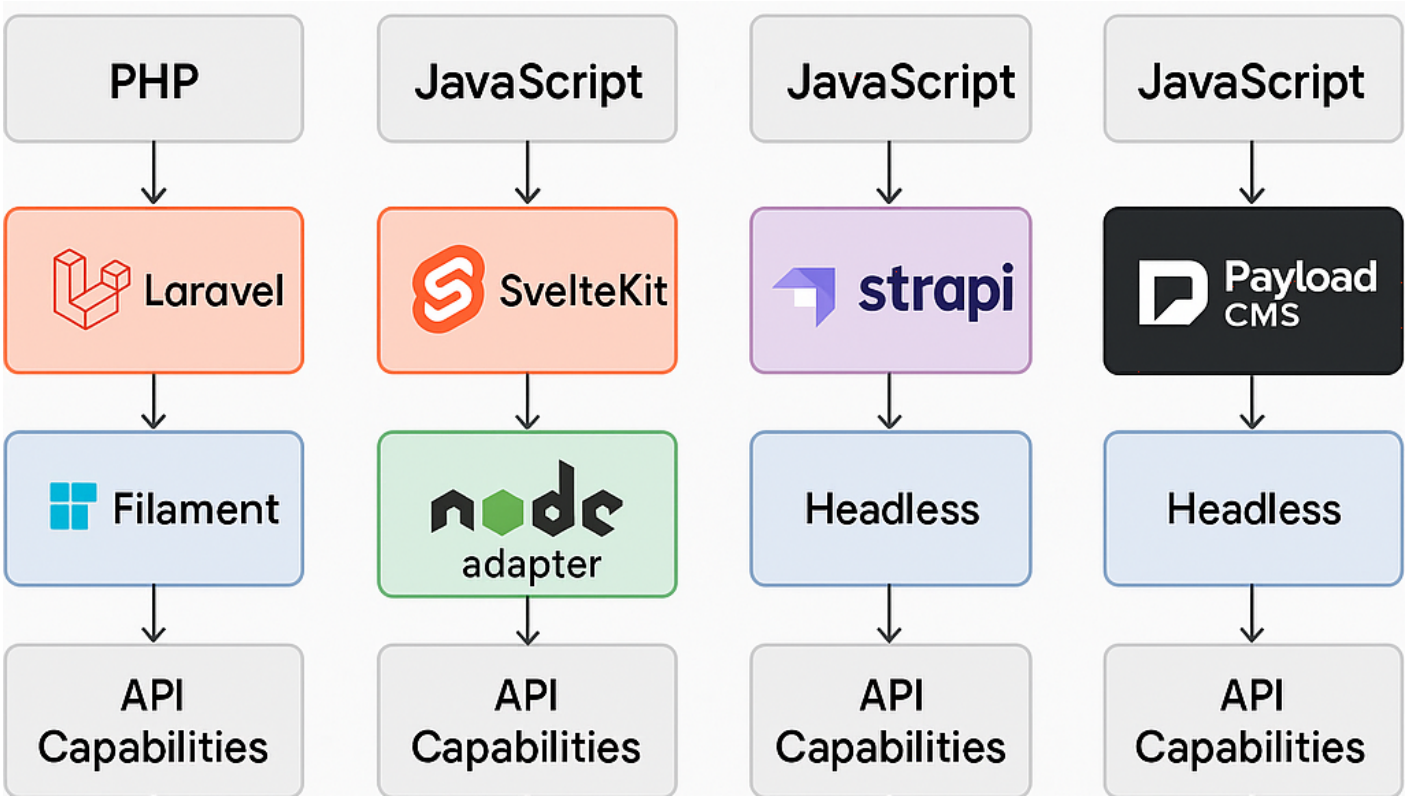
Mein Weg zu PayloadCMS: Backend-Vergleich aus der Praxis

In meinem Projekt stand ich vor der Aufgabe, ein modernes CMS aufzusetzen – mit möglichst viel Flexibilität, einem angenehmen Workflow und gutem API-Zugriff. Hier dokumentiere ich meine Evaluierung verschiedener Backend-Technologien und Frameworks – und wie ich schließlich bei **PayloadCMS** gelandet bin.



