LU08a - Warum braucht es Frontend-Frameworks?

Das Problem: Vanilla JS in grossen Projekten

Für kleine Projekte wie einen Toggle-Button oder ein einfaches Accordion funktioniert Vanilla JavaScript problemlos. Sie schreiben etwas Code, er läuft – fertig.

Aber mit wachsenden Anwendungen zeigt Vanilla JS seine Schwächen. Das haben Sie selbst beim FAQ-Accordion-Projekt erlebt:

```
// Sie mussten manuell:
// 1. Alle Buttons finden
const buttons = document.querySelectorAll('.accordion-btn');

// 2. Über alle iterieren
buttons.forEach(button => {

  // 3. Auf Klicks hören
  button.addEventListener('click', (e) => {
    const btn = e.currentTarget;

    // 4. Zum richtigen Geschwister-Element navigieren
    const panel = btn.nextElementSibling;
    const isOpen = panel.classList.contains('open');

    // 5. Zuerst alle anderen schliessen
    buttons.forEach(other => {
      other.nextElementSibling.classList.remove('open');
      other.setAttribute('aria-expanded', 'false');
    });

    // 6. Dann das richtige öffnen
    if (!isOpen) {
      panel.classList.add('open');
      btn.setAttribute('aria-expanded', 'true');
    }
  });
});
```

Das ist **imperativer Code** – Sie sagen dem Browser Schritt für Schritt, *wie* er etwas tun soll. Es funktioniert. Aber stellen Sie sich vor, das für eine ganze Applikation mit Dutzenden von Komponenten zu machen.

Drei Probleme treten schnell auf:

DOM-Abhängigkeit: Ihr JavaScript ist eng mit Ihrer HTML-Struktur verknüpft. Ändern Sie ein `<div>` im HTML, bricht `nextElementSibling` lautlos.

Unsauberer Zustand: Wir lesen den aktuellen Zustand *aus dem DOM* (`classList.contains('open')`). Der DOM wird zur Datenbank – wofür er nie gedacht war.

Die riesige Datei: Alles landet in einer einzigen `script.js`. Etwas zu finden, zu ändern oder wiederzuverwenden wird mühsam. Entwickler nennen das **Spaghetti-Code**.

Die Lösung: Eine andere Denkweise

Frameworks wie Vue.js, React oder Angular wechseln den Ansatz: von imperativ zu **deklarativ**.

Imperativ (Vanilla JS)	Deklarativ (Framework)
Wie? – jeden Schritt beschreiben	*Was?* – das Endresultat beschreiben
«Button finden, Klick abwarten, Geschwister traversieren, Klasse togglen»	«Wenn `isOpen` wahr ist, zeige das Panel»
Sie verwalten das DOM	Das Framework verwaltet das DOM

Dieselbe Accordion-Logik in Vue:

```
<button @click="isOpen = !isOpen">Frage</button>
<div :class="{ open: isOpen }">Antwort</div>
```

Zwei Zeilen. Kein `querySelector`. Kein manuelles Klassen-Togglen. Wir ändern die Daten (isOpen) – und die Oberfläche aktualisiert sich automatisch. Das nennt sich **Reaktivität**.

Drei zentrale Vorteile

1. Komponentenbasierte Architektur

Ein Framework erlaubt es, eine Benutzeroberfläche aus eigenständigen, wiederverwendbaren Bausteinen – sogenannten **Komponenten** – aufzubauen. Jede Komponente enthält ihr eigenes HTML, ihre Logik und ihre Styles gebündelt.

```
App
├── Navbar
```

```
├── AccordionItem  ← einmal bauen, überall einsetzen
├── AccordionItem
└── AccordionItem
```

Anstatt HTML zu kopieren und JS jedes Mal manuell zu verdrahten, schreiben wir `<AccordionItem />` – und es funktioniert einfach. Wie ein eigener HTML-Tag, den wir selbst erfunden haben.

2. Automatische Datenbindung

In Vanilla JS müssen wir bei einer Datenänderung das DOM manuell aktualisieren. In einem Framework sind Daten und Oberfläche *miteinander verbunden*. Ändern sich die Daten, folgt die Oberfläche automatisch – kein manuelles DOM-Update nötig.

```
// Vanilla JS: Sie erledigen die Arbeit selbst
const panel = document.querySelector('.panel');
panel.classList.toggle('open'); // manuell

// Vue: Das Framework erledigt die Arbeit
isOpen.value = true; // nur die Daten ändern – der Rest
passiert automatisch
```

3. Single Source of Truth

Der Zustand (State) lebt an einem einzigen Ort – in einer JavaScript-Variable – und nicht verteilt über Dutzende von DOM-Elementen. Wenn Sie wissen wollen, ob das Accordion offen ist, prüfen Sie `isOpen` – nicht `element.classList.contains('open')`.

Das macht Ihre Applikation deutlich einfacher zu verstehen, zu debuggen und zu erweitern.

Wann benutze ich was?

Frameworks sind nicht immer das richtige Werkzeug. Eine grobe Orientierung:

Situation	Ansatz
Kleines Skript, einmaliger Toggle	Vanilla JS genügt
Mehrere interaktive Komponenten	Framework sinnvoll
Grosse App mit gemeinsamem State	Framework unerlässlich

Wichtig ist, dass wir zuerst Vanilla JS gelernt haben – wie Sie es mit der Alarado-Landingpage und dem FAQ-Accordion getan haben. Sie verstehen nun, *was* ein Framework für Sie übernimmt. Sie haben die Probleme kennengelernt (Elemente abrufen, Zustand aus dem HTML abrufen etc.), die

Frameworks lösen.