

Aktuelle Themen der Web-Entwicklung

Software- und Web-Engineering I
Informatik – 3. Semester



Struktur, Gestaltung, Verhalten, Typisierung, Abhängigkeitsinjektion, REST und ein Back-End-Framework. Damit habt ihr **einen kompletten Stack von Hand gebaut**.

Heute geht es um das, was in der Industrie darauf aufsetzt, und um die Frage, die euch seit dem ersten Front-End-Code im Projekt begleitet: warum eigentlich kein React?

1 Frontend-Frameworks

3 KI-gestützte Entwicklung

5 Wie geht es weiter?

2 SSR, SSG & Hydration

4 Web-Security-Basics

Vier Kurzthemen. Nichts davon ist klausurrelevant.

Was in eurem Front-End mühsam war:

- Nach jeder Datenänderung von Hand die richtigen DOM-Knoten suchen und aktualisieren
- Der Zustand lag verteilt: teils in Variablen, teils im DOM, teils in `data-`-Attributen
- Eine vergessene Stelle, und die Anzeige passte nicht mehr zu den Daten
- Dieselbe Kartenkomponente an drei Stellen leicht unterschiedlich gebaut

Das war keine Ungeschicklichkeit, sondern ein bekanntes Problem. Genau dafür gibt es Frontend-Frameworks.

Imperativ (was ihr gemacht habt)

Ihr beschreibt **die Schritte**: Element suchen, Text setzen, Klasse umschalten, Kind anhängen.

Ihr seid dafür verantwortlich, dass der DOM den Daten folgt.

Deklarativ (Framework)

Ihr beschreibt **das Ergebnis**: so sieht die Oberfläche für diesen Zustand aus.

Das Framework berechnet die nötigen DOM-Änderungen selbst.

Die Formel lautet $\text{view} = f(\text{state})$. Ändert sich der Zustand, wird die Ansicht neu abgeleitet, nicht von Hand nachgezogen.

Vanilla JS (bekannt aus web-05)

```
function render(items) {  
  const ul = document.querySelector("#list");  
  ul.innerHTML = "";  
  for (const it of items) {  
    const li = document.createElement("li");  
    li.textContent = it.title;  
    ul.appendChild(li);  
  }  
}
```

React (nur lesen, nicht lernen)

```
function List({ items }) {  
  return (  
    <ul>  
      {items.map(it =>  
        <li key={it.id}>{it.title}</li>  
      )}  
    </ul>  
  );  
}
```

Links steht, **wie** die Liste entsteht. Rechts steht, **wie sie aussieht**. Den Rest erledigt das Framework.

Komponenten

Wiederverwendbare Bausteine mit eigener Logik und eigenem Aussehen.

Zustandsverwaltung

Ein definierter Ort für den Zustand statt verteilter Variablen.

Routing

Mehrere Ansichten ohne vollständigen Seitenwechsel.

Ökosystem

Build-Werkzeuge,
Testbibliotheken,
Komponentensammlungen.

Nichts davon ist magisch. Alles davon könntet ihr selbst bauen, und in eurem Projekt habt ihr Teile davon selbst gebaut.

React

Größte Verbreitung, Bibliothek statt Framework, sehr viele Zusatzentscheidungen.

Vue

Sanfter Einstieg, Template-Syntax nah an HTML.

Angular

Vollständiges Framework, TypeScript und **Dependency Injection** von Haus aus.

Svelte

Verlagert die Arbeit in den Compiler, wenig Laufzeit-Overhead.

Angular ist der direkte Anschluss an web-09: dieselbe DI-Idee, die ihr bei Spring gesehen habt, im Front-End.

Die Regel war didaktisch, nicht ideologisch:

- Wer `querySelector` nie benutzt hat, versteht nicht, was React ihm abnimmt
- Wer nie eine Fetch-Antwort von Hand ins DOM gebracht hat, debuggt später blind
- Frameworks wechseln, das darunterliegende **Plattformwissen** bleibt

Reihenfolge

Erst das Fundament, dann die Abstraktion. Eine Abstraktion, deren Problem man nie hatte, ist nur zusätzliche Komplexität.

Ab jetzt dürft ihr. In SWE II und in euren eigenen Projekten ist ein Framework die normale Wahl.

Web Components

(web-06)

- Standard im Browser, keine Abhängigkeit
- Custom Elements, Shadow DOM, Slots
- Framework-übergreifend einsetzbar

Aber

- Kein deklaratives Rendering out of the box
- Zustandsverwaltung müsst ihr selbst lösen
- Weniger Werkzeuge und Beispiele

Für ein Design-System oder eine Handvoll wiederverwendbarer Bausteine sind Web Components oft die ruhigere Wahl. Für eine ganze Anwendung greift man meist zum Framework.

