Projektdokumentation Projekt Modul 248

M248 - ICT-Lösungen mit aktuellen Technologien realisieren

Projektarbeit von  
- Karl Zenker  
- Lars Marty

Inhaltsverzeichnis

[1. Projektbeschrieb 1](#_Toc197550120)

[2. Variantenentscheid / Technologienevaluation (1) 2](#_Toc197550121)

[3. Best Practices(2),(3),(4) 11](#_Toc197550122)

[4. Usability Prototype 13](#_Toc197550123)

[5. Vorgehensweise 14](#_Toc197550124)

[6. Reflexionen 15](#_Toc197550125)

[7. Git-Repository 16](#_Toc197550126)

[8. Quellenverzeichnis 17](#_Toc197550127)

# Projektbeschrieb

Die Idee des Projekts ist es, dass wir eine Trails Management App implementieren, die für die fiktive Firma ‘RappiTours’ entwickelt wird. Das Wichtigste dabei ist, dass wir dafür möglichst verbreitete und effiziente Technologien (die wir auch aufgrund einer entsprechenden Evaluation auswählen) verwenden, damit die Applikation später von einer (zum aktuellen Zeitpunkt noch unbekannten) Agentur weiterentwickelt werden kann.

Die Anforderungen an die Applikation entsprechen grossmehrheitlich denen, die in der uns vorgegebenen Anforderungsspezifikation beschrieben sind. Diese haben wir auch entsprechen in unsere Projektplanung (siehe -> ‘5. Vorgehensweise’) übernommen.

Zusätzlich zur Applikation wird die vorliegende Dokumentation gefordert, die diverse Dinge wie zum Beispiel die Technologienevaluation, ‘Best Practices’ zu den Technologien sowie eine Beschreibung der Vorgehensweise und Reflexionen beinhalten soll.

# Variantenentscheid / Technologienevaluation (1)

Im folgenden Abschnitt beschreiben wir, wie und weshalb wir uns für den entsprechenden JavaScript-Package Manager und die entsprechende Front- und Backendtechnologie entschieden haben. Wir haben uns sowohl für eine Front- als auch eine Backendtechnologie entschieden, da wir unsere Applikation gerne hybrid randern würden. Dabei sollen die statischen Elemente im Backend und die veränderbaren Elemente (wie z.B. Formulare und Trails) im Frontend gerandert werden.

Als erstes haben wir uns jedoch damit beschäftigt, welchen JavaScript-Package Manager wir verwenden wollen, wobei wir uns zwischen NPM, Yarn und Bun entscheiden mussten.

Ein Bild, das Text, Screenshot, Display, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Erklärung der einzelnen Punkte:

* **Installation / Setup**

Unter ‘**Installation / Setup**’ ist gemeint, wie einfach es ist, den entsprechenden Package Manager zu installieren und zu konfigurieren. Die Gewichtung für diesen Punkt beträgt 15%, weil eine unkomplizierte Einrichtung einen schnellen Projektstart ermöglicht und Zeit spart.

* **NPM** hat in diesem Punkt nicht so gut abgeschnitten, weil es standardmäßig installiert ist, aber bei größeren Projekten oft manuelle Anpassungen für optimale Nutzung erfordert.
* **Yarn** hat etwas besser abgeschnitten, weil es sich leicht über eine einzelne Befehlszeile installieren lässt und ein effizienteres Dependency-Handling mit yarn.lock bietet.
* **Bun** hat am besten abgeschnitten, weil es auf Performance ausgelegt ist und eine minimalistische, schnelle Installation ohne zusätzliche Konfiguration ermöglicht.
* **Geschwindigkeit**

Unter ‘**Geschwindigkeit**’ ist gemeint, wie schnell der Package Manager bei der Installation und Verwaltung von Abhängigkeiten arbeitet. Die Gewichtung beträgt 20%, weil die Geschwindigkeit direkten Einfluss auf die Produktivität und Entwicklungszeit hat.

* **NPM** hat in diesem Punkt nicht so gut abgeschnitten, weil es langsamer arbeitet, insbesondere bei der Erstinstallation großer Mengen an Dependencies.
* **Yarn** hat etwas besser abgeschnitten, weil es durch Parallelisierung und Caching die Installationszeit reduziert.
* **Bun** hat am besten abgeschnitten, weil es speziell für schnelle Installationen optimiert wurde und in Benchmarks oft als der schnellste Package Manager abschneidet.
* **Speicherverbrauch**

Unter ‘**Speicherverbrauch**’ ist gemeint, wie viel RAM und Speicherplatz der Package Manager während der Installation und Nutzung benötigt. Die Gewichtung beträgt 15%, weil ein effizienter Speicherverbrauch wichtig ist für Performance und Ressourcennutzung, insbesondere in großen Projekten.

