

Projektdokumentation Projekt Modul 248

M248 - ICT-Lösungen mit aktuellen Technologien realisieren

Projektarbeit von

- Karl Zenker

- Lars Marty

Inhaltsverzeichnis

1.	Projektbeschrieb.....	1
2.	Variantenentscheid / Technologienevaluation ⁽¹⁾	2
3.	Best Practices ^{(2), (3), (4)}	11
4.	Usability Prototype	14
5.	Vorgehensweise.....	17
6.	Reflexionen	22
6.1	Tag 1 (02.05.2025)	22
6.2	Tag 2 (09.05.2025)	22
6.3	Tag 3 (16.05.2025)	23
6.4	Tag 4 (23.05.2025)	24
6.5	Gesamtfazit (06.06.2025)	24
7.	Git-Repository	25
8.	Quellenverzeichnis	26

1. Projektbeschreibung

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.

2. Variantenentscheid / Technologieevaluation ⁽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 uns eine App vorgestellt haben, die die Vorteile von beiden Welten miteinander kombiniert. Da es jedoch erforderlich war, dass wir alles Client-seitig rendern, mussten wir schlussendlich in React die 'use client'-Verhaltensregel auf oberster Ebene definieren. Somit wurden die Vorteile der gemeinsamen Nutzung von Front- und Backendtechnologie zwar fast gänzlich negiert, jedoch konnten wir so immerhin mit beiden Technologien arbeiten.

Man muss an dieser Stelle aber (vor allem auch aus späterer Sicht) sagen, dass es so besser gewesen wäre, wenn man sich schlicht und einfach auf die Frontendtechnologie konzentriert hätte. Da wir nun aber beide Welten miteinander verbunden haben, haben wir auch für beide Technologien eine entsprechende Evaluation durchgeführt.

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.

	A	B	C	D	E	F	G	H	I
1		Gewichtung	NPM - Note		Yarn - Note		Bun - Note		
2	Installation / Setup	0.15	4.25	0.6375	4.5	0.675	5	0.75	
3	Geschwindigkeit	0.2	3.75	0.75	4.5	0.9	5	1	
4	Speicherverbrauch	0.15	4	0.6	4.25	0.6375	5	0.75	
5	Kompatibilität / Community-Support	0.2	5	1	4.75	0.95	3.5	0.7	
6	Dokumentation / Benutzerfreundlichkeit	0.1	4.75	0.475	4.5	0.45	4.25	0.425	
7	Zusätzliche Funktionen (z.B. Skripte, Dependency Handling)	0.1	4.5	0.45	4.5	0.45	4.25	0.425	
8	Persönliche Erfahrung	0.1	4.75	0.475	5.5	0.55	4	0.4	
9									
10	Gesamt	1		4.3875		4.6125		4.45	
11	Rang			3		1		2	
12									
13	Yarn ist die beste Wahl für unser Projekt, da es eine gute Balance zwischen Performance, stabiler Dependency-Verwaltung und einfacher Handhabung bietet. Es ist schneller als NPM, aber leichter zu erlernen als Bun, was ideal für unsere (noch) begrenzte Node.js-Erfahrung ist. Zudem sorgt Yarn für sogenannte 'deterministische Abhängigkeiten', was zukünftige Weiterentwicklungen erleichtert. Für ein kleines, effizientes und wartbares Projekt ist Yarn daher eine kluge, pragmatische Entscheidung.								
14									
15									

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.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Gewichtung	Angular - Note		React - Note		Vue.js - Note		Svelte - Note		jQuery - Note		
2	Installation / Setup	0.15	4	0.6	4.5	0.675	4.75	0.7125	4.75	0.7125	4.25	0.6375	
3	Lernkurve / Verständlichkeit	0.2	3.5	0.7	4.5	0.9	4.75	0.95	5	1	4	0.8	
4	Performance / Effizienz	0.15	4	0.6	4.5	0.675	4.5	0.675	5	0.75	3.75	0.5625	
5	Community / Weiterentwicklung	0.2	4.75	0.95	5	1	4.5	0.9	3.75	0.75	3.5	0.7	
6	Dokumentation / Support	0.1	4.75	0.475	4.5	0.45	4.5	0.45	4	0.4	4	0.4	
7	Modularität / Skalierbarkeit	0.1	5	0.5	4.75	0.475	4.5	0.45	4	0.4	3.75	0.375	
8	Persönliche Erfahrung	0.1	4	0.4	5.5	0.55	4	0.4	4	0.4	4	0.4	
9													
10	Gesamt	1		4.225		4.725		4.5375		4.4125		3.875	
11	Rang			4		1		2		3		5	
12													
13	React ist die beste Wahl für unser Projekt, da es eine hohe Verbreitung, eine einfache Lernkurve und eine aktive Entwicklergemeinschaft bietet. React ist intuitiv und flexibel, ideal für Einsteiger mit wenig Erfahrung. Die komponentenbasierte Architektur ermöglicht eine saubere Code-Struktur, die spätere Weiterentwicklung erleichtert. Zudem sorgt React für effiziente UI-Updates, was die Performance verbessert. Für ein kleines, aber zukunftsfähiges Projekt ist React daher eine kluge, nachhaltige Entscheidung.												
14													
15													

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.

	A	B	C	D	E	F	G	H	I	J	K
1		Gewichtung	Express.js - Note		Next.js - Note		ASP.NET - Note		Spring Boot - Note		
2	Installation / Setup	0.15	4.5	0.675	4.25	0.6375	4	0.6	3.75	0.5625	
3	Lernkurve / Verständlichkeit	0.2	4.75	0.95	4.5	0.9	3.75	0.75	3.5	0.7	
4	Performance / Effizienz	0.15	4.25	0.6375	4.75	0.7125	4.5	0.675	4	0.6	
5	Community / Weiterentwicklung	0.2	4.5	0.9	5	1	4.75	0.95	4.25	0.85	
6	Dokumentation / Support	0.1	4.5	0.45	4.75	0.475	5	0.5	4.25	0.425	
7	Modularität / Skalierbarkeit	0.1	4.25	0.425	4.75	0.475	5	0.5	4.5	0.45	
8	Persönliche Erfahrung	0.1	4	0.4	5	0.5	6	0.6	5	0.5	
9											
10	Gesamt	1		4.4375		4.7		4.575		4.0875	
11	Rang			3		1		2		4	
12											
13	Next.js ist die beste Wahl für unser Projekt, da es optimale Performance, einfache Handhabung und eine enge Integration mit React bietet. Es erleichtert die serverseitige Datenverarbeitung, was die Effizienz steigert und für zukünftige Erweiterungen ideal ist. Die automatische Optimierung von Code und die flexible Architektur machen es perfekt für ein kleines, aber professionell umsetzbares Projekt.										
14											
15											

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.

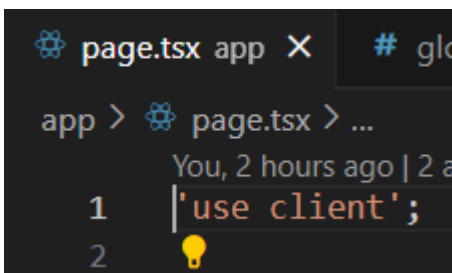
- **Express.js Handlebars** hat hier die tiefste Bewertung erhalten, da wir beide noch keine Erfahrung damit gemacht hatten.
- **Next.js** hat besser abgeschnitten, da bereits Erfahrungen aus dem Kurzpraktikum da waren.
- **ASP.NET Razor Pages** hat am besten abgeschnitten, da wir beide uns damit schon 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.

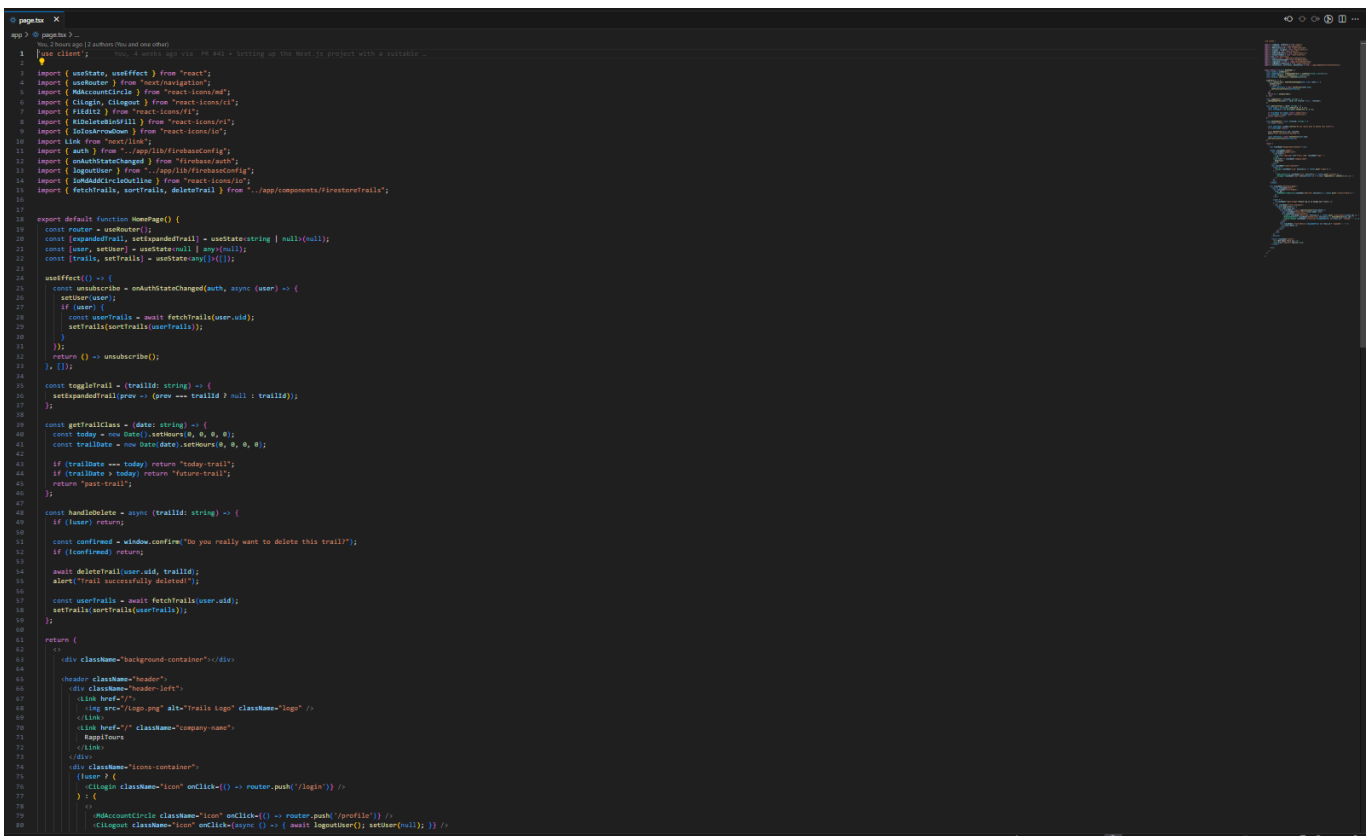
3. Best Practices (2), (3), (4)

Unter diesem Punkt ist beschrieben, wie wir unsere Technologien eingesetzt haben und was wir daraus gelernt haben.

Wie bereits unter Punkt 2 beschrieben, verwenden wir die 'use client'-Verhaltensregel auf oberster Ebene, damit die Next.js-Applikation die Seite nicht Server-seitig rendern will, sondern Client-seitig rendert.

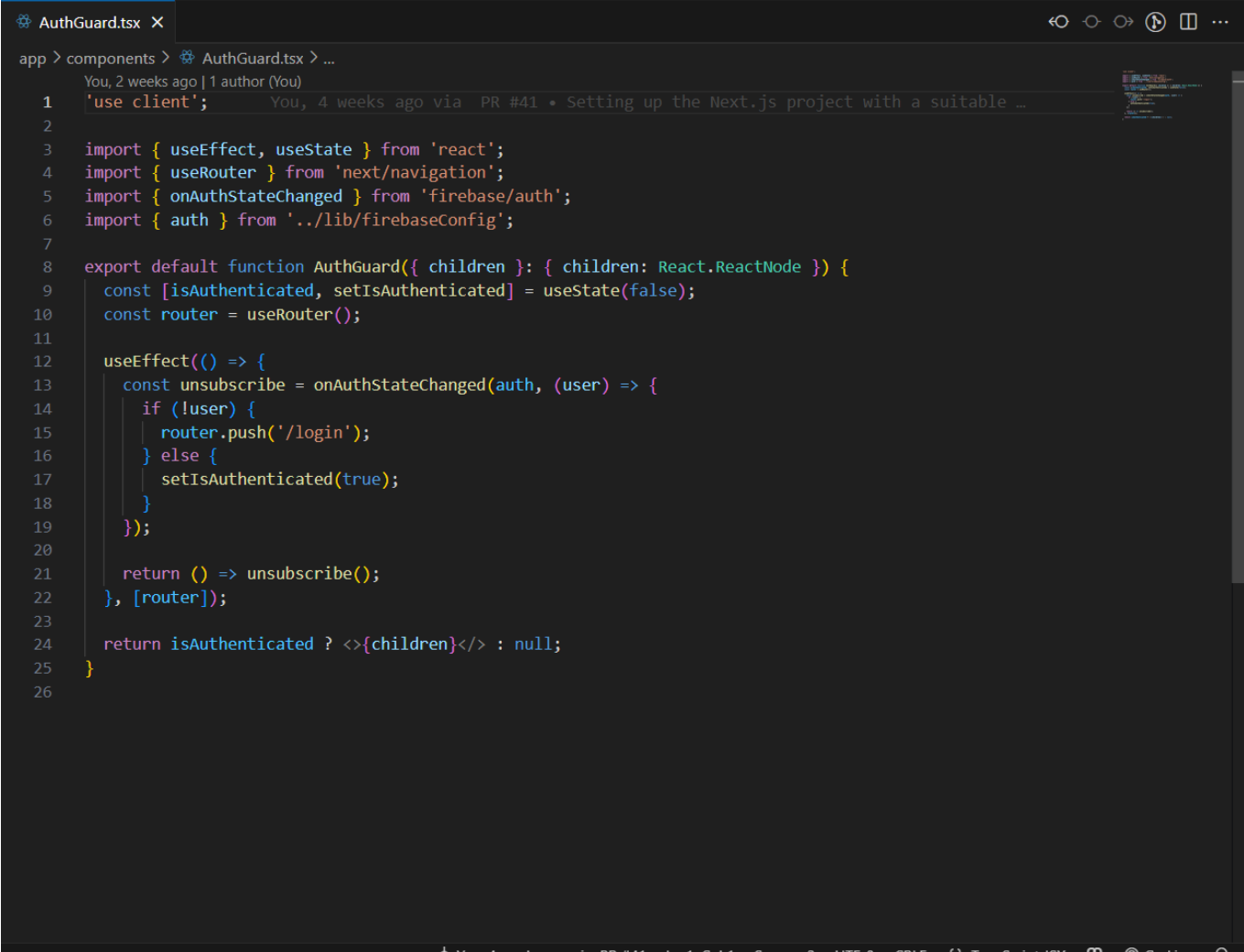


Der Rest der Files besteht danach grundsätzlich vollständig aus React Komponenten und Hooks. Ein Beispiel zu einer entsprechenden Seite wäre das folgende:



Wir haben uns dazu entschieden TypeScript anstelle von JavaScript zu verwenden, da wir so nicht auch noch auf Probleme mit den Variablentypen stossen.

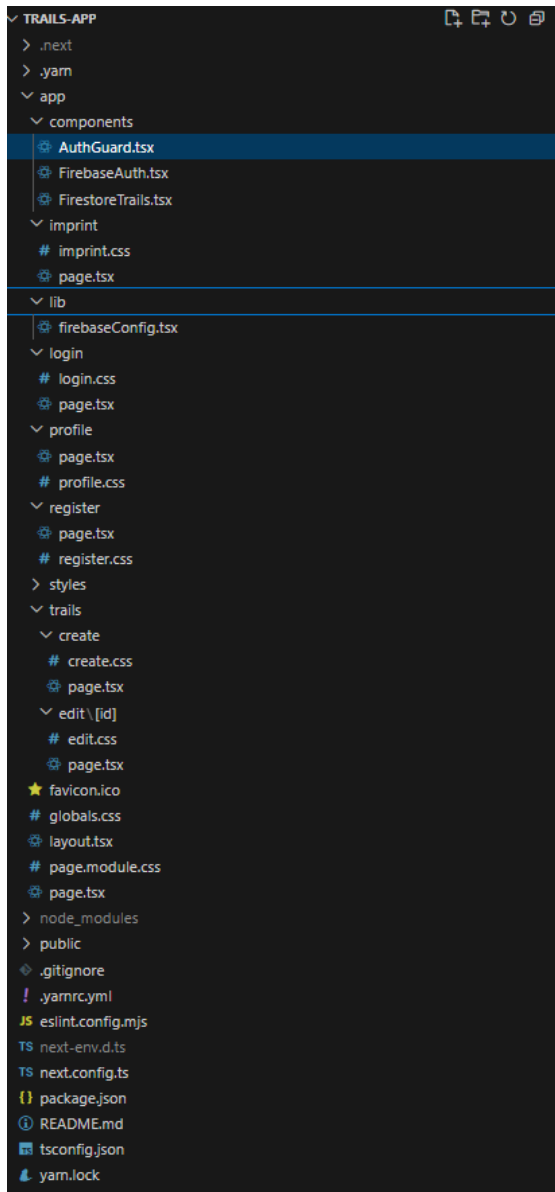
Ein Beispiel für eine separat von uns definierte Komponente wäre diese hier:



```

1  'use client';
2
3  import { useEffect, useState } from 'react';
4  import { useRouter } from 'next/navigation';
5  import { onAuthStateChanged } from 'firebase/auth';
6  import { auth } from '../lib/firebaseConfig';
7
8  export default function AuthGuard({ children }: { children: React.ReactNode }) {
9    const [isAuthenticated, setIsAuthenticated] = useState(false);
10   const router = useRouter();
11
12   useEffect(() => {
13     const unsubscribe = onAuthStateChanged(auth, (user) => {
14       if (!user) {
15         router.push('/login');
16       } else {
17         setIsAuthenticated(true);
18       }
19     });
20
21     return () => unsubscribe();
22   }, [router]);
23
24   return isAuthenticated ? <>{children}</> : null;
25 }
26
  
```

Das Routing der einzelnen Seiten funktioniert aber über den App-Router von Next. Dabei sind alle Seiten in Unterordnern des App-Ordners abgelegt und werden so automatisch von Next als Routen erkannt. Die Ordnerstruktur sieht entsprechend so aus:



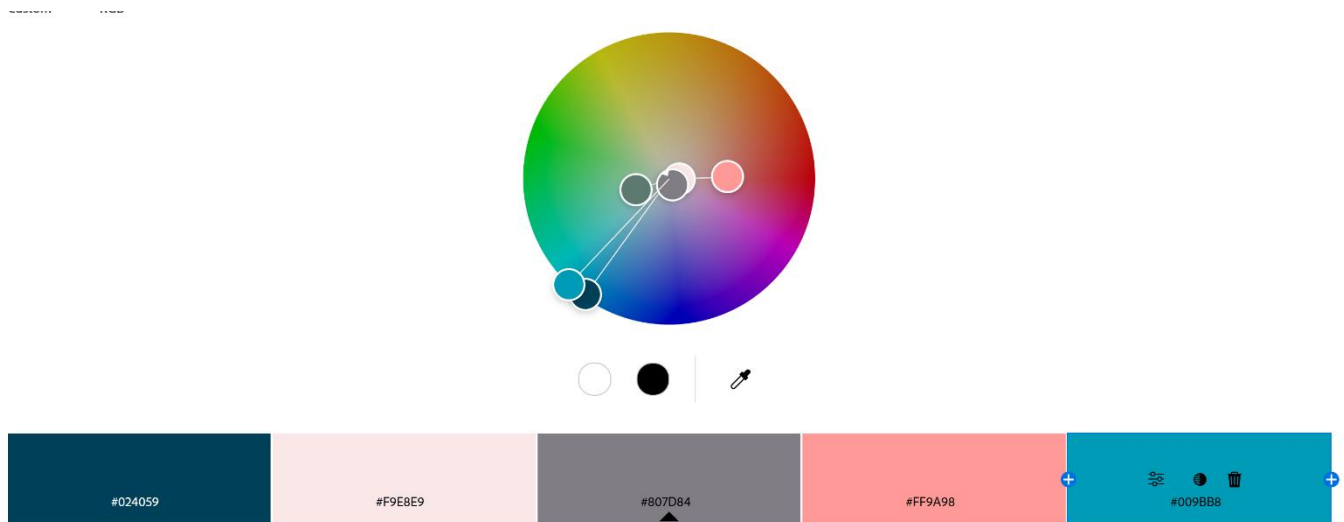
Diese Grundsätze wurden zu Beginn des Projektes so definiert und wurden über das ganze Projekt hinweg bei allen Files durchgezogen. Somit gelten diese als 'Best Practices'.

4. Usability Prototype

Alle Mockup und Wireframe Seiten sind in einem Figma Projekt unter folgendem Link zu finden:

<https://www.figma.com/design/ocBrs5HsHH2O8fXsaGd5PH/Wireframe-and-Mockup-M248?node-id=48-516&t=TvrKOa8NWUM6pORR-1>

Farbrad:



Begründung der Farbwahl:

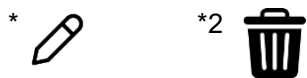
Die Farben sind praktisch und modern. Der Hintergrund ist neutral und in weiss oder hellgrau. So kann man die Inhalte besser lesen. Das Design ist nicht so wichtig. Die Hauptfarbe Blau wurde gewählt, weil sie Vertrauen und Professionalität vermittelt. Diese Farbe hebt interaktive Elemente wie Buttons oder aktive Zustände hervor, ohne aufdringlich zu wirken. Sekundärfarben wie verschiedene Graustufen strukturieren die Oberfläche, trennen Bereiche voneinander und schaffen Ordnung.

Schriftwahl:

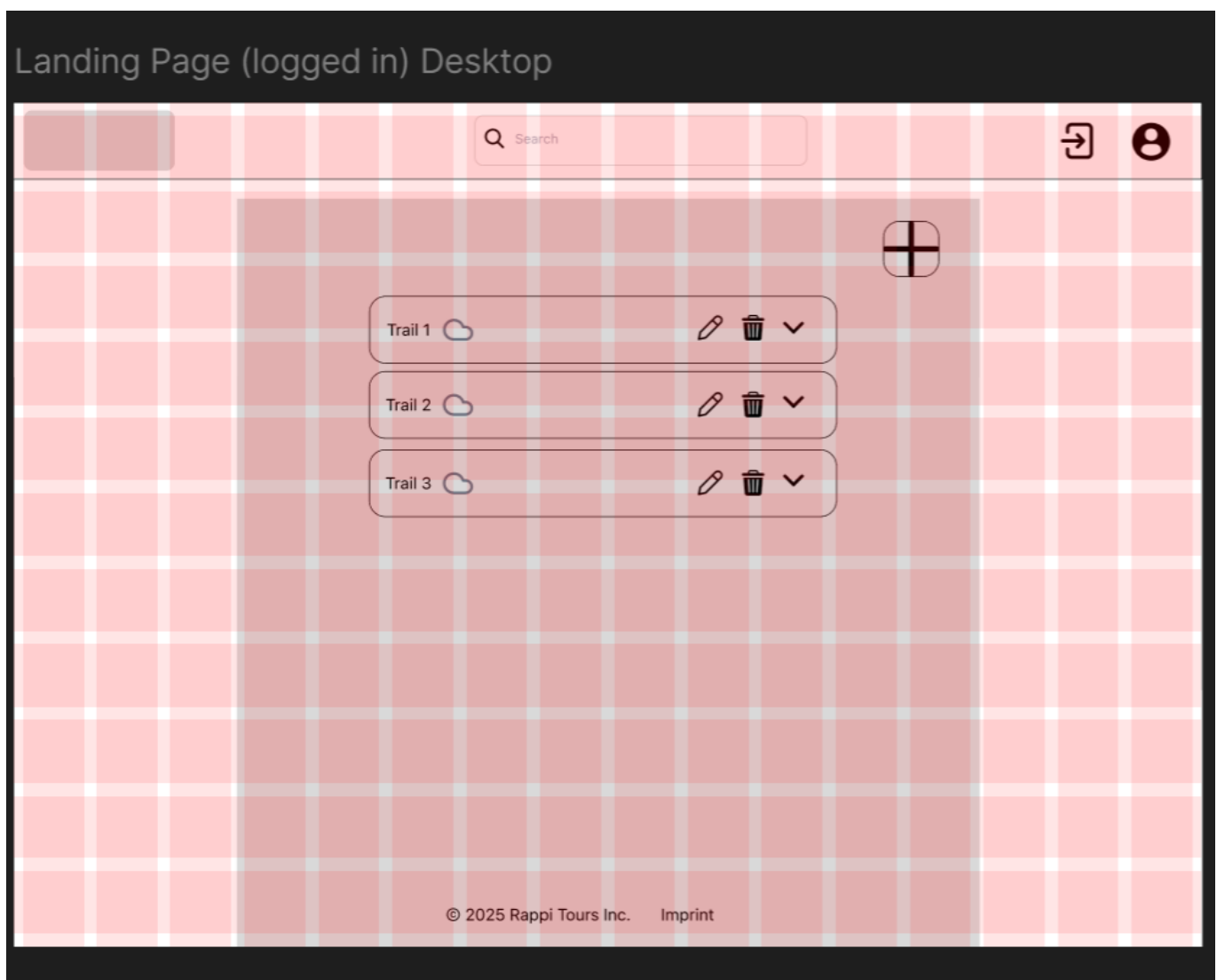
Für die Schrift wurde SF Pro verwendet, Apple's Systemschrift. Sie ist modern, praktisch und für digitale Oberflächen gemacht. SF Pro ist unauffällig, professionell und technisch und somit perfekt für Anwendungen, die Klarheit und Benutzerfreundlichkeit wichtig sind. SF Pro wird auf allen Apple-Geräten gleich angezeigt.

Iconwahl:

Die Icons wurden nach dem Prinzip der visuellen Gewichtung eingesetzt. Häufig genutzte oder besonders wichtige Funktionen verwenden gefüllte Symbole. Diese sind gut sichtbar und lenken die Aufmerksamkeit der Nutzer gezielt auf wichtige Aktionen, die etwas grundlegend ändern (z.B. der delete Button*2). Nicht ausgefüllte Icons zeigen Funktionen, die weniger genutzt werden, oder weniger gravierende Sachen machen (z.B. der edit Button*). Das sorgt für einen guten Überblick und zeigt auf einen Blick, wie wichtig die einzelnen Funktionen sind.



Screenshot der Wireframe-Desktop-Hauptpage mit den eingeteilten Gutter:

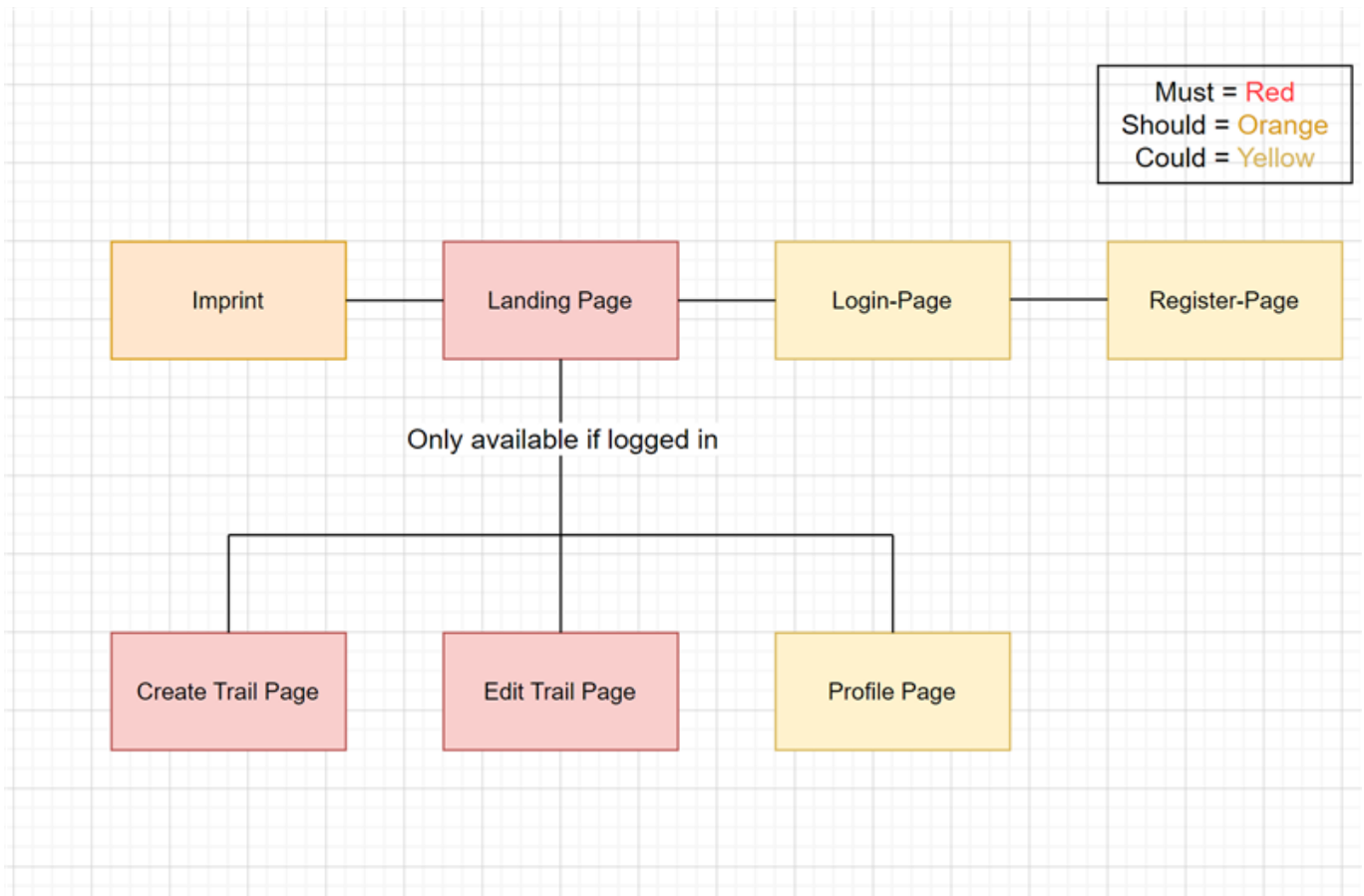


Format:

16 Columns

11 Rows

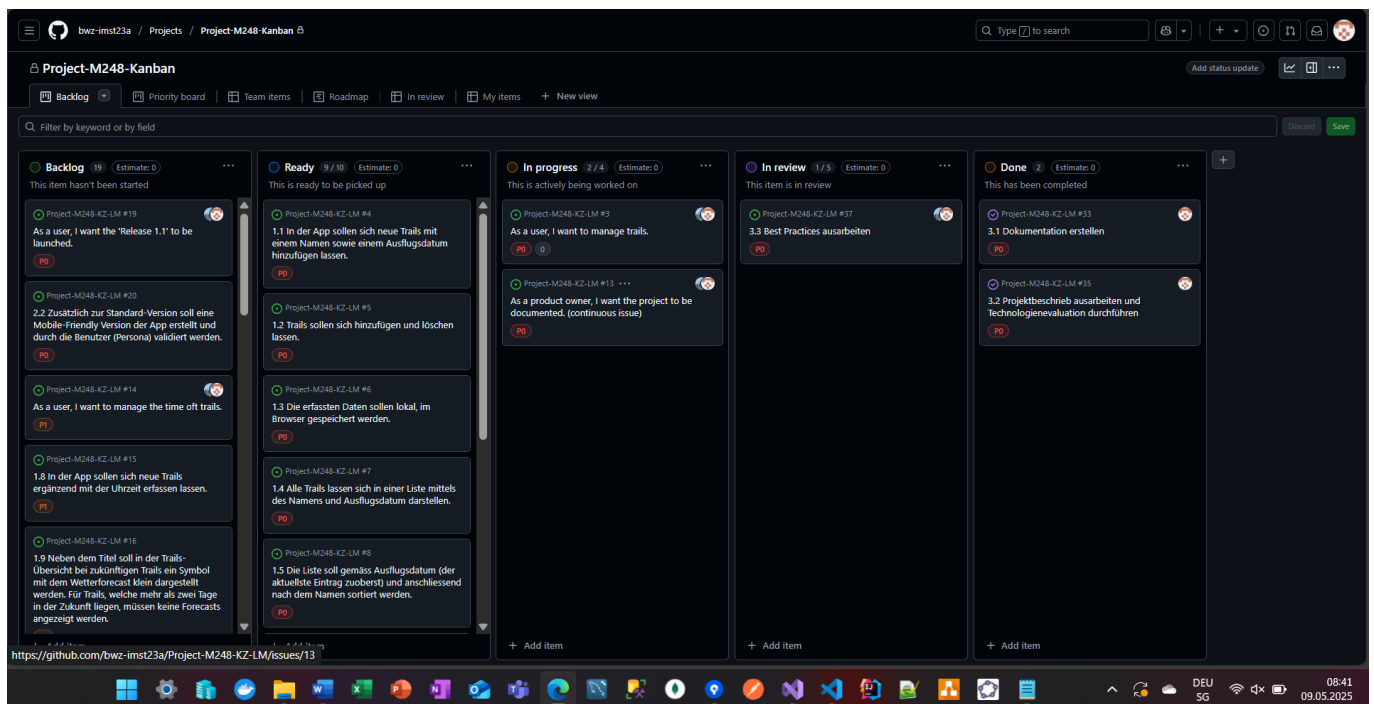
Informationsarchitektur:



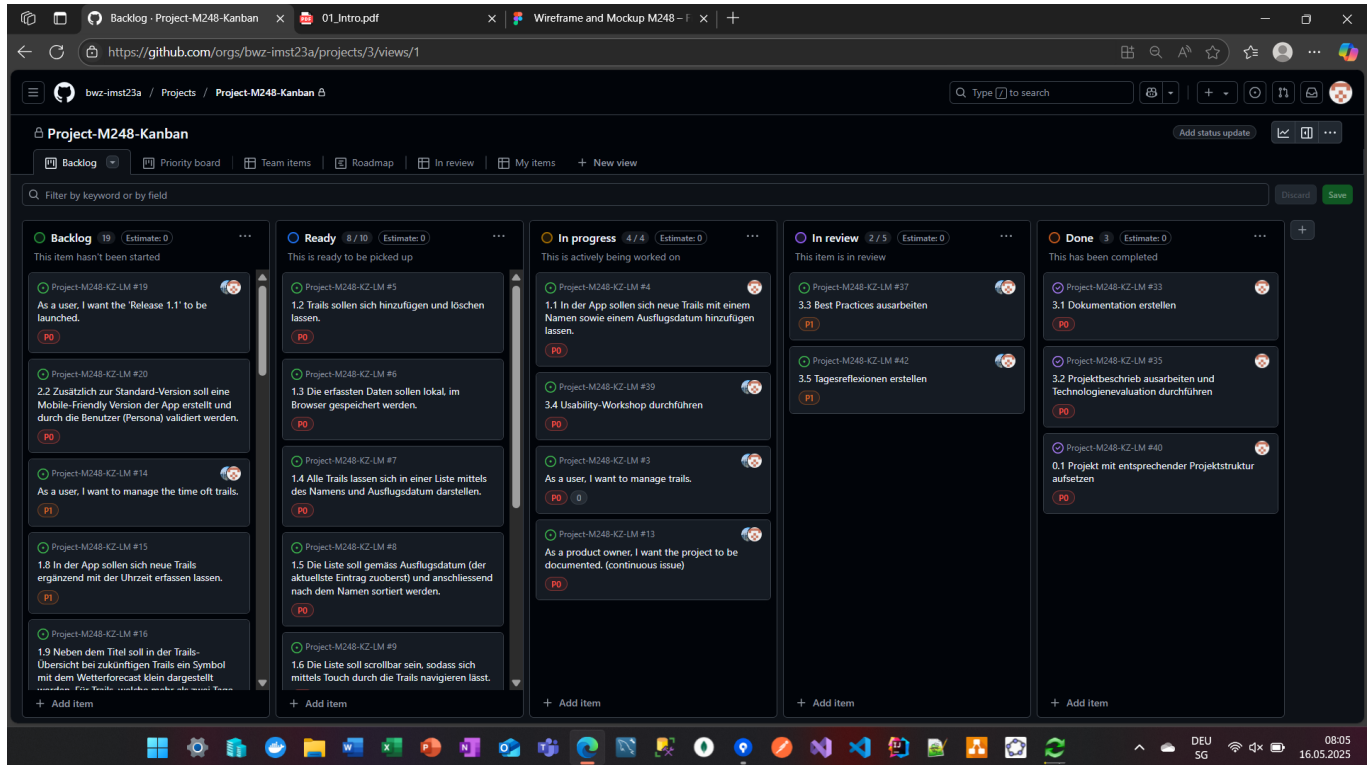
5. Vorgehensweise

Wir haben die Aufgaben im Team über ein Kanban-Board organisiert. Dieses haben wir wöchentlich geupdated:

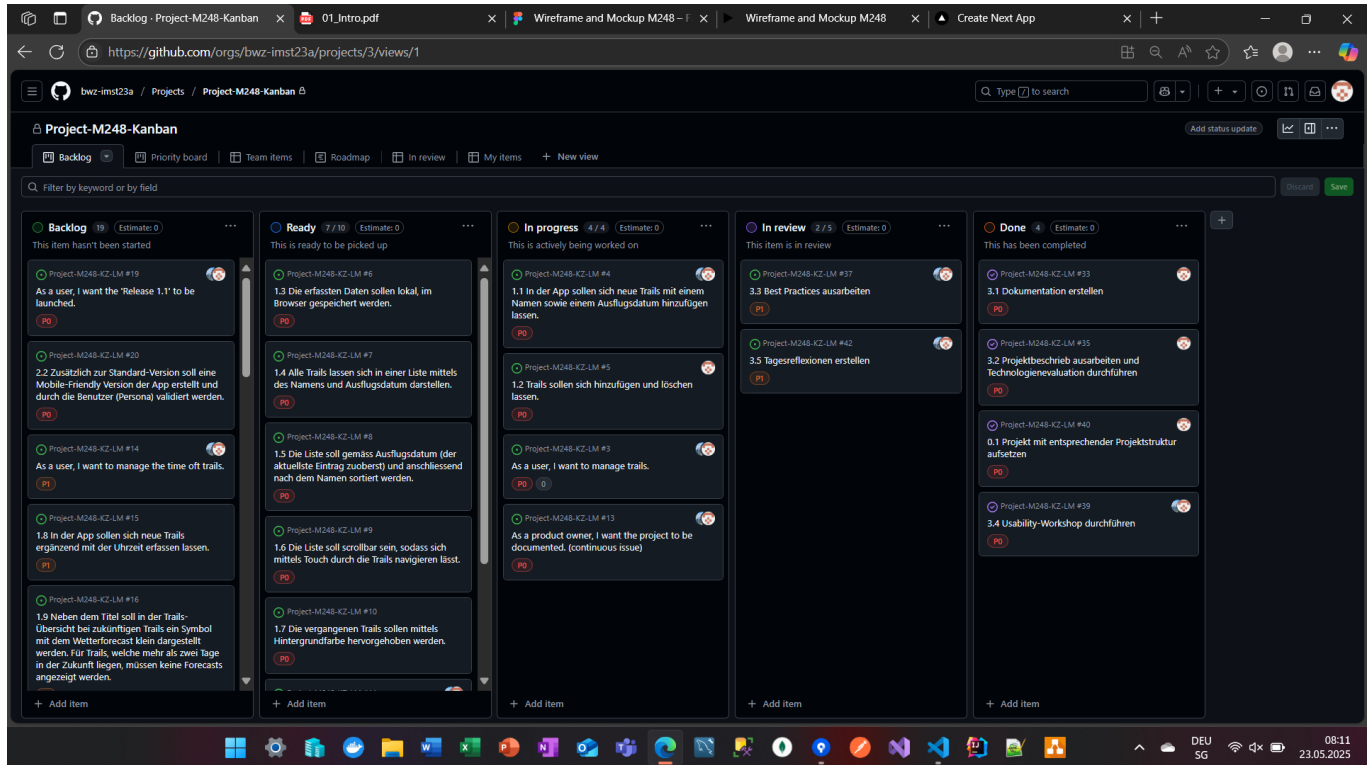
Stand nach Woche 1:



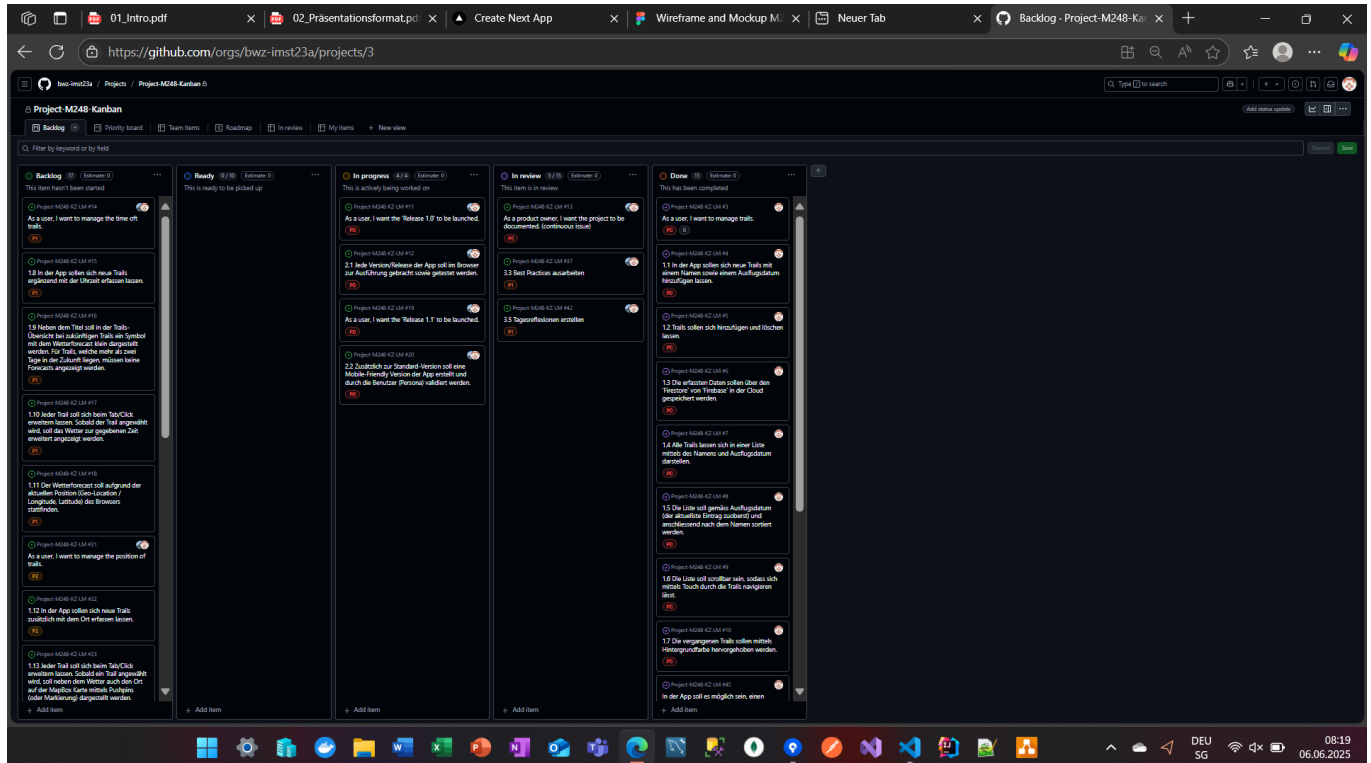
Stand nach Woche 2:



Stand nach Woche 3:



Stand nach Woche 4 (respektive nach den entsprechend über daheim nachgeholten Arbeiten):



Wir sehen also, dass wir zuletzt die nötigsten Tasks abgearbeitet haben.

Die Versionsverwaltung ist über Git erfolgt, wobei wir für jedes Teammitglied einen Working-Branch angelegt haben, von wo aus wir unseren aktuellen Stand jeweils auf den Main-Branch gemerged haben:

Alle Zweige	<input checked="" type="checkbox"/> Zeige Branches von Remote	Nach Datum ordnen	Author Name	Springe zu:
Graph				
Beschreibung			Datum	Commit
origin/Working_branch_Karl add final touches, structural changes and placeholder when there are no trails			6 Jun 2025 11:25	SOSmichi08 <karl. 1b39567
Working_branch_Lars 41 origin/main Merge pull request #52 from bwz-imst23a/Working_branch_Lars			6 Jun 2025 9:36	LarsMarty <lars.n 079a2a3
origin/Working_branch_Lars Add the login button to the header of the registration page			6 Jun 2025 9:32	Lars Marty <lars.m fbe6938
Add a primed background for the main sections and the footer of all pages in the desktop version of the app			6 Jun 2025 9:23	Lars Marty <lars.m d6a4602
Merge pull request #51 from bwz-imst23a/Working_branch_Karl			6 Jun 2025 8:29	IMST2023a_QARRL 8b934b2
Add reflections for missing days and improved overall document structure and errors			6 Jun 2025 8:25	SOSmichi08 <karl. 78cbb06
Merge pull request #50 from bwz-imst23a/Working_branch_Lars			5 Jun 2025 21:41	LarsMarty <lars.m 53b37e2
Change the indentations in certain files for a better overview, and add minor adjustments for the desktop version			5 Jun 2025 21:40	Lars Marty <lars.m 6f1169f
Remove comments, add advanced styling, add and implement color, font size and font family variables in every page, add redirect feature, add password reveal, add desktop sty			29 Mai 2025 13:33	SOSmichi08 <karl. a7a8e97
Add styling and structuring for multiple pages			28 Mai 2025 15:01	SOSmichi08 <karl. 83813a1
Merge pull request #49 from bwz-imst23a/Working_branch_Lars			28 Mai 2025 10:20	LarsMarty <lars.m 3f8e6c6
Add some comments to support the further styling of the pages and point out previous mistakes that have been made			28 Mai 2025 10:19	Lars Marty <lars.m 6957941
Change some text from German to English			27 Mai 2025 9:11	Lars Marty <lars.m 9bc1130
Change the main page so that a trail can be deleted			27 Mai 2025 8:35	Lars Marty <lars.m 6f2fc24
Merge pull request #48 from bwz-imst23a/Working_branch_Lars			26 Mai 2025 20:06	LarsMarty <lars.m d11c836
Change the main page so that a user's trails are displayed neatly sorted and change the edit page so that the corresponding trails can be edited			26 Mai 2025 20:01	Lars Marty <lars.m 0ad393f

6. Reflexionen

6.1 Tag 1 (02.05.2025)

Karl:

Heute planten Lars und Ich den Ablauf und die Struktur des Projekts. Wir erstellten eine Art von Zeit-/Terminplan in Form eines Kanban-Boards, indem wir verschiedene Aufgaben festhalten und aufteilen konnten. Um unsere Projektdateien nahtlos zu teilen und aktualisieren, erstellten wir ein Git-Repository, auf dem wir beide einen eigenen Branch haben, um effizient Änderungen an Projektdateien vorzunehmen. Bis zur nächsten Reflexion will ich den Designer Figma erfolgreich aufgesetzt und schon mit den ersten Konzepten des Wireframes begonnen haben.

