

Doku GIBZ-Fiesta

Ramon Camenzind, Timo Schlumpf & André Möll

M326

GIBZ

**Inhaltsverzeichnis**

[THEORIE 3](#_Toc122116043)

[1 Vorwort 3](#_Toc122116044)

[2 Aufgabenstellung 3](#_Toc122116045)

[2.1 Aufbau 3](#_Toc122116046)

[2.1.1 Projektmitglieder 3](#_Toc122116047)

[2.2 Funktionen 4](#_Toc122116048)

[2.2.1 Mindestanforderungen 4](#_Toc122116049)

[2.2.2 Erweiterte Anforderungen 4](#_Toc122116050)

[3 Mittel und Methoden 5](#_Toc122116051)

[4 Vorkenntnisse 5](#_Toc122116052)

[5 Zeitplan 5](#_Toc122116053)

[6 Arbeitsjournal 7](#_Toc122116054)

[KW 35 (29.08 – 04.09) - Projektstart 7](#_Toc122116055)

[KW 36 (05.09 – 11.09) - Planungsbeginn 7](#_Toc122116056)

[KW 37 (12.09 – 18.09) – Planung beenden 7](#_Toc122116057)

[KW 38 (19.09 – 25.09) – Realisierungsbeginn 7](#_Toc122116058)

[KW 39 (26.09 – 02.10) – Realisierung 8](#_Toc122116059)

[KW 40 (03.10 – 09.10) – Realisierung 8](#_Toc122116060)

[KW 41 (10.10 – 16.10) – Realisierung 8](#_Toc122116061)

[KW 42 (17.10 – 23.10) – Realisierung 8](#_Toc122116062)

[KW 43 (24.10 – 30.10) – Realisierung 9](#_Toc122116063)

[KW 44 (31.10 – 06.11) – Realisierung 9](#_Toc122116064)

[KW 45 (07.11 – 13.11) – Realisierung 9](#_Toc122116065)

[KW 46 (14.11 – 20.11) – Realisierung 10](#_Toc122116066)

[KW 47 (21.11 – 27.11) – Realisierung 10](#_Toc122116067)

[KW 48 (28.11 – 04.12) – Realisierungs Schluss 10](#_Toc122116068)

[KW 49 (05.12 – 11.12) – Finalisierung 10](#_Toc122116069)

[KW 50 (12.12 – 18.12) – Abgabe 11](#_Toc122116070)

[PRAXIS 11](#_Toc122116071)

[1 Einleitung 11](#_Toc122116072)

[1.1 Ausgangslage 11](#_Toc122116073)

[1.2 Umsetzung (Zusammenfassung IPERKA) 11](#_Toc122116074)

[2 Informieren 11](#_Toc122116075)

[3 Planen 12](#_Toc122116076)

[3.1 Mockups 12](#_Toc122116077)

[3.2 Klassendiagramm 14](#_Toc122116078)

[3.3 Gesamtsystem 14](#_Toc122116079)

[4 Entscheiden 15](#_Toc122116080)

[4.1 Programmiersprachen 15](#_Toc122116081)

[4.2 Skills VS User 16](#_Toc122116082)

[4.3 Admin Berechtigungen 16](#_Toc122116083)

[4.4 Anzeige der Skills / Subskills 16](#_Toc122116084)

[5 Realisieren 16](#_Toc122116085)

[5.0 Lernphase 16](#_Toc122116086)

[5.2 Frontend 17](#_Toc122116087)

[5.2.1 Homepage / Landing Page 17](#_Toc122116088)

[5.2.2 Profil 18](#_Toc122116089)

[5.2.3 Skill-Seite 18](#_Toc122116090)

[5.2.4 Admin-Panel 19](#_Toc122116091)

[5.3 Backend 22](#_Toc122116092)

[5.4 Verbindung Backend – Frontend 22](#_Toc122116093)

[5.4.1 Cookies, die nicht gespeichert werden 22](#_Toc122116094)

[6 Kontrollieren 23](#_Toc122116095)

[7 Auswerten 23](#_Toc122116096)

[7.2 Probleme 23](#_Toc122116097)

[7.2.1 Cookies 23](#_Toc122116098)

[7.2.2 Aktualisierung des UI bei Änderungen 23](#_Toc122116099)

[7.2.3 Accordions auf Admin Page 23](#_Toc122116100)

[7.3 Verbesserungen 24](#_Toc122116101)

[7.4 Fazit 24](#_Toc122116102)

[8 Anhang 25](#_Toc122116103)

[8.1 Linkverzeichnis 25](#_Toc122116104)

# THEORIE

## 1 Vorwort

Diese Dokumentation dient zur Erklärung und Festhaltung von relevanten Eckdaten der Projektarbeit des Jahres 2022 im Modul 326 am GIBZ.

