	Tailwind-Klassen im Überblick	11
1.16	Zusammenfassung	11
2	Styling mit Tailwind CSS und Urbanist-Schriftart	11
2.1	Schriftart Urbanist installieren	12
2.2	Index.css – Das Herzstück des Stylings	12
2.2.1	Tailwind Import	12
2.2.2	Schriftart-Definitionen (5 Schnitte)	12
2.2.3	Tailwind Theme	13
2.2.4	Base-Layer (globale Stile)	14
2.2.5	Custom Scrollbar (WebKit)	14
2.2.6	Firefox Scrollbar	15
2.2.7	Hilfsklasse: Scrollbar ausblenden	15
2.3	App.tsx – Die React-Komponente erklärt	16
2.4	Verwendete Tailwind-Klassen – Cheat Sheet	17
2.5	Scrollbar-Debugging	17
2.6	Zusammenfassung	18

1 Portfolio-Seite: Von Null zum Live-Deployment

In diesem Tutorial bauen wir eine komplette Portfolio-Webseite mit React, Vite und Tailwind CSS – von der ersten Codezeile bis zur automatisch deployten Live-Seite per Gitea CI/CD.

1.1 Projektstruktur (Monorepo mit pnpm)

Wir verwenden ein **Monorepo** mit pnpm als Package-Manager. Das ermöglicht, mehrere Projekte (Frontend, Backend) in einem Repository zu verwalten.

Ordnerstruktur nach diesem Schritt:

```
portfolio/
├── apps/
│   └── web/                                # React-Frontend mit Vite + Tailwind
│       ├── src/
│       │   ├── App.tsx                    # Hauptkomponente
│       │   ├── index.css                  # Tailwind-Import
│       │   └── main.tsx                    # Einstiegspunkt
│       ├── package.json                    # Frontend-Abhängigkeiten
│       └── vite.config.ts                  # Vite + Tailwind Konfiguration
├── .gitea/
│   └── workflows/
│       └── deploy.yaml                    # CI/CD Pipeline
├── .gitignore
├── .npmrc                                  # pnpm Build-Scripts erlauben
├── Dockerfile                             # Docker-Build für Produktion
├── package.json                           # Root-Konfiguration
├── pnpm-lock.yaml                         # Lockfile (automatisch erstellt)
└── pnpm-workspace.yaml                    # Workspace-Definition
```

1.2 Schritt 1: Monorepo initialisieren

```
1 cd ~/projects
2 mkdir portfolio
3 cd portfolio
4
5 # pnpm initialisieren
6 pnpm init
7
8 # Workspace-Struktur definieren
9 cat > pnpm-workspace.yaml << 'EOF'
10 packages:
11   - "apps/*"
12 EOF
13
14 # Root package.json anpassen
15 cat > package.json << 'EOF'
16 {
17   "name": "portfolio",
18   "version": "1.0.0",
19   "private": true,
```

```

20 "scripts": {
21   "dev": "pnpm --filter web dev",
22   "build": "pnpm --filter web build"
23 }
24 }
25 EOF
26
27 # Apps-Ordner für das Frontend
28 mkdir -p apps/web

```

Listing 1: Monorepo mit pnpm einrichten

Erklärung der Dateien:

- `pnpm-workspace.yaml` – Teilt pnpm mit, dass alle Ordner unter `apps/` eigenständige Pakete sind
- `package.json` – Root-Konfiguration mit praktischen Scripts. `--filter web` führt den Befehl nur im `apps/web`-Paket aus

1.3 Schritt 2: React + Vite + TypeScript einrichten

```

1 cd ~/projects/portfolio
2
3 # Vite-Projekt mit React und TypeScript erstellen
4 pnpm create vite apps/web --template react-swc-ts
5
6 # In den web-Ordner wechseln
7 cd apps/web
8
9 # Abhängigkeiten installieren
10 pnpm install

```

Listing 2: Vite-Projekt erstellen

Erklärung:

- `pnpm create vite` – Erstellt ein neues Vite-Projekt im angegebenen Ordner
- `--template react-swc-ts` – Verwendet die Vorlage mit React, SWC (schneller Compiler) und TypeScript
- `pnpm install` – Installiert alle Abhängigkeiten aus `package.json`

