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

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

```
cd ~/projects
mkdir portfolio
cd portfolio

# pnpm initialisieren
pnpm init

# Workspace-Struktur definieren
cat > pnpm-workspace.yaml << 'EOF'
packages:
  - "apps/*"
EOF

# Root package.json anpassen
cat > package.json << 'EOF'
{
  "name": "portfolio",
  "version": "1.0.0",
  "private": true,
```

```
  "scripts": {
    "dev": "pnpm --filter web dev",
    "build": "pnpm --filter web build"
  }
}
EOF

# Apps-Ordner für das Frontend
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

```
cd ~/projects/portfolio

# Vite-Projekt mit React und TypeScript erstellen
pnpm create vite apps/web --template react-swc-ts

# In den web-Ordner wechseln
cd apps/web

# Abhängigkeiten installieren
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

```
cd ~/projects/portfolio/apps/web

# Tailwind-Pakete installieren
pnpm add tailwindcss @tailwindcss/vite

# vite.config.ts überschreiben
cat > vite.config.ts << 'EOF'
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import tailwindcss from "@tailwindcss/vite";

export default defineConfig({
```

```
  plugins: [react(), tailwindcss()],
});
EOF

# index.css anpassen (nur Tailwind-Import)
cat > src/index.css << 'EOF'
@import "tailwindcss";
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

```
cat > src/App.tsx << 'EOF'
function App() {
  return (
    <div className="min-h-screen bg-gray-950 text-white flex items-center
    justify-center">
      <h1 className="text-4xl font-bold"> Portfolio</h1>
    </div>
  );
}

export default App;
EOF
```

Listing 4: Minimale App.tsx

Lokal testen:

```
cd ~/projects/portfolio
pnpm run dev
```

Listing 5: Entwicklungsserver starten

Im Browser: `http://localhost:5173`

1.6 Schritt 5: Git initialisieren

```
cd ~/projects/portfolio

# .gitignore erstellen
cat > .gitignore << 'EOF'
node_modules
dist
```

```
.vs
.idea
*.db
EOF

# Git initialisieren und ersten Commit machen
git init
git add .
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

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

Listing 7: Gitea als Remote hinzufügen und pushen

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

```
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:

```
FROM node:22-alpine AS build
WORKDIR /app
COPY pnpm-lock.yaml pnpm-workspace.yaml package.json .npmrc ./
COPY apps/web/package.json apps/web/
RUN npm install -g pnpm && pnpm install --no-frozen-lockfile
COPY apps/web/ apps/web/
WORKDIR /app/apps/web
RUN pnpm run build

FROM nginx:stable-alpine
COPY --from=build /app/apps/web/dist /usr/share/nginx/html
EXPOSE 80
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

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

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

Listing 11: Workflow-Ordner erstellen

```
name Deploy Portfolio
on
  push
    branches [ "master" ]

jobs
  deploy
    runs-on ubuntu-latest
    container
      image catthehacker/ubuntuact-latest
    steps
      - name Checkout
        uses actions/checkout@v4

      - name Build and Deploy
        run |
          docker build -t portfoliolatest .
          docker stop portfolio 2>/dev/null || true
```

```
19          docker rm portfolio 2>/dev/null || true
20          docker run -d --name portfolio -p 808180 portfoliolatest
```

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:

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

Listing 15: Portfolio-URL

Container-Status prüfen:

```
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

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

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