Die Dokumentation ist dabei in zwei Teile unterteilt: Theorie und Praxis – Theorie dient dabei zur vorgängigen Klärung der Umstände und Angehensweise, während es in der Praxis dann um die tatsächliche Umsetzung des Projektes geht.

Für die Realisierung des Projektes wurde die Projektplanungsmethode IPERKA verwendet.

Dieses Projekt trägt den Namen **GIBZ FIESTA**.

## 2 Aufgabenstellung

Im Modul 326 soll gemäss Modulidentifikation eine objektorientierte Analyse (OOA) in ein objektorientiertes Design (OOD) überführt sowie implementiert, getestet und dokumentiert werden.

### 2.1 Aufbau

Dieses Projekt dient unter Anderem zur Vorbereitung auf die praktische Abschlussarbeit, welche wir als Abschluss der Informatiker-Lehre ablegen müssen. In besagter PA haben die Individuen jeweils 10 Tage Zeit, ein Projekt in die Realität umzusetzen. Dieses Projekt soll also helfen, um Erfahrungen mit einem längerfristigen Projekt und dessen Ablauf zu sammeln und erweitern.

Inhaltlich geht es in der Projektarbeit des Moduls 326 um die Erstellung einer Software, welche von den Lehrpersonen des GIBZ zur langfristigen Evaluation ihrer digitalen, unterrichtsbezogenen Anwendungskompetenzen genutzt werden kann: Aus einem Set definierter Kompetenzen sollen Lehrpersonen per Selbstdeklaration die individuelle Erreichung unterschiedlicher Kompetenzen erfassen und planen. Zur Unterstützung der Lehrpersonen können für die Kompetenzen unterschiedliche Ressourcen bereitgestellt werden. Wir haben uns hier für ein Webtool entschieden. Die Anforderungen an dieses Tool sind unten aufgelistet.

#### 2.1.1 Projektmitglieder

Am Projekt sind folgende Personen beteiligt:

|  |  |
| --- | --- |
| Name | Rolle |
| Ramon Camenzind | Projektmitglied |
| Timo Schlumpf | Projektmitglied |
| André Möll | Projektmitglied |
| Peter Gisler | Experte, Fachvorgesetzte Person |

### 2.2 Funktionen

#### 2.2.1 Mindestanforderungen

Die Mindestanforderung für die Projektarbeit umfasst die Konzeption, Realisierung, Testung und Dokumentation eines Tools für die Erfassung von unterrichtsbezogenen Informatik-Anwendungskompetenzen.

Benutzerinnen und Benutzer mit spezieller Berechtigung können Anwendungskompetenzen definieren. Dabei wird jede Kompetenz einem frei wählbaren Kompetenzbereich und einer Kompetenzstufe innerhalb dieses Bereichs zugewiesen. So könnte beispielsweise die Kompetenz *Ein Inhaltsverzeichnis erstellen* zum Kompetenzbereich Textverarbeitung gehören und auf der zweiten von insgesamt drei Kompetenzstufen eingeordnet sein.

Jeder Kompetenz können beliebig viele Ressourcen (URLs) zugeordnet werden. Die Inhalte der Ressourcen unterstützen die Lehrpersonen bei der Erreichung der jeweiligen Kompetenz. Die Ressourcen werden durch die gleichen Personen verwaltet, welche die Kompetenzen und Kompetenzbereiche verwalten.

Lehrpersonen können die zu entwickelnde Applikation nutzen, um ihre persönlichen Anwendungskompetenzen zu erfassen und weiterzuentwickeln. Dazu können Sie alle Kompetenzen (inkl. Ressourcen) einsehen, die individuell vorhandenen Kompetenzen in anonymisierter Form erfassen und persistieren. Zu einem späteren Zeitpunkt soll es möglich sein, den Kompetenzzuwachs abzubilden, indem die zusätzlich erworbenen Anwendungskompetenzen ergänzt werden.

#### 2.2.2 Erweiterte Anforderungen

Erweiterte Anforderungen ergänzen die Mindestanforderungen. Die nachfolgend aufgeführten, erweiterten Anforderungen sind keine abschliessende Aufzählung.

* **Geplante Kompetenzentwicklung:** Lehrpersonen können mit der zu entwickelnden Applikation Ihre persönliche Kompetenzentwicklung planen. Dazu werden einzelne Kompetenzen durch eine spezielle Markierung vorgemerkt. Auf diese Weise kann die Lehrperson die individuelle Weiterbildung im Bereich der IT-Anwendungskompetenzen im Rahmen der regelmässig stattfinden Mitarbeiterfördergesprächen mit der zuständigen Person planen.
* **Berufsspezifische Kompetenzen:** Berufsspezifische IT-Anwendungskompetenzen können zur Ergänzung der allgemeinen IT-Anwendungskompetenzen erfasst werden. Diese berufsspezifischen Kompetenzen werden nur auf Lehrpersonen des jeweiligen Berufes angewendet.
* **Berufsspezifische Kompetenzstufen:** Die vordefinierte Zuweisung der Kompetenzen zu einer spezifischen Kompetenzstufe kann durch berufsspezifische Überschreibungen ergänzt werden. Dabei kann sowohl eine Erhöhung als auch eine Verringerung der Kompetenzstufe für einzelne oder mehrere Berufe des GIBZ erfasst werden.
* **Auswertungsmöglichkeiten:** Berechtigte Benutzerinnen und Benutzer können eine anonymisierte Übersicht der erreichten IT-Anwendungskompetenzen einsehen. Dabei wird wahlweise der absolute und/oder relative Anteil der Kompetenzerreichung für alle Kompetenzen ausgewiesen. Mit verschiedenen Filterkriterien – beispielsweise Kompetenzbereich, Kompetenzstufe oder Berufsgruppe – kann die Darstellung Auswertung modifiziert werden.