* **NPM** hat in diesem Punkt nicht so gut abgeschnitten, weil es durch die node\_modules-Struktur oft sehr viel Speicher belegt.
* **Yarn** hat etwas besser abgeschnitten, weil es weniger Speicher verbraucht, indem es effizientere Dependency-Management-Techniken nutzt.
* **Bun** hat am besten abgeschnitten, weil es eine schlanke Architektur hat und auf reduzierten Speicherverbrauch optimiert ist.
* **Kompatibilität / Community-Support**

Unter ‘**Kompatibilität / Community-Support**’ ist gemeint, wie gut der Package Manager mit verschiedenen Tools und Systemen funktioniert und wie aktiv die Entwickler-Community ihn unterstützt. Die Gewichtung beträgt 20%, weil eine große Community schnell Lösungen für Probleme bereitstellt und breite Kompatibilität zukünftige Erweiterungen erleichtert.

* **NPM** hat am besten abgeschnitten, weil es der meistgenutzte Package Manager ist und eine sehr große Community sowie umfangreiche Dokumentation bietet.
* **Yarn** hat etwas schlechter abgeschnitten, weil es nicht ganz so weit verbreitet ist wie NPM, aber dennoch eine solide Unterstützung hat.
* **Bun** hat am schlechtesten abgeschnitten, weil es noch relativ neu ist und die Community sowie die Ökosystem-Unterstützung begrenzter sind.
* **Dokumentation / Benutzerfreundlichkeit**

Unter ‘**Dokumentation / Benutzerfreundlichkeit**’ ist gemeint, wie gut dokumentiert der Package Manager ist und wie intuitiv er zu nutzen ist. Die Gewichtung beträgt 10%, weil eine klare Dokumentation und einfache Bedienung den Einstieg erleichtern und Fehler reduzieren.

* **NPM** hat in diesem Punkt nicht so gut abgeschnitten, weil die Dokumentation zwar umfangreich ist, aber teils unübersichtlich.
* **Yarn** hat etwas besser abgeschnitten, weil es eine verständlichere Dokumentation und einen klaren Workflow für Dependency-Management bietet.
* **Bun** hat am besten abgeschnitten, weil es eine moderne und gut strukturierte Dokumentation mit einfachen Befehlen für schnelle Nutzung bietet.
* **Zusätzliche Funktionen (z.B. Skripte, Dependency Handling)**

Unter ‘**Zusätzliche Funktionen**’ ist gemeint, welche Features der Package Manager bietet, wie z. B. Skriptverwaltung, Dependency-Optimierung und Sicherheitsprüfungen. Die Gewichtung beträgt 10%, weil leistungsstarke Zusatzfunktionen Flexibilität bieten und Entwicklungsprozesse vereinfachen.

* **NPM** hat in diesem Punkt nicht so gut abgeschnitten, weil es zwar viele grundlegende Funktionen bietet, aber weniger moderne Features zur Optimierung von Dependencies hat.
* **Yarn** hat etwas besser abgeschnitten, weil es Plug’n’Play-Technologie unterstützt, die eine schnellere und konsistente Dependency-Verwaltung ermöglicht.
* **Bun** hat am besten abgeschnitten, weil es von Grund auf mit besseren Skript- und Build-Tools sowie schnellem Dependency-Handling entwickelt wurde.
* **Persönliche Erfahrung**

Unter ‘**Persönliche Erfahrung**’ ist gemeint, ob ein oder beide Teammitglieder bereits Erfahrung mit dem entsprechenden Packet Manager gemacht haben und ob diese Erfahrung positiv war.

* **NPM** hat in diesem Punkt mittelmässig abgeschnitten, da wir beide in diesem Modul damit in Kontakt gekommen sind und auch bereits einzelne Erfahrungen im Kurzpraktikum damit gemacht hatten, uns aber noch nicht wirklich damit auskennen.
* **Yarn** hat besser abgeschnitten, da wir dafür bereits auf einige Erfahrung aus dem Praktikum zurückgreifen können.
* **Bun** hat am schlechtesten abgeschnitten, da wir damit noch keine Erfahrungen gemacht haben.

Die Evaluation hat schlussendlich ergeben, dass Yarn die beste Option für unser Projekt darstellt. Dies ist im Text direkt unterhalb der Nutzwertanalyse genauer beschrieben.

Als zweites haben wir uns damit beschäftigt, welche Frontend-Technologie wir für unser Projekt verwenden wollen. Zur Auswahl standen Angular, React, Vue.js, Svelte und jQuery.

Ein Bild, das Text, Screenshot, Software, Display enthält.

KI-generierte Inhalte können fehlerhaft sein.

Erklärung der einzelnen Punkte:

* **Installation / Setup**

Unter ‘**Installation / Setup**’ versteht man, wie einfach es ist, das Framework zu installieren und mit einem neuen Projekt zu starten. Die Gewichtung beträgt 15%, da eine unkomplizierte Einrichtung den Entwicklungsprozess beschleunigt und Fehler bei der Konfiguration vermeidet.

* **Angular** hat in diesem Punkt nicht so gut abgeschnitten, weil es eine komplexere Setup-Struktur hat und mehrere Dateien manuell konfiguriert werden müssen.
* **React** hat etwas besser abgeschnitten, weil es durch create-react-app eine einfache, schnelle Installation bietet.
* **Vue.js** und **Svelte** haben am besten abgeschnitten, da sie sehr intuitive Setup-Prozesse haben, die schnell und direkt starten.
* **jQuery** erhielt eine mittlere Bewertung, da es sehr einfach eingebunden werden kann, aber in modernen Projekten oft zusätzliche Konfigurationsschritte erfordert.
* **Lernkurve / Verständlichkeit**

Unter ‘**Lernkurve / Verständlichkeit**’ versteht man, wie schnell und einfach Entwickler das Framework erlernen können. Die Gewichtung beträgt 20%, weil eine niedrige Lernkurve wichtig ist, besonders für Einsteiger oder kleinere Projekte.

* **Angular** hat in diesem Punkt nicht so gut abgeschnitten, weil es eine komplexe Architektur und steile Lernkurve mit vielen Konzepten wie Dependency Injection erfordert.
* **React** hat etwas besser abgeschnitten, weil es eine klare Dokumentation und einfache Komponentenstruktur bietet.
* **Vue.js** und **Svelte** haben am besten abgeschnitten, weil ihre API besonders intuitiv ist und sie eine besonders sanfte Einführung in reaktive Webentwicklung ermöglichen.
* **jQuery** erhielt eine mittlere Bewertung, da es leicht verständlich ist, aber für moderne Entwicklungen weniger relevant.
* **Performance / Effizienz**

Unter ‘**Performance / Effizienz**’ versteht man, wie schnell das Framework Webseiten rendert und wie gut es mit Ressourcen umgeht. Die Gewichtung beträgt 15%, da die Performance direkten Einfluss auf die Nutzererfahrung hat.

* **Angular** und **React** haben solide, aber nicht beste Bewertungen erhalten, da ihre Render-Prozesse gut optimiert, aber etwas schwergewichtiger sind.
* **Vue.js** hat gut abgeschnitten, weil es eine schlankere Architektur als Angular bietet und schnelle Reaktionszeiten ermöglicht.
* **Svelte** hat die beste Bewertung erhalten, weil es ein kompilierendes Framework ist und keine unnötige Laufzeit-Bibliothek benötigt, was die schnellste Performance bietet.
* **jQuery** hat die niedrigste Bewertung erhalten, da es keine effizienten Optimierungen für moderne Webentwicklung bietet.
* **Community / Weiterentwicklung**

Unter ‘**Community / Weiterentwicklung**’ versteht man, wie aktiv die Entwicklung des Frameworks vorangetrieben wird und wie groß die Entwicklergemeinschaft ist. Die Gewichtung beträgt 20%, weil eine starke Community regelmäßige Updates, Support und Lernressourcen ermöglicht.

* **React** hat die beste Bewertung erhalten, da es das meistgenutzte Frontend-Framework ist und eine enorme Community sowie viele Lernressourcen bietet.
* **Angular** liegt knapp dahinter, da es ebenfalls weit verbreitet ist, jedoch eine geringere Entwicklerakzeptanz als React hat.
* **Vue.js** hat gut abgeschnitten, da es aktiv weiterentwickelt wird, auch wenn die Community kleiner als bei React ist.
* **Svelte** hat etwas schlechter abgeschnitten, da es zwar innovativ ist, aber noch eine relativ kleine Community hat.
* **jQuery** hat die niedrigste Bewertung erhalten, da es heute kaum noch aktiv weiterentwickelt wird.
* **Dokumentation / Support**

Unter ‘**Dokumentation / Support**’ versteht man, wie gut die offizielle Dokumentation strukturiert ist und wie schnell man Lösungen für Probleme findet. Die Gewichtung beträgt 10%, da eine klare Dokumentation den Entwicklungsprozess erleichtert.

* **Angular** hat die beste Bewertung erhalten, da es eine sehr umfangreiche, strukturierte Dokumentation mit offiziellen Tutorials bietet.
* **React** und **Vue.js** haben solide Bewertungen erhalten, da sie leicht verständliche und gut gepflegte Dokumentationen haben.
* **Svelte** hat etwas schlechter abgeschnitten, da es weniger Lernressourcen als die großen Frameworks hat.
* **jQuery** hat die niedrigste Bewertung erhalten, da es zwar dokumentiert ist, aber wenig neue Inhalte bietet.
* **Modularität / Skalierbarkeit**

Unter ‘**Modularität / Skalierbarkeit**’ versteht man, wie flexibel das Framework für große und wachsende Projekte geeignet ist. Die Gewichtung beträgt 10%, da Skalierbarkeit langfristig über die Wartbarkeit entscheidet.

* **Angular** hat die beste Bewertung erhalten, da es eine strukturierte Architektur mit klaren Design-Prinzipien für große Projekte bietet.
* **React** hat ebenfalls eine hohe Bewertung, da seine flexible komponentenbasierte Architektur viele Skalierungsmöglichkeiten bietet.
* **Vue.js** hat gut abgeschnitten, da es leicht anpassbar und erweiterbar ist, aber weniger strukturierte Vorgaben als Angular hat.
* **Svelte** hat etwas schlechter abgeschnitten, da es für kleinere Projekte ideal ist, aber für große Enterprise-Anwendungen noch weniger etabliert ist.
* **jQuery** erhielt die niedrigste Bewertung, da es für größere Projekte nicht empfohlen wird.
* **Persönliche Erfahrung**

Unter ‘**Persönliche Erfahrung**’ ist gemeint, ob ein oder beide Teammitglieder bereits Erfahrung mit dem entsprechenden Frontend Framework gemacht haben und ob diese Erfahrung positiv war.

* **Angular** wurde hier nicht gut bewertet, da keiner von uns damit bis jetzt Erfahrungen gemacht hat.
* **React** wurde hier sehr gut bewertet, da wir da auf bereits gemachte Erfahrungen aus dem Kurzpraktikum zurückgreifen können.
* **Vue.js** wurde hier nicht gut bewertet, da keiner von uns damit bis jetzt Erfahrungen gemacht hat.
* **Svelte** wurde hier nicht gut bewertet, da keiner von uns damit bis jetzt Erfahrungen gemacht hat.
* **jQuery** wurde hier nicht gut bewertet, da keiner von uns damit bis jetzt Erfahrungen gemacht hat.

Die Evaluation hat schlussendlich ergeben, dass React die beste Option für unser Projekt darstellt. Dies ist im Text direkt unterhalb der Nutzwertanalyse genauer beschrieben.

Als letztes haben wir uns damit beschäftigt, welche Backend-Technologie wir für unser Projekt verwenden wollen. Zur Auswahl standen Express.js, Next.js, ASP.NET und Spring Boot.

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Erklärung der einzelnen Punkte:

* **Installation / Setup**

Unter ‘**Installation / Setup**’ versteht man, wie einfach es ist, das Framework zu installieren und ein Projekt aufzusetzen. Die Gewichtung beträgt 15%, da eine unkomplizierte Einrichtung den Entwicklungsprozess beschleunigt und Zeit spart.

* **Express.js Handlebars** hat eine hohe Bewertung erhalten, weil es einfach zu installieren und zu konfigurieren ist. Die Einrichtung besteht aus wenigen Befehlen und ist schnell erledigt.
* **Next.js** hat eine leicht niedrigere Bewertung, da es zwar einfach eingerichtet werden kann, aber mehr Konfiguration erfordert, wenn spezifische Rendering-Methoden genutzt werden.
* **ASP.NET Razor Pages** hat etwas schlechter abgeschnitten, da die Installation und Konfiguration komplexer ist und oft zusätzliche Schritte für die Entwicklungsumgebung notwendig sind.
* **Spring Boot Thymeleaf** erhielt die niedrigste Bewertung, weil es für Java-Entwicklung aufgesetzt werden muss, was mehr initiale Konfiguration erfordert.
* **Lernkurve / Verständlichkeit**

Unter ‘**Lernkurve / Verständlichkeit**’ versteht man, wie schnell und einfach Entwickler das Framework erlernen können. Die Gewichtung beträgt 20%, weil eine niedrige Lernkurve gerade für Einsteiger oder kleinere Projekte wichtig ist.

* **Express.js Handlebars** hat am besten abgeschnitten, da es ein minimalistisches Framework ist, das mit wenig Konzepten auskommt und leicht verständlich ist.
* **Next.js** liegt knapp dahinter, weil es eine gute Dokumentation bietet und für React-Entwickler einfach zu verstehen ist.
* **ASP.NET Razor Pages** hat eine mittlere Bewertung erhalten, da es zwar für Entwickler mit .NET-Erfahrung gut verständlich ist, aber eine steilere Lernkurve für Neueinsteiger hat.
* **Spring Boot Thymeleaf** hat die niedrigste Bewertung erhalten, da Java-Entwicklung generell komplexer ist und das Spring-Ökosystem viele zusätzliche Konzepte mit sich bringt.
* **Performance / Effizienz**

Unter ‘**Performance / Effizienz**’ versteht man, wie schnell das Framework Daten verarbeitet und Serveranfragen beantwortet. Die Gewichtung beträgt 15%, da die Performance direkten Einfluss auf die Nutzererfahrung hat.

* **Next.js** hat die beste Bewertung erhalten, da es serverseitiges Rendering optimiert und durch intelligente Caching-Mechanismen eine schnelle Verarbeitung ermöglicht.
* **ASP.NET Razor Pages** und **Express.js Handlebars** haben ebenfalls gute Bewertungen erhalten, da sie beide auf effiziente Serverarchitekturen setzen.
* **Spring Boot Thymeleaf** hat etwas schlechter abgeschnitten, da Java generell etwas mehr Ressourcen benötigt und die Laufzeitumgebung mehr Overhead erzeugt.
* **Community / Weiterentwicklung**

Unter ‘**Community / Weiterentwicklung**’ versteht man, wie aktiv die Entwicklung des Frameworks vorangetrieben wird und wie groß die Entwicklergemeinschaft ist. Die Gewichtung beträgt 20%, weil eine starke Community regelmäßige Updates, Support und Lernressourcen ermöglicht.

* **Next.js** hat die höchste Bewertung erhalten, da es ein modernes und weit verbreitetes Framework mit vielen Community-Beiträgen und einer aktiven Weiterentwicklung ist.
* **ASP.NET Razor Pages** und **Spring Boot Thymeleaf** haben ebenfalls hohe Bewertungen erhalten, da sie von großen Unternehmen unterstützt werden und regelmäßige Updates bekommen.
* **Express.js Handlebars** hat eine geringere Bewertung erhalten, da es zwar eine große Community hat, aber als Framework im Vergleich zu neueren Technologien weniger aktiv weiterentwickelt wird.
* **Dokumentation / Support**

Unter ‘**Dokumentation / Support**’ versteht man, wie gut die offizielle Dokumentation strukturiert ist und wie schnell man Lösungen für Probleme findet. Die Gewichtung beträgt 10%, da eine klare Dokumentation den Entwicklungsprozess erleichtert.

* **ASP.NET Razor Pages** hat die beste Bewertung erhalten, da Microsoft exzellente, umfassende Dokumentation bereitstellt.
* **Next.js** und **Spring Boot Thymeleaf** haben ebenfalls hohe Bewertungen erhalten, da sie eine gut strukturierte und verständliche Dokumentation haben.
* **Express.js Handlebars** hat eine etwas niedrigere Bewertung erhalten, da es zwar dokumentiert ist, aber nicht so detaillierte Lernmaterialien bietet wie andere Technologien.
* **Modularität / Skalierbarkeit**

Unter ‘**Modularität / Skalierbarkeit**’ versteht man, wie flexibel das Framework für große und wachsende Projekte geeignet ist. Die Gewichtung beträgt 10%, da Skalierbarkeit langfristig über die Wartbarkeit entscheidet.