Laravel (PHP) – Stark, aber überdimensioniert und etwas altbacken?

Was mir gefallen hat: Laravel ist mächtig. Besonders das Eloquent ORM und das Ecosystem (Jobs, Queues, Policies etc.) bieten viele Features direkt „out of the box“. Mit **Filament** steht zudem ein sehr starkes Admin-Panel-Toolkit zur Verfügung.

Aber:

- Für ein CMS war mir das Setup oft zu schwergewichtig.
- PHP ist für viele Dinge okay – aber die Arbeit mit JSON, REST oder modernen Frontend-Workflows wirkt oft sperriger als bei node-basierten Lösungen.
- Auch das Hosting (z. B. auf Vercel oder Netlify) ist nicht ganz so smooth wie mit JavaScript-Stacks.

Fazit: Ein tolles Framework, aber für mein Ziel „Headless CMS mit modernem JS-Frontend“ war es nicht die erste Wahl.

SvelteKit – Elegantes Frontend, aber keine komplexen Backend-Lösungen

Was ich mochte: SvelteKit ist superschnell, modern und macht Spaß. Besonders die neue V5 ist sehr spannend für reaktive Frontends. Die Integration mit dem Node-Adapter ist einfach – und fürs **Frontend** sehe ich darin meine langfristige Lösung.

Was mir fehlte:

- Komplexere Backend-Funktionalität wie Rollen, dynamische Models, Auth oder Admin-Panels fehlen oder müssen selbst gebaut werden.
- Es gibt zwar ein paar Tools (z. B. Lucid, Prisma), aber der Ökosystem-Vergleich mit React zeigt: weniger Auswahl.
- Kein echtes CMS-Feeling.

Fazit: Perfekt für Frontends. Als CMS-Backend nicht geeignet.

Strapi – Gute API, aber UI nicht meins

Pluspunkte:

- Sehr reifes Headless CMS auf Basis von Node.js.
- Gute REST- und GraphQL-Schnittstellen.
- Rollen-/Rechtemanagement, dynamische Content-Types, einfache Auth.

Aber:

- Das Admin-Panel wirkt auf mich visuell und UX-technisch nicht ansprechend.
- Die Konfiguration ist teilweise uneinheitlich (Code vs. UI).
- Etwas schwergewichtig für kleinere Projekte.

Fazit: Technisch solide – aber ich habe mich im Admin nicht wohlgefühlt.

PayloadCMS

Warum ich geblieben bin:

- Komplett in TypeScript.
- Das Admin-Panel ist schnell, intuitiv und sehr anpassbar.
- Content-Modeling über Code (statt Click UI).
- Lokale Entwicklung, gute Dokumentation, offene Architektur.
- Built-in Auth, Access Control, Dateiupload, Versioning etc.
- Ideal für Entwickler:innen.

Fazit: PayloadCMS ist genau das, was ich gesucht habe: Entwicklerfreundlich, modern, headless und API-zentriert. Es passt gut zu meinem Setup mit SvelteKit als Frontend – und ist damit mein Favorit für das Projekt.

Beispiel-Systemarchitektur

```
|     Backend (PayloadCMS)    |
|-----------------------------|
|  - Payload Server (Next.js)  |
|  - Access Control & Auth     |
|  - Custom Hooks / Logic      |
|  - ORM / Content Models      |
|  - File Uploads / Admin UI   |
+-------------+---------------+
              |
              v
+-----------------------------+
|         MongoDB             |
|-----------------------------|
|  - Persistente Speicherung   |
|    von Inhalten & Usern      |
+-----------------------------+
```

Weiterführende Ressourcen

- PayloadCMS Docs
- SvelteKit V5 Docs
- Filament für Laravel

Revision #6
Created 23 April 2025 09:36:22 by Marius Klein
Updated 23 April 2025 10:03:26 by Marius Klein