### 2.3 Bezug zur Realität

Unser Endprodukt sollte am Ende der Realisierung in der realen Welt einsetzbar sein. Zum Beispiel hier am GIBZ. Denn in gewissen Informatik Modulen mussten wir selbstständig an Kompetenzen Arbeiten. Das Tool würde es uns Schülern, und auch Lehrern, viel einfacher machen, die bereits Abgeschlossenen Kompetenzen zu markieren. Denn bisher mussten wir immer ein PDF editieren.

## 3 Mittel und Methoden

Da wir vorhaben, eine Web-Applikation zu entwickeln, brauchen wir passende Programmiersprachen und Tools, um dieses Vorhaben umzusetzen.

Um die Seite zu entwickeln, werden wir verschiedene Tools verwenden. Zum einen wäre da **VSCode**, ein gratis Open-Source Texteditor.

Zusätzlich werden wir **Datagrip** und **Webstorm** der Jetbrains Toolbox verwenden, um allfällige Probleme mit VSCode zu kompensieren.

Für das Frontend haben wir uns für **Svelte** entschieden. Es ist einfach zu verstehen und vielseitig einsetzbar.

Beim Backend fiel unsere Entscheidung auf **Rust**, da es eine moderne, effiziente (bezüglich der Ressourcen) und beliebte Sprache ist, die für Backend-Services viel verwendet wird.

Natürlich müssen die Daten auch irgendwo gespeichert werden. Unsere Wahl fällt auf **MongoDB**, da wir mit diesem Tool schon vertraut sind und es Erfahrungsgemäss einfach zu verwenden ist.

Im Teil **Entscheiden** vom Praxis-Teil wird noch einmal etwas genauer auf diese Entscheidungen eingegangen.

## 4 Vorkenntnisse

Wir sind alle angehende Informatiker (Fachrichtung Applikationsentwicklung) im 3. Lehrjahr. In der Schule verwenden wir primär die Programmiersprache C#. Für dieses spezifische Projekt wird allerdings Svelte / Rust verwendet.

Ramon hat schon kleine Svelte-Kenntnisse, Rust hingegen ist für alle neu. Es muss also auch eine gewisse Zeit für das Erlernen der Grundkenntnisse eingeplant werden. MongoDB allerdings haben wir alle schon genutzt und verstehen wir zum grossen Teil auch recht gut. ++Genauer beschreiben

## 5 Zeitplan

Wie in folgender Tabelle ersichtlich ist, hatten wir für dieses Projekt Zeit vom **26.08.22** bis zum **16.12.22**.

Dies beinhaltet einen Termin von Zwischengesprächen, an welchen wir die Möglichkeit haben, uns mit der Fachperson über unseren bisherigen Stand zu unterhalten, allfällige Probleme bereits zu behandeln und die weitere Vorgehensweise zu besprechen. Zudem gibt es nach der Abgabe noch, ganz nach dem Ablauf der IPA, ein Fachgespräch mit dem Experten welches Teil der Bewertung sein wird.

|  |  |
| --- | --- |
| Datum | Termin |
| 26.08.22 | Modulstart |
| 02.09.22 | Abgabe des Projektauftrags / Projektstart |
| 09.09.22 | Beginn Umsetzungsarbeiten |
| 04.11.22 | Zwischengespräche |
| 11.11.22 |
| 16.12.22 | Abgabetermin Projektarbeit / Projektbericht |
| 13.01.23 | Fachgespräche |
| 20.01.23 |
| 22.01.23 | Notenabgabe |



## 6 Arbeitsjournal

### KW 35 (29.08 – 04.09) - Projektstart

André, Ramon & Timo

Zu Beginn unseres Projektes mussten wir den Projektauftrag mit allen Spezifikationen erstellen. Dazu gehört selbstverständlich auch ein Projekt Name. Lustig wie wir sind, haben wir auf GitHub so lange Repository Namen generiert, bis irgendwas Knackiges dabei war. Irgendwann kam dann mal das Wort „Fiesta“ vor und wir wussten sofort, dass unser Projekt „GIBZ-Fieśta“ heisst. Dass es ein wenