Projektsetup mit SvelteKit

SvelteKit ist das offizielle Meta-Framework rund um Svelte. Es bringt alles mit, was moderne Webentwicklung braucht: Dateibasiertes Routing, SSR, API-Routen, Layouts, und eine cleane DX. In diesem Artikel wird Schritt für Schritt erklärt, wie man ein neues Projekt mit SvelteKit aufsetzt. Dabei werden die Optionen der SvelteKit CLI beleuchtet und sinnvolle Add-ons wie **TailwindCSS**, **Prettier**, **ESLint** sowie der passende **Adapter** für das Deployment ausgewählt.

Voraussetzungen

- Node.js (empfohlen: aktuelle LTS-Version)
- Ein Terminal (z. B. iTerm2, VS Code Terminal)
- Paketmanager, Vorzugsweise NPM

Projekt erstellen

Initialisiert wird das Projekt mit folgendem Befehl:

```
npx sv create my-sveltekit-project
```

`my-sveltekit-project` kann dabei durch einen beliebigen Projektnahmen ersetzt werden. `npx` muss ggf. je nach verwendetem Package-Manager angepasst werden.

Bei erstmaliger Ausführung wird das CLI-Paket automatisch installiert. Bestätigung erfolgt mit y.

```
Need to install the following packages:
sv@0.8.1
Ok to proceed? (y) y
```

SvelteKit CLI: Optionen im Überblick

Nun wird das Projekt interaktiv konfiguriert.

1. Template-Auswahl

```
Which template would you like?
❯ SvelteKit minimal (barebones scaffolding for your new app)
  SvelteKit demo
  Svelte library
```

- **SvelteKit minimal**: Reduziertes Grundgerüst für ein eigenes Projekt ohne Beispielcode.
- **SvelteKit demo**: Enthält Beispielseiten und eine Beispielnavigation.
- **Svelte library**: Template zur Entwicklung wiederverwendbarer Svelte-Komponenten.

Wir wählen für diese Demonstration SvelteKit minimal.

2. TypeScript-Unterstützung

```
Add type checking with TypeScript?
❯ Yes, using TypeScript syntax
  Yes, using JavaScript with JSDoc comments
  No
```

Empfohlen wird die Verwendung der TypeScript-Syntax, um statische Typüberprüfung und IntelliSense zu nutzen.

3. Erweiterungen im Projektsetup

Nach der Projekterstellung fragt das CLI nach zusätzlichen Tools und Bibliotheken, die dem Projekt hinzugefügt werden sollen. Die Auswahl erfolgt interaktiv über die Leertaste (Mehrfachauswahl möglich). Die Optionen sind:

```
What would you like to add to your project?
☐ prettier          (formatter – https://prettier.io)
☐ eslint            (linter – https://eslint.org)
☐ vitest            (unit testing – https://vitest.dev)
☐ playwright        (e2e testing – https://playwright.dev)
☐ tailwindcss       (CSS utility framework – https://tailwindcss.com)
☐ sveltekit-adapter (adapter for deployment – https://kit.svelte.dev/docs#adapters)
☐ drizzle           (ORM – https://orm.drizzle.team/)
☐ lucia             (auth – https://lucia-auth.com)
☐ mdsvex            (Markdown + Svelte – https://mdsvex.pngwn.io/)
```

❒ paraglide (i18n – https://paraglide.dev/)

❒ storybook (component explorer – https://storybook.js.org)

Erweiterung	Beschreibung	Einsatzzweck
Prettier	Automatisches Codeformatierungs-Tool. Definiert einheitlichen Stil für z. B. Einrückung, Semikolons.	Formatierung & Clean Code
ESLint	Linter zur statischen Analyse von JavaScript/TypeScript-Code. Findet potenzielle Fehler & Stilprobleme.	Codequalität und Fehlervorbeugung
Vitest	Schnelles Unit-Testing-Framework, optimiert für Vite und moderne Frontends.	Komponententests, Logiktests
Playwright	Framework für End-to-End-Tests. Ermöglicht UI-Tests mit echten Browserinstanzen.	Testen von Userflows & Accessibility
TailwindCSS	Utility-first CSS-Framework für schnelles und responsives Styling.	Styling über Utility-Klassen
SvelteKit Adapter	Bindeglied zwischen dem Framework und der Zielplattform (z. B. statisches HTML, SSR, Vercel etc.).	Deployment-Anpassung
Drizzle	TypeScript-ORM für SQL-Datenbanken mit gutem DX.	Datenbankzugriff (PostgreSQL etc.)
Lucia	Authentifizierungs-Framework mit Fokus auf Einfachheit und Sicherheit.	Login-Mechanismen, Zugriffskontrolle
mdsvex	Markdown-Präprozessor für Svelte-Komponenten. Kombination von Markdown und Svelte möglich.	Content Management, Dokusysteme
Paraglide	Internationalisierungs-Tool mit Compile-Time-Optimierung.	Mehrsprachigkeit (i18n)
Storybook	Tool zur Visualisierung und Dokumentation einzelner UI-Komponenten.	Komponentenkatalog & Dokumentation

In diesem Beispiel wählen wir: prettier, eslint, tailwindcss, sveltekit-adapter

4. TailwindCSS Plugins

Wir haben uns im Schritt zuvor entschieden, **TailwindCSS** in das Projekt einzubinden. TailwindCSS ist ein Utility-first CSS-Framework, das die Gestaltung von Benutzeroberflächen durch vorgefertigte

CSS-Klassen stark vereinfacht. Nach Auswahl von tailwindcss im Add-on-Menü wird abgefragt, welche **Plugins** integriert werden sollen:

```
tailwindcss: Which plugins would you like to add?
❐ typography (@tailwindcss/typography)
❐ forms      (@tailwindcss/forms)
```

Plugin	Beschreibung
Typography	Stellt sinnvolle Standard-Styles für typografische Inhalte bereit. Eignet sich besonders für Content-lastige Seiten, z. B. Dokumentation oder Blogbeiträge.
Forms	Vereinheitlicht das Styling von nativen HTML-Formular-Elementen wie input, select, textarea etc. Passt sich automatisch an das Tailwind-Designsystem an.

Wir wählen beide Plugins aus.

5. Auswahl des SvelteKit Adapters

SvelteKit ist ein Meta-Framework, das Applikationen sowohl statisch als auch serverseitig oder als Hybrid generieren kann. **Adapter** übernehmen dabei die Aufgabe, die Anwendung für eine bestimmte Zielplattform aufzubereiten (z. B. als statische HTML-Seiten oder als Server-Handler für Vercel, Netlify, Node.js etc.).

Die Auswahl erfolgt im CLI nach Aktivierung des Add-ons `sveltekit-adapter`:

```
sveltekit-adapter: Which SvelteKit adapter would you like to use?
  auto
  node
❯ static (@sveltejs/adapter-static)
  vercel
  cloudflare-pages
  netlify
```

Adapter	Beschreibung	Typ	Empfohlene Einsatzzwecke
auto	Automatische Auswahl anhand der Umgebung. Praktisch für Entwicklung, aber nicht für produktives Deployment empfohlen.	auto-detect	Nur lokale Nutzung, z. B. für Tests

Adapter	Beschreibung	Typ	Empfohlene Einsatzzwecke
node	Erstellt ein Node.js-Handler zur Laufzeit, z. B. für Express, Fastify oder Cloud-Server.	SSR	Eigener Serverbetrieb, z. B. VPS, Docker-Container
static	Exportiert alle Seiten als HTML/CSS/JS. Kein dynamisches Routing, aber sehr performant.	SSG (Static Site)	GitHub Pages, Netlify, klassische Webserver
vercel	Optimierung für das Vercel-Ökosystem. Deployment erfolgt über Vercel CLI oder GitHub-Integration.	Edge-/SSR-ready	Hosting über vercel.com
cloudflare-pages	Unterstützung für Cloudflare Pages, inkl. Worker-Skripte.	Edge SSR	Deployment auf pages.cloudflare.com
netlify	Adapter mit Funktionen wie Functions, Redirects, SSR via Netlify.	Hybrid	Deployment auf netlify.com

Für unser Beispiel wählen wir `static`.

6. Wahl des Paketmanagers

```
Which package manager do you want to install dependencies with?
❯ npm
  yarn
  pnpm
  bun
  deno
```

Die Auswahl erfolgt entsprechend der individuellen Präferenz. `npm` ist in den meisten Fällen ausreichend.

7. Nächste Schritte nach dem Setup

Nach erfolgreichem Setup wird vom CLI folgender Ablauf vorgeschlagen:

```
cd my-sveltekit-project
git init && git add -A && git commit -m "Initial commit"   # optional
```

```
npm run dev -- --open
```

Damit wird der lokale Entwicklungsserver gestartet. Der Zugriff erfolgt typischerweise unter http://localhost:5173.

Projektstruktur nach dem Setup

Nach erfolgreichem Setup mit der SvelteKit CLI liegt eine vorstrukturierte Projektbasis vor. Diese Struktur ermöglicht den sofortigen Einstieg in die Anwendungsentwicklung, getrennt nach Konfiguration, Komponenten, Routen und Styling.

Ein typisches Grundgerüst (vereinfacht dargestellt):

```
my-sveltekit-project/
├── src/
│   ├── lib/
│   │   └── components/     ← Wiederverwendbare UI-Komponenten
│   ├── routes/             ← Seitenstruktur (basierend auf Datei-Routing)
│   │   └── +page.svelte    ← Startseite (Route: "/")
│   └── app.css             ← TailwindCSS-Einstiegspunkt
├── static/                 ← Öffentliche Assets (z. B. Bilder, favicon)
├── svelte.config.js        ← Zentrale SvelteKit-Konfiguration
├── tailwind.config.cjs     ← TailwindCSS-Konfiguration
├── postcss.config.cjs      ← PostCSS-Integration für Tailwind
├── tsconfig.json           ← TypeScript-Projektdefinition
├── package.json            ← Abhängigkeiten und Skripte
└── vite.config.js          ← Build-Tool-Konfiguration (Vite)
```

Pfad	Beschreibung
src/routes/	Implementierung der Seitenstruktur der Anwendung. Jede Datei oder jeder Ordner stellt eine Route dar.
src/routes/+page.svelte	Einstiegspunkt der Anwendung (Startseite, Route /). Kann sofort bearbeitet werden.
src/lib/	Globale Hilfsmittel, z. B. Komponenten, Stores, Services. Nicht Routen-spezifisch.
src/app.css	Haupt-Stylesheet. Hier wird TailwindCSS eingebunden (@tailwind base, components, utilities).
static/	Enthält öffentlich zugängliche Dateien (z. B. robots.txt, favicon.ico, statische Bilder).
svelte.config.js	Konfiguriert Adapter, Preprocessing, Pfade etc.
tailwind.config.cjs	Definiert Tailwind-Themes, Plugins und Pfade zur Content-Erkennung.

Pfad	Beschreibung
tsconfig.json	Konfiguriert das Verhalten des TypeScript-Compilers.

Revision #6
Created 1 April 2025 06:55:48 by Marius Klein
Updated 16 April 2025 13:20:54 by Marius Klein