Lars:

Am ersten Projekttag stand die Planung im Vordergrund. Wir erstellten gemeinsam einen Projektplan in Form eines Kanban-Boards, setzten ein gemeinsames Git-Repository auf und erstellten eine entsprechende Projektdokumentation. Meine persönliche Aufgabe wird es nun sein, bis zur nächsten Durchführung die diversen Technologieevaluationen durchzuführen, die für unser Projekt benötigt werden.

6.2 Tag 2 (09.05.2025)

Karl:

Was habe ich seit der letzten Reflexion erreicht?

Seit der letzten Reflexion konnte ich Figma erfolgreich aufsetzen, die mobile Wireframes designen und konstruieren. Ebenso konnte ich dies mit zwei der totalen acht Desktop-Wireframes. Auch konnte ich die erste Mockup-Page designen und «zusammenbasteln».

Was werde ich bis zur nächsten Reflexion erreichen?

Bis zur nächsten Reflexion werde ich die Mockup-Wireframes fertig haben und mit dem coden der Webseite angefangen haben.

Gab es Hindernisse? Und welche Hindernisse waren das?

Heute gab es keine bedeutsamen Hindernisse, auf die ich stossen musste.

Lars:

Zwischen der ersten und der zweiten Durchführung konnte ich die entsprechenden Technologieevaluationen für unser Projekt durchführen. Ich hatte dabei schon Vorstellungen, wie wir diese Technologien sinnvoll einsetzen könnten. Nach Rücksprache mit dem Projektleiter mussten wir den Einsatz dieser Technologien jedoch umplanen. Um zu sehen, ob die neue Idee zum Einsatz der Technologien nun entsprechend funktioniert, musste ich ein Testprojekt aufsetzen. Nachdem dies erfolgreich gelungen war und durch den Projektleiter abgesegnet wurde, konnten wir uns weiter in Richtung Umsetzung begeben, wobei ich am Nachmittag dann eher noch meinem Kollegen beim Usability-Workshop geholfen habe. Somit heisst es für mich nun möglichst bis zur nächsten Durchführung am Code zu arbeiten. Denn bis zum Mittag der nächsten Durchführung soll der erste Release (1.0) erscheinen.

6.3 Tag 3 (16.05.2025)

Karl:

Was habe ich seit der letzten Reflexion gemacht:

Seit der letzten Reflexion konnte ich erfolgreich das mobile mockup, desktop mockup und desktop wireframe vervollständigen. Ich konnte auch die roll-out Funktion in der MainPage zum Laufen bringen und ein erstes funktionelles Konzept der search Funktion realisieren.

Was will ich bis zur nächsten Reflexion erreichen?

Bis zur nächsten Reflexion, will ich mindestens den Release 1.0 fertig haben und mit dem Release 1.1 begonnen haben.

Hatte ich irgendwelche Schwierigkeiten? / Wenn ja welche?

Heute hatte ich gravierende Schwierigkeiten mit dem Git-Repository, da es viele Merge-Konflikte gab und die Behebung dieser, uns viel Zeit nahm.

Lars:

Den dritten Projekttag konnte ich nutzen, um den Grundstein für das Frontend zu legen. Eigentlich wäre geplant gewesen, dass wir an diesem dritten Projekttag bereits weiter sind und den ersten Release bereits abschliessen können. Aufgrund von Problemen mit der Versionsverwaltung seitens meines Teampartners kam es aber nicht dazu. Nun wäre das Ziel, dass wir bis und mit der vierten Durchführung des Projektmoduls, den ersten Release fertigstellen.

6.4 Tag 4 (23.05.2025)

Karl:

Was habe ich seit der letzten Reflexion erreicht?

Seit der letzten Reflexion konnte ich erfolgreich alle Seiten der Applikation bis auf die Login und Register Pages verlinken und richtig stylen, sodass sie wie im Mockup aussehen.

Was werde ich bis zur nächsten Reflexion erreicht haben?

Bis zur nächsten Reflexion werden ich mindestens den Release 1.1 der Projektvorgaben vervollständigt haben.

Gab es irgendwelche Schwierigkeiten, wenn ja welche?

Es traten Schwierigkeiten beim Platzieren der Icons innerhalb der Input-Felder auf. Ich konnte die «Navbar» auch nicht richtig strukturieren. Diese Schwierigkeiten konnte ich jedoch in der Zwischenzeit alle überwinden.

Lars:

Am vierten Projekttag habe ich die restliche Funktionalität implementiert. Vieles davon musste allerdings in Heimarbeit erfolgen, da ich über weite Strecken auf Code von meinem Teampartner angewiesen war und deshalb auf ihn warten musste. Dies war definitiv nicht förderlich. Schlussendlich konnte ich aber alle Funktionalitäten auch per Heimarbeit noch fertigstellen.

6.5 Gesamtfazit (06.06.2025)

Karl:

Im Gesamten ist das Projekt äusserst flüssig verlaufen. Es gab ein paar "holprige" Stellen, wie ein Problem mit unserem Github-Repository, die wir jedoch alle beseitigen konnte. Zeitlich war es nicht möglich sehr viele zusätzliche Features zu realisieren. Wir konzentrierten uns dafür nur auf die "MUST" Anforderungen und schauten, dass diese perfekt, nahtlos und effizient funktionieren. Mir gefiel das Projekt sehr, da man viele Freiheiten hatte wie z.B. das Framework oder das ganze Styling. Auch der Umgang mit Figma und das Realisieren der Konzepte hat mir viel Spass gemacht. Im Grossen und Ganzen habe ich auch viel über den Umgang mit JavaScript-Frameworks, Figma und das realisieren der Benutzerfreundlichkeit der Website.

Lars:

Insgesamt haben wir mit unserem Projekt ein Endresultat erreicht. So weit so gut. Allerdings gab es immer wieder Schwierigkeiten in der Teamarbeit, was zu enormen Zeitverlusten führte. Somit ist das Projekt aus meiner Sicht kein Vollerfolg. Mit dem Resultat kann man soweit aber trotzdem zufrieden sein.

7. Git-Repository

<https://github.com/bwz-imst23a/Project-M248-KZ-LM.git>

Letzter Commit: <https://github.com/bwz-imst23a/Project-M248-KZ-LM/commit/cac87e354bcc481f94773b912dc43866faf5d3d0>

8. 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>