1.4 Schritt 3: Tailwind CSS einrichten

```

1 cd ~/projects/portfolio/apps/web
2
3 # Tailwind-Pakete installieren
4 pnpm add tailwindcss @tailwindcss/vite
5
6 # vite.config.ts überschreiben
7 cat > vite.config.ts << 'EOF'
8 import { defineConfig } from "vite";
9 import react from "@vitejs/plugin-react";
10 import tailwindcss from "@tailwindcss/vite";
11
12 export default defineConfig({

```

```

13   plugins: [react(), tailwindcss()],
14 });
15 EOF
16
17 # index.css anpassen (nur Tailwind-Import)
18 cat > src/index.css << 'EOF'
19 @import "tailwindcss";
20 EOF

```

Listing 3: Tailwind CSS installieren und konfigurieren

Erklärung:

- tailwindcss – Das Tailwind CSS Framework (Version 4)
- @tailwindcss/vite – Das offizielle Vite-Plugin für Tailwind CSS. Es verarbeitet die Tailwind-Klassen direkt beim Build.
- @import "tailwindcss" – Importiert alle Tailwind-Basis-Styles, Komponenten und Utilities

Wichtig: Tailwind 4 verwendet @import "tailwindcss" statt der alten @tailwind base/components/utilities-Direktiven. Kein tailwind.config.js mehr nötig!

1.5 Schritt 4: Erste App-Komponente

```

1 cat > src/App.tsx << 'EOF'
2 function App() {
3   return (
4     <div className="min-h-screen bg-gray-950 text-white flex items-center justify-center">
5       <h1 className="text-4xl font-bold"> Portfolio</h1>
6     </div>
7   );
8 }
9
10 export default App;
11 EOF

```

Listing 4: Minimale App.tsx

Lokal testen:

```

1 cd ~/projects/portfolio
2 pnpm run dev

```

Listing 5: Entwicklungsserver starten

Im Browser: <http://localhost:5173>

1.6 Schritt 5: Git initialisieren

```

1 cd ~/projects/portfolio
2
3 # .gitignore erstellen
4 cat > .gitignore << 'EOF'
5 node_modules
6 dist

```

```

7 .vs
8 .idea
9 *.db
10 EOF
11
12 # Git initialisieren und ersten Commit machen
13 git init
14 git add .
15 git commit -m "Initial commit: Monorepo mit React + Vite + Tailwind"

```

Listing 6: Git Repository einrichten

1.7 Schritt 6: Gitea-Repository anlegen und pushen

1. Im Browser <http://185.209.229.167:3000> öffnen
2. Rechts oben auf **+** → **New Repository**
3. Name: portfolio, auf **Create Repository** klicken

```

1 git remote add gitea http://185.209.229.167:3000/robre/portfolio.git
2 git push gitea master

```

Listing 7: Gitea als Remote hinzufügen und pushen

Credential Helper (damit Git sich Username/Passwort merkt):

```

1 git config --global credential.helper store

```

Listing 8: Git Credential Helper aktivieren

Beim nächsten Push einmalig Username (robre) und Gitea-Passwort eingeben – danach nie wieder.

1.8 Schritt 7: Dockerfile für Produktion

Da das Portfolio nur aus statischen Dateien besteht (nach dem Vite-Build), brauchen wir einen zweistufigen Docker-Build:

```

1 FROM node:22-alpine AS build
2 WORKDIR /app
3 COPY pnpm-lock.yaml pnpm-workspace.yaml package.json .npmrc ./
4 COPY apps/web/package.json apps/web/
5 RUN npm install -g pnpm && pnpm install --no-frozen-lockfile
6 COPY apps/web/ apps/web/
7 WORKDIR /app/apps/web
8 RUN pnpm run build
9
10 FROM nginx:stable-alpine
11 COPY --from=build /app/apps/web/dist /usr/share/nginx/html
12 EXPOSE 80
13 CMD ["nginx", "-g", "daemon off;"]

```

Listing 9: Dockerfile für das Portfolio

Zeile für Zeile erklärt:

- FROM node:22-alpine AS build – Leichtes Node.js-Image für den Build

- COPY pnpm-lock.yaml pnpm-workspace.yaml package.json .npmrc ./ – Konfigurationsdateien für pnpm. Ohne pnpm-workspace.yaml findet pnpm die Pakete nicht!
- COPY apps/web/package.json apps/web/ – Nur package.json zuerst kopieren (Docker-Cache für schnellere Builds)
- RUN npm install -g pnpm && pnpm install --no-frozen-lockfile – pnpm installieren und Abhängigkeiten installieren
- COPY apps/web/ apps/web/ – Restlichen Code kopieren
- WORKDIR /app/apps/web – Ins Frontend-Verzeichnis wechseln
- RUN pnpm run build – Produktions-Build mit Vite (erstellt dist/)
- FROM nginx:stable-alpine – Neues, schlankes Image für den Webserver
- COPY --from=build /app/apps/web/dist /usr/share/nginx/html – Nur den Build-Output kopieren
- EXPOSE 80 / CMD ["nginx", g, "daemon off;"] – Nginx starten

1.9 Schritt 8: .npmrc für Build-Scripts

```
1 cat > .npmrc << 'EOF'
2 pnpm.onlyBuiltDependencies=*
3 EOF
```

Listing 10: Build-Scripts erlauben

Erklärung: pnpm blockt standardmäßig Build-Scripts aus Sicherheitsgründen. Diese Datei erlaubt alle Build-Scripts – notwendig für Pakete wie @swc/core oder esbuild.

1.10 Schritt 9: CI/CD-Pipeline mit Gitea Actions

```
1 mkdir -p .gitea/workflows
```

Listing 11: Workflow-Ordner erstellen

```
1 name Deploy Portfolio
2 on
3   push
4     branches [ "master" ]
5
6 jobs
7   deploy
8     runs-on ubuntu-latest
9     container
10      image catthehacker/ubuntuct-latest
11     steps
12       - name Checkout
13         uses actions/checkout@v4
14
15       - name Build and Deploy
16         run |
17           docker build -t portfolio:latest .
18           docker stop portfolio 2>/dev/null || true
```

```

19 docker rm portfolio 2>/dev/null || true
20 docker run -d --name portfolio -p 8081:80 portfolio:latest

```

Listing 12: .gitea/workflows/deploy.yaml

Erklärung:

- on: push: branches: ["master"] – Der Workflow läuft bei jedem Push auf master
- runs-on: ubuntu-latest – Virtuelle Maschine für den Job
- container: image: catthehacker/ubuntu:act-latest – Docker-Image mit Ubuntu + Docker CLI
- actions/checkout@v4 – Checkt den Code aus dem Repository aus
- docker build -t portfolio:latest . – Baut das Docker-Image
- docker stop/rm 2>/dev/null || true – Stoppt alten Container (ignoriert Fehler, falls nicht existiert)
- docker run -d -name portfolio -p 8081:80 portfolio:latest – Startet neuen Container auf Port 8081

1.11 Schritt 10: Firewall öffnen und deployen

```

1 ssh testserver "ufw allow 8081/tcp"

```

Listing 13: Port 8081 freigeben

```

1 git add .
2 git commit -m "Dockerfile + CI/CD Pipeline hinzugefügt"
3 git push gitea master

```

Listing 14: Alles pushen – löst Pipeline aus!

1.12 Aufgetretene Fehler und ihre Lösungen

1.12.1 Fehler 1: pnpm-workspace.yaml not found

Fehlermeldung: /pnpm-workspace.yaml: not found

Ursache: Die Datei wurde nie erstellt, weil git init zurückgesetzt wurde.

Lösung: pnpm-workspace.yaml manuell erstellen und committen.

1.12.2 Fehler 2: pnpm-lock.yaml not found

Fehlermeldung: /pnpm-lock.yaml: not found

Ursache: pnpm install wurde nie im Root ausgeführt, daher kein Lockfile.

Lösung: pnpm install im Root ausführen, dann die erstellte pnpm-lock.yaml committen.

1.12.3 Fehler 3: ERR_PNPM_IGNORED_BUILDS

Fehlermeldung: [ERR_PNPM_IGNORED_BUILDS] Ignored build scripts

Ursache: pnpm blockt Build-Scripts aus Sicherheitsgründen.

Lösung: .npmrc mit pnpm.onlyBuiltDependencies=* erstellen.