* **ASP.NET Razor Pages** hat die höchste Bewertung erhalten, da es eine klare Architektur für große Enterprise-Anwendungen bietet.
* **Next.js** hat ebenfalls eine hohe Bewertung, da es für moderne skalierbare Web-Apps optimiert ist.
* **Spring Boot Thymeleaf** hat eine solide Bewertung erhalten, da es gute Skalierungsoptionen für Java-Services bietet.
* **Express.js Handlebars** hat die niedrigste Bewertung erhalten, da es zwar leichtgewichtig ist, aber bei größeren Projekten weniger strukturierte Skalierungsoptionen bietet.
* **Persönliche Erfahrung**

Unter ‘**Persönliche Erfahrung**’ ist gemeint, ob ein oder beide Teammitglieder bereits Erfahrung mit dem entsprechenden Backend Framework gemacht haben und ob diese Erfahrung positiv war.

* **Express.js Handlebars** hat hier die tiefste Bewertung erhalten, da wir beide noch keine Erfahrung damit gemacht hatten.
* **Next.js** hat besser abgeschnitten, da bereits Erfahrungen aus dem Kurzpraktikum da waren.
* **ASP.NET Razor Pages** hat am besten abgeschnitten, da wir beide uns damit schon gründlich im Backend-Modul 295 beschäftigt hatten.
* **Spring Boot Thymeleaf** hat etwas schlechter als ASP.NET abgeschnitten, da das Modul, aus welchem wir diese Technologie kennen, noch laufend ist und da wir beide nicht so stark überzeugt von der Java-Welt sind.

Die Evaluation hat schlussendlich ergeben, dass Next.js die beste Option für unser Projekt darstellt. Dies ist im Text direkt unterhalb der Nutzwertanalyse genauer beschrieben.

# Best Practices

**Projektstruktur:**

* Dateien nach Features oder Domains strukturieren (nicht nach Dateitypen). Beispiel: features/blog/BlogLits.tsx statt components/BlogList.tsx
* Bei Next.js Version 13+ den App-Router (app/-Verzeichnis) mit layout-, pages-, loading- und error.tsx verwenden

**React:**

Component Design:

* Funktionale Komponenten mit Hooks und **keine** Klassenkomponenten verwenden
* Custom Hooks verwenden, um gewisse Logik wiederverwenden zu können
* Validierung und Dokumentation von Props über TypeScript-Interfaces

Code Beispiel:

A computer screen with text and symbols

AI-generated content may be incorrect.

Zustand & Daten:

Local State (Component State):

* *useState*, *useReducer* oder *useRef* verwenden, um UI-bezogene States innerhalb einzelner Components zu verwalten.
* Typische Use Cases: Formularwerte, Tabs, Errors

Code Beispiel:

Man soll vermeiden, den lokalen UI-State in einem global Store zu halten, da die zu unnötiger Komlexität führt.

Global State (Shared State):

* Global State nur nurtzen, wenn mehrere Components auf dieselben Date zugreifen oder diese ändern müssen
* Gute Einsatzbereiche: Auth-Status, Benutzerdaten, Spracheinstellungen, Warenkorb

A screen shot of a computer code

AI-generated content may be incorrect.Code Beispiel:

Datenfetching:

* Daten kann man entweder clientseitig mit *useEffect*, serverseitig mit *getServerSideProps*, oder in Server Components mit *fetch()* holen.
* SWR oder React Query verwenden wenn man effizient clientseitig Daten holen, cachen und aktualisieren will

Rendering:

* Keys bei Listen korrekt verwenden, d.h. keine Indexattribute sondern eindeutige ID’s

**Next.js:**

* Statischer Content mit *generateStaticParams* oder *getStaticProps* (Pages Router) bevorzugen, wo möglich
* Tailwind CSS oder CSS Modules verwenden fürs Styling
* Keine unstrukturierten globalen Styles benutzen
* *fetch()* in Server Components bevorzugen (besser für SEO und Performance).
* *useEffect* oder SWR / React Query nur bei clientseitigem Bedarf (z.B. User interactions).
* Verwende *cache()* oder *revalidate* für ISR und Caching.

# Usability Prototype

# Vorgehensweise

# Reflexionen

# Git-Repository

# Quellenverzeichnis

1. Künstliche Intelligenz: Mit der entsprechenden Nummer markierte Stellen wurden zu Teilen mit künstlicher Intelligenz erstellt oder benutzten künstliche Intelligenz als Informationsquelle
2. React State und Lifecycle: <https://reactjs.org/docs/state-and-lifecycle.html>
3. React Rules: <https://react.dev/reference/rules>
4. Next.js Best Practices und Rules:

<https://nextjs.org/docs/14/app/building-your-application/data-fetching/patterns>