1 Frontend-Frameworks

3 KI-gestützte Entwicklung

5 Wie geht es weiter?

2 **SSR, SSG & Hydration**

4 Web-Security-Basics

Wo das HTML herkommt, das im Browser ankommt.

CSR

Client-Side Rendering
Server schickt ein fast leeres
HTML plus JavaScript. Der
Browser baut die Seite.

SSR

Server-Side Rendering
Server baut das fertige HTML
pro Anfrage und schickt es.

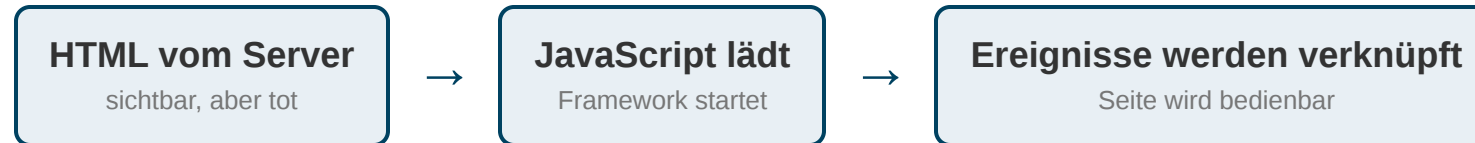
SSG

Static Site Generation
HTML wird **beim Build** erzeugt
und liegt als Datei bereit.

Das kennt ihr schon

Thymeleaf (web-10) und
Handlebars (web-12) sind SSR.

	Erste Anzeige	Gut geeignet für
CSR	spät: erst nach dem JavaScript	Anwendungen hinter dem Login, Dashboards
SSR	früh: HTML kommt fertig an gut für Suchmaschinen	Inhalte, die pro Nutzer:in unterschiedlich sind
SSG	sofort: nur eine Datei ausliefern	Blogs, Dokumentation, Marketing-Seiten



Hydration

Das Framework übernimmt im Browser das bereits vorhandene, serverseitig erzeugte HTML und macht es interaktiv, statt es neu aufzubauen.

Die Lücke dazwischen ist spürbar: die Seite ist sichtbar, aber Klicks passieren noch nichts. Ein häufig unterschätztes Problem.

- **Next.js** (React), **Nuxt** (Vue), **SvelteKit** (Svelte), **Analog** (Angular)
- Pro Seite entscheidbar: statisch erzeugen, pro Anfrage rendern oder erst im Browser
- Routing, Datenabruf und Build gehören zum Paket

Der Trend geht nicht zu „alles auf dem Server“ oder „alles im Browser“, sondern zur **Wahl pro Seite**.

Largest Contentful Paint

Wann ist der größte sichtbare Inhalt da?

Interaction to Next Paint

Wie schnell reagiert die Seite auf Eingaben?

Cumulative Layout Shift

Springt das Layout beim Laden?

Diese Werte sind messbar, vergleichbar und beeinflussen die Sichtbarkeit in Suchmaschinen. Deshalb ist die Frage „wo entsteht das HTML“ eine Geschäftsfrage, keine Geschmacksfrage.

- 1 Frontend-Frameworks
- 2 SSR, SSG & Hydration
- 3 KI-gestützte Entwicklung**
- 4 Web-Security-Basics
- 5 Wie geht es weiter?

Ehrlich über das, was die meisten von euch längst benutzen.

Gut

- Wiederkehrendes Gerüst: DTOs, Konfiguration, Boilerplate
- Testfälle zu vorhandenem Code
- Fremden Code erklären lassen
- Fehlermeldungen einordnen
- Erste Fassung, die man selbst überarbeitet

Schwach

- Die richtigen **Anforderungen** finden
- Architekturentscheidungen mit Folgen
- Konsistenz über viele Dateien hinweg
- Alles, was Domänenwissen aus eurem Projekt braucht
- Erfundene Bibliotheken und Methoden, überzeugend formuliert

Ein Assistent kann euch Code schreiben. Er kann nicht wissen, **was** eure Nutzer:innen brauchen.

- Anforderungen und Anwendungsszenarien (**P2**) entstehen aus Gesprächen, nicht aus Prompts
- Die MVP-Auswahl (**P4**) ist eine Entscheidung über Wert und Aufwand in eurem Kontext
- Die Architektur muss zu eurem Team passen, nicht zum Durchschnitt des Trainingsdatensatzes

Die Anforderungsarbeit bleibt eure Arbeit. Genau sie ist der Teil des Berufs, der nicht wegautomatisiert wird.