1.13 Die Seite erreichen

Im Browser:

```
1 http://185.209.229.167:8081
```

Listing 15: Portfolio-URL

Container-Status prüfen:

```
1 ssh testserver "docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}' |  
  grep portfolio"
```

Listing 16: Container-Check

Erwartete Ausgabe:

```
portfolio    Up 2 minutes    0.0.0.0:8081->80/tcp
```

Port-Mapping lesen:

- 0.0.0.0:8081->80/tcp – Von außen über Port 8081 erreichbar, intern läuft Nginx auf Port 80
- Die 0.0.0.0 bedeutet: Auf ALLEN Netzwerkschnittstellen des Servers (IPv4 und IPv6)

1.14 Vollständiger Code: App.tsx mit Tailwind

```
1 function App() {  
2   return (  
3     <div className="min-h-screen bg-gradient-to-br from-slate-900 via-purple-900  
4       to-slate-900 text-white p-8">  
5       <div className="max-w-4xl mx-auto">  
6         <header className="text-center mb-16 pt-20">  
7           <span className="inline-block px-4 py-1 rounded-full bg-emerald-500/20  
8             text-emerald-300 text-sm font-medium mb-4 animate-pulse">  
9             Portfolio 2026  
10          </span>  
11          <h1 className="text-7xl font-bold mb-4 bg-gradient-to-r from-emerald  
12            -400 via-cyan-400 to-purple-400 text-transparent bg-clip-text">  
13            Mein Portfolio  
14          </h1>  
15          <p className="text-xl text-gray-300">  
16            Full-Stack Entwickler & DevOps Enthusiast  
17          </p>  
18        </header>  
19        <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
```

```

18     <div className="group bg-white/5 backdrop-blur rounded-xl p-8 text-
    center hover:bg-white/10 transition-all duration-300 border border-white/10
    hover:border-emerald-500 hover:scale-105 cursor-pointer">
19         <span className="text-5xl block mb-4"> </span>
20         <h2 className="text-2xl font-semibold group-hover:text-emerald-400
    transition-colors">
21             Projekte
22         </h2>
23         <p className="text-gray-400 mt-2">React, .NET, Docker</p>
24     </div>
25
26     <div className="group bg-white/5 backdrop-blur rounded-xl p-8 text-
    center hover:bg-white/10 transition-all duration-300 border border-white/10
    hover:border-cyan-500 hover:scale-105 cursor-pointer">
27         <span className="text-5xl block mb-4"> </span>
28         <h2 className="text-2xl font-semibold group-hover:text-cyan-400
    transition-colors">
29             Skills
30         </h2>
31         <p className="text-gray-400 mt-2">TypeScript, C#, SQL</p>
32     </div>
33
34     <div className="group bg-white/5 backdrop-blur rounded-xl p-8 text-
    center hover:bg-white/10 transition-all duration-300 border border-white/10
    hover:border-purple-500 hover:scale-105 cursor-pointer">
35         <span className="text-5xl block mb-4"> </span>
36         <h2 className="text-2xl font-semibold group-hover:text-purple-400
    transition-colors">
37             Kontakt
38         </h2>
39         <p className="text-gray-400 mt-2">Immer erreichbar</p>
40     </div>
41 </div>
42
43 <div className="mt-16 p-8 bg-white/5 rounded-xl backdrop-blur border
    border-white/10">
44     <h3 className="text-xl font-bold mb-4"> Tailwind Farb-Test</h3>
45     <div className="flex flex-wrap gap-2">
46         {[
47             "bg-red-500", "bg-orange-500", "bg-yellow-500", "bg-green-500",
48             "bg-emerald-500", "bg-cyan-500", "bg-blue-500", "bg-purple-500",
49             "bg-pink-500", "bg-rose-500"
50         ].map((color) => (
51             <div
52                 key={color}
53                 className={`w-12 h-12 rounded-lg ${color} hover:scale-125
    transition-transform cursor-pointer`}
54                 title={color}
55             />
56         ))}
57     </div>
58 </div>
59 </div>
60 </div>
61 );
62 }
63
64 export default App;

```

Listing 17: Vollständige App.tsx mit Tailwind-Styling

1.15 Tailwind-Klassen im Überblick

Tabelle 1: Verwendete Tailwind-Klassen und ihre Bedeutung

Klasse	Bedeutung
<code>min-h-screen</code>	Mindesthöhe = Bildschirmhöhe
<code>bg-gradient-to-br</code>	Hintergrund-Farbverlauf von oben-links nach unten-rechts
<code>from-/via-/to-COLOR</code>	Farben des Farbverlaufs
<code>text-transparent</code>	Text mit Farbverlauf füllen
<code>bg-clip-text</code>	Hintergrund-Weichzeichner (Glass-morphismus)
<code>backdrop-blur</code>	
<code>bg-white/5</code>	Weiß mit 5% Deckkraft
<code>group</code>	Parent für Gruppen-Hover-Effekte
<code>group-hover:text-COLOR</code>	Textfarbe ändert sich bei Hover auf Parent
<code>group-hover:scale-105</code>	Vergrößerung bei Hover auf Parent
<code>transition-all duration-300</code>	Sanfte Übergänge über 300ms
<code>animate-pulse</code>	Pulsierende Animation

1.16 Zusammenfassung

In diesem Tutorial haben wir:

- Ein Monorepo mit pnpm Workspace eingerichtet
- React + Vite + TypeScript + Tailwind CSS 4 installiert
- Ein Dockerfile für den Produktions-Build erstellt
- Eine CI/CD-Pipeline mit Gitea Actions konfiguriert
- Die Seite automatisch bei jedem Push deployed
- Drei typische Fehler analysiert und behoben
- Eine vollständige Portfolio-Landingpage mit Tailwind gestaltet

Die Seite ist live unter: <http://185.209.229.167:8081>

Die Pipeline läuft bei jedem Push auf master automatisch!

2 Styling mit Tailwind CSS und Urbanist-Schriftart

In diesem Schritt gestalten wir unsere Portfolio-Seite mit Tailwind CSS, binden eine eigene Schriftart ein und erstellen einen benutzerdefinierten Scrollbalken. Jede Zeile Code wird ausführlich erklärt.

2.1 Schriftart Urbanist installieren

Urbanist ist eine moderne, serifenlose Schriftart von Google Fonts mit 18 verschiedene Schnitten (Thin bis Black, jeweils normal und italic). Wir verwenden fünf Grundschnitte.

Schritt 1: Schriftart-Dateien ins Projekt kopieren

```
1 cd ~/projects/portfolio
2 cp -r Urbanist apps/web/public/Urbanist
```

Listing 18: Schriftart-Dateien ins Public-Verzeichnis kopieren

Warum public/?

- Alles im public/-Ordner wird von Vite unverändert übernommen
- Dateien sind unter /Urbanist/static/... im Browser erreichbar
- Kein Import nötig – direkter Pfad in CSS

2.2 Index.css – Das Herzstück des Stylings

Die Datei apps/web/src/index.css vereint Tailwind, Schriftarten und Custom-Styles. Hier jede Zeile im Detail erklärt.

2.2.1 Tailwind Import

```
1 @import "tailwindcss";
```

Listing 19: Tailwind CSS importieren

Erklärung:

- @import – CSS-Regel zum Importieren anderer Stylesheets
- "tailwindcss" – Lädt alle Tailwind-Basis-Styles, Komponenten und Utilities
- In Tailwind 4 **kein** @tailwind base/components/utilities mehr!
- Diese einzige Zeile ersetzt Hunderte von handgeschriebenen CSS-Zeilen

2.2.2 Schriftart-Definitionen (5 Schnitte)

```
1 @font-face {
2   font-family: "Urbanist";
3   src: url("/Urbanist/static/Urbanist-Regular.ttf") format("truetype");
4   font-weight: 400;
5   font-style: normal;
6   font-display: swap;
7 }
```

Listing 20: Urbanist Regular (400)

Zeile für Zeile:

- @font-face – CSS-At-Regel zum Definieren einer eigenen Schriftart

- `font-family: Urbanist` – Der Name, unter dem wir die Schrift später verwenden (z.B. in `font-family: var(--font-display)`)
- `src: url(/Urbanist/...)` – Pfad zur Schriftdatei. Beginnt mit `/`, ist also relativ zur Domain-Root
- `format("truetype")` – Sagt dem Browser, welches Format die Datei hat (.ttf = TrueType)
- `font-weight: 400` – Diesen Schnitt für normale Textstärke verwenden (400 = Regular)
- `font-style: normal` – Kein Kursiv (italic wäre `font-style: italic`)
- `font-display: swap` – WICHTIG! Zeigt sofort Text in einer Ersatzschrift an und tauscht sie aus, sobald Urbanist geladen ist. Verhindert "Flash of Invisible Text"(FOIT)

Die 5 definierten Schnitte:

1. `Urbanist-Light.ttf` – Gewicht 300 (dünn, für Fließtext)
2. `Urbanist-Regular.ttf` – Gewicht 400 (normal, Standard)
3. `Urbanist-Medium.ttf` – Gewicht 500 (halbfett)
4. `Urbanist-SemiBold.ttf` – Gewicht 600 (fast fett)
5. `Urbanist-Bold.ttf` – Gewicht 700 (fett, für Überschriften)

2.2.3 Tailwind Theme

```

1 @theme {
2   --font-display: "Urbanist", sans-serif;
3   --breakpoint-3xl: 1920px;
4   --color-primary: #8dff69;
5 }

```

Listing 21: Tailwind Custom Theme

Zeile für Zeile:

- `@theme` – Tailwind 4 Direktive zum Erweitern/Überschreiben des Standard-Themes
- `--font-display: Urbanist, sans-serif` – Definiert eine neue Schrift-Klasse `font-display`. `sans-serif` ist der Fallback, falls Urbanist nicht lädt
- `--breakpoint-3xl: 1920px` – Neuer Breakpoint für sehr große Bildschirme. Nutzbar als `3xl:text-7xl`
- `--color-primary: #8dff69` – Definiert eine neue Farbe `primary` (helles Grün). Nutzbar als `text-primary`, `bg-primary`, `border-primary`, etc.

Verwendung der Custom-Properties in Tailwind:

- `font-display` in `font-family: var(--font-display)`
- `3xl` als Breakpoint: `3xl:grid-cols-4`
- `primary` als Farbe: `text-primary`, `bg-primary/50`, `border-primary`

2.2.4 Base-Layer (globale Stile)

```
1 @layer base {
2   html {
3     font-family: var(--font-display);
4     scroll-behavior: smooth;
5   }
6
7   body {
8     -webkit-font-smoothing: antialiased;
9     -moz-osx-font-smoothing: grayscale;
10    background-color: #000000;
11    color: #ffffff;
12    overflow-x: hidden;
13  }
14 }
```

Listing 22: Globale Stile

Zeile für Zeile:

- @layer base – Tailwinds "baseLayer. Styles hier haben niedrigste Spezifität und können von Utilities überschrieben werden
- font-family: var(--font-display) – Setzt Urbanist als Standardschrift für die gesamte Seite
- scroll-behavior: smooth – Sanftes Scrollen bei Ankerlinks (z.B. #projekte)
- -webkit-font-smoothing: antialiased – Glättet Schrift auf macOS/iOS (WebKit-Browser)
- -moz-osx-font-smoothing: grayscale – Gleiches für Firefox auf macOS
- background-color: #000000 – Schwarzer Hintergrund
- color: #ffffff – Weiße Schrift als Standard
- overflow-x: hidden – Verhindert horizontales Scrollen (wichtig für Animationen, die über den Rand hinausgehen)

2.2.5 Custom Scrollbar (WebKit)

```
1 ::-webkit-scrollbar {
2   width: 16px;
3 }
4
5 ::-webkit-scrollbar-track {
6   background: #1a1a1a;
7 }
8
9 ::-webkit-scrollbar-thumb {
10  background: linear-gradient(180deg, #8dff69, #6fe047);
11  border-radius: 8px;
12 }
13
14 ::-webkit-scrollbar-thumb:hover {
15  background: linear-gradient(180deg, #a8ff8d, #5bc92e);
16 }
```

Listing 23: Scrollbar für Chrome, Edge, Safari

Zeile für Zeile:

- `::-webkit-scrollbar` – Pseudo-Element für die gesamte Scrollbar (Breite, Hintergrund)
- `width: 16px` – Breite der Scrollbar. Größer = dickerer Balken
- `::-webkit-scrollbar-track` – Der "Hintergrund" der Scrollbar (die Schiene)
- `background: #1a1a1a` – Dunkelgrauer Track (fast schwarz)
- `::-webkit-scrollbar-thumb` – Der bewegliche "Griff" der Scrollbar
- `linear-gradient(180deg, #8dff69, #6fe047)` – Farbverlauf von hellgrün (oben) nach dunkelgrün (unten). `180deg` = von oben nach unten
- `border-radius: 8px` – Abgerundete Ecken am Griff
- `::-webkit-scrollbar-thumb:hover` – Wenn die Maus über dem Griff ist
- `#a8ff8d, #5bc92e` – Hellere Grüntöne beim Hover für visuelles Feedback

2.2.6 Firefox Scrollbar

```
1 * {  
2   scrollbar-width: auto;  
3   scrollbar-color: #6fe047 #1a1a1a;  
4 }
```

Listing 24: Scrollbar für Firefox

Erklärung:

- Firefox unterstützt `::-webkit-scrollbar` **nicht**
- `scrollbar-width: auto` – Normale Breite (Alternative: `thin`)
- `scrollbar-color: GRIFF SCHIENE` – Erste Farbe = Griff, zweite Farbe = Schiene

2.2.7 Hilfsklasse: Scrollbar ausblenden

```
1 .hide-scrollbar::-webkit-scrollbar {  
2   display: none;  
3 }
```

Listing 25: Hilfsklasse für Karussells

Für horizontale Karussells – der Inhalt scrollt, aber die Scrollbar ist unsichtbar.

2.3 App.tsx – Die React-Komponente erklärt

```
1 function App() {
2   return (
3     // Äußerster Container
4     // min-h-screen: mindestens Bildschirmhöhe (wichtig für zentrierte Inhalte)
5     // bg-black: schwarzer Hintergrund
6     // text-white: weiße Schrift als Basis
7     // p-8: 32px Innenabstand auf allen Seiten
8     <div className="min-h-screen bg-black text-white p-8">
9
10      {/* Zentrierte Content-Box */}
11      {/* text-center: zentriert Text und Inline-Elemente */}
12      {/* max-w-2xl: maximale Breite 672px */}
13      {/* mx-auto: automatischer horizontaler Margin = zentriert die Box */}
14      {/* pt-20: 80px Abstand nach oben (Padding-Top) */}
15      <div className="text-center max-w-2xl mx-auto pt-20">
16
17        {/* Kategorie-Badge: Kleine Beschriftung ÜBER der Überschrift */}
18        {/* text-primary: verwendet unsere Custom-Farbe #8dff69 */}
19        {/* text-sm: kleinere Schriftgröße (14px) */}
20        {/* font-medium: halbfette Schrift (Gewicht 500) */}
21        {/* tracking-widest: sehr breite Buchstabenabstände */}
22        {/* uppercase: GROSSBUCHSTABEN */}
23        <span className="text-primary text-sm font-medium tracking-widest
24        uppercase">
25          Full-Stack Entwickler
26        </span>
27
28        {/* Hauptüberschrift: Dein Name */}
29        {/* text-7xl: sehr große Schrift (72px) */}
30        {/* font-bold: fette Schrift (Gewicht 700) */}
31        {/* mt-4: 16px Abstand nach oben */}
32        {/* mb-6: 24px Abstand nach unten */}
33        {/* leading-tight: enger Zeilenabstand (1.25) */}
34        <h1 className="text-7xl font-bold mt-4 mb-6 leading-tight">
35          Robert Bretz
36        </h1>
37
38        {/* Beschreibungstext */}
39        {/* text-xl: etwas größerer Text (20px) */}
40        {/* text-gray-400: graue Schrift (gedämpfter als weiß) */}
41        {/* max-w-lg: maximale Breite 512px (schmäler = besser lesbar) */}
42        {/* leading-relaxed: lockerer Zeilenabstand (1.625) */}
43        <p className="text-xl text-gray-400 max-w-lg mx-auto leading-relaxed">
44          Ich baue moderne Webanwendungen mit React, .NET und Docker.
45          Minimalistisch, schnell und zuverlässig.
46        </p>
47
48        {/* Button-Gruppe */}
49        {/* mt-10: 40px Abstand nach oben */}
50        {/* flex: Flexbox-Layout */}
51        {/* gap-4: 16px Abstand zwischen den Buttons */}
52        {/* justify-center: Buttons horizontal zentrieren */}
53        <div className="mt-10 flex gap-4 justify-center">
54
55          {/* Button "Projekte" */}
56          {/* px-6: 24px horizontaler Innenabstand */}
57          {/* py-3: 12px vertikaler Innenabstand */}
58          {/* bg-primary: Hintergrund in Custom-Farbe #8dff69 */}
```



```

58      {/* text-black: schwarze Schrift auf grünem Grund */}
59      {/* font-semibold: halbfette Schrift (Gewicht 600) */}
60      {/* rounded-full: komplett runde Ecken (Pillenform) */}
61      {/* hover:opacity-80: beim Hover auf 80% Deckkraft */}
62      {/* transition: sanfter Übergang */}
63      <a href="#" className="px-6 py-3 bg-primary text-black font-semibold
rounded-full hover:opacity-80 transition">
64          Projekte
65      </a>
66
67      {/* Button "Kontakt" */}
68      {/* border border-gray-700: dunkelgrauer Rahmen statt Hintergrund */}
69      {/* text-gray-300: hellgraue Schrift */}
70      {/* hover:border-primary: Rahmen wird grün beim Hover */}
71      {/* hover:text-primary: Schrift wird grün beim Hover */}
72      <a href="#" className="px-6 py-3 border border-gray-700 text-gray-300
font-semibold rounded-full hover:border-primary hover:text-primary transition
">
73          Kontakt
74      </a>
75  </div>
76 </div>
77
78  {/* SCROLL-TEST: Erzwingt Scrollen für den Custom Scrollbar */}
79  {/* mt-20: 80px Abstand nach oben */}
80  {/* space-y-8: 32px Abstand zwischen den Kind-Elementen */}
81  {/* max-w-2xl: gleiche maximale Breite wie oben */}
82  <div className="mt-20 space-y-8 max-w-2xl mx-auto">
83      {/* JavaScript: Erstelle 5 Test-Blöcke */}
84      {/* [...Array(5)]: Array mit 5 leeren Plätzen */}
85      {/* .map( (_, i): für jedes Element, _ = ignorierter Wert, i = Index
(0-4) */}
86      {[...Array(5)].map( (_, i) => (
87          <div
88              key={i}
89              className="h-64 bg-gray-900 rounded-xl flex items-center justify-
center text-gray-500 text-2xl"
90          >
91              Sektion {i + 1}
92          </div>
93      )]}
94  </div>
95 </div>
96 );
97 }
98
99 export default App;

```

Listing 26: Vollständige App.tsx mit Zeilen-Erklärung

2.4 Verwendete Tailwind-Klassen – Cheat Sheet

2.5 Scrollbalken-Debugging

Problem: "Ich sehe den Scrollbalken nicht!"

Ursache: Der Scrollbalken erscheint nur, wenn die Seite tatsächlich scrollbar ist. Bei kurzen Inhalten (nur Name + Buttons) passt alles auf eine Bildschirmseite – kein

Tabelle 2: Alle verwendeten Tailwind-Klassen und ihre CSS-Entsprechung

Klasse	CSS-Entsprechung	Erklärung
min-h-screen	min-height: 100vh	Mindestens Bildschirmhöhe
bg-black	background: #000	Schwarzer Hintergrund
text-white	color: #fff	Weißer Schrift
p-8	padding: 2rem	32px Innenabstand
text-center	text-align: center	Text zentrieren
max-w-2xl	max-width: 42rem	Max 672px breit
mx-auto	margin-left/right: auto	Horizontal zentrieren
pt-20	padding-top: 5rem	80px oben

Scrollbalken nötig.

Lösungen:

1. Browser-Fenster vertikal verkleinern
2. Genug Test-Inhalt einfügen (5 große Divs wie im Code oben)
3. Mit `overflow-y: scroll` einen permanenten Scrollbalken erzwingen

2.6 Zusammenfassung

In diesem Schritt haben wir:

- Die Schriftart Urbanist mit 5 Schnitten eingebunden
- Ein Tailwind Custom-Theme mit eigener Farbe und Breakpoint definiert
- Globale Stile im `@layer base` gesetzt
- Einen benutzerdefinierten Scrollbalken mit Farbverlauf erstellt
- Jede Zeile der `App.tsx` und `index.css` ausführlich dokumentiert