- Ihr gebt Code ab, den ihr **erklären können müsst**: im Praktikum wird genau danach gefragt
- Generierter Code ist ungeprüfter Code, bis ihn jemand gelesen und getestet hat
- Lizenz- und Datenschutzfragen: was ihr in ein Werkzeug eingibt, verlässt euren Rechner
- Prüfungsrechtlich gelten die Regeln des Kurses und der Hochschule, nicht die Gewohnheit

Die Präsentationskriterien in P4 prüfen Hintergrundwissen. Wer den eigenen Code nicht erklären kann, verliert dort Punkte, unabhängig davon, wie er entstanden ist.

In die Runde gefragt, ehrlich und ohne Bewertung:

- Wofür habt ihr im Projekt Assistenten benutzt?
- Wo hat es euch wirklich Zeit gespart?
- Wo habt ihr am Ende mehr Zeit mit Nachbessern verbracht als gespart?

5 Min

- 1 Frontend-Frameworks
- 2 SSR, SSG & Hydration
- 3 KI-gestützte Entwicklung
- 4 **Web-Security-Basics**
- 5 Wie geht es weiter?

Drei Angriffe, die euer Projekt heute betreffen.

Der Angriff: Fremder Text landet als **Code** in eurer Seite.

```
// Kommentar aus einem Formular  
el.innerHTML = kommentar;
```

Enthält `kommentar` ein `<script>` oder ein `<img onerror="...">`, führt der Browser es aus.

Das betrifft euer Projekt direkt: `innerHTML` mit Daten aus einem Eingabefeld ist der häufigste XSS-Weg.

Die Gegenmaßnahme:

```
el.textContent = kommentar;
```

- `textContent` schreibt **Text**, niemals Markup
- Frameworks maskieren beim Rendern automatisch
- Wer die Maskierung umgeht, ist wieder angreifbar

Der Angriff: Eingaben werden Teil der Abfrage.

```
"SELECT * FROM users WHERE name = '"  
  + name + "'"
```

Mit `name = ' OR '1'='1` liefert die Abfrage alle Datensätze.

Die Gegenmaßnahme: Werte **binden** statt zusammenkleben.

- JPA und TypeORM erzeugen parametrisierte Abfragen
- Repository-Methoden und Query-Parameter sind bereits sicher
- Gefährlich wird es erst bei selbst gebautem SQL

Wer Spring Data oder TypeORM so benutzt, wie es in web-10 und web-12 gezeigt wurde, ist hier bereits geschützt.

Typische Fehler

- Nur im Front-End validieren
- Passwörter im Klartext speichern
- Autorisierung nur über eine ausgeblendete Schaltfläche
- Geheimnisse im Repository

Im Kurs-Stack

- Validation Pipes und DTO-Validierung (web-12)
- Spring Security mit gehashten Passwörtern (web-10)
- Prüfung **im Back-End**, bei jeder Anfrage
- Konfiguration über Umgebungsvariablen

Front-End-Validierung ist Bedienkomfort, keine Sicherheit. Wer die Anfrage direkt schickt, umgeht sie vollständig.

Frameworks schützen euch nur so lange, wie ihr sie nicht umgeht.

Die drei typischen Umgehungen:

- `innerHTML` statt `textContent`, weil Markup gebraucht wird
- Selbst gebautes SQL statt Repository-Methode, weil die Abfrage komplizierter war
- Ein Endpunkt ohne Berechtigungsprüfung, weil er „nur intern“ benutzt wird

Wenn ihr eine Abkürzung nehmt, schreibt einen Kommentar dazu, warum sie sicher ist. Wenn euch keiner einfällt, ist sie es nicht.

- 1 Frontend-Frameworks
- 2 SSR, SSG & Hydration
- 3 KI-gestützte Entwicklung
- 4 Web-Security-Basics
- 5 Wie geht es weiter?**

Das Semester läuft aus, die Themen nicht.

Im Studium

- **SWE II:** baut auf diesem Stack auf
- Projekte und Praktika: hier zählt sich Plattformwissen aus
- Abschlussarbeiten mit Web-Anteil

Selbst weitermachen

- Ein kleines Framework-Projekt bauen: euer Projekt-Front-End noch einmal, diesmal mit React oder Vue
- MDN als Referenz, nicht als Tutorial-Sammlung
- Den eigenen Projektcode in einem halben Jahr noch einmal lesen

Der Unterschied zwischen „kann ein Framework bedienen“ und „versteht Web-Entwicklung“ ist genau das, was ihr dieses Semester aufgebaut habt.

Diese Woche

- **Endabgabe P4** am Sonntag, 17.01.
- Danach: Abschlussgespräche im Praktikum

Nächste Woche

- **Projekt-Pitches** in der Vorlesung
- Hinweise zur Klausuranmeldung
- Fragen zur Klausur mitbringen

Denkt an die Klausuranmeldung. Die Frist liegt vor dem Prüfungszeitraum und wird nicht verlängert.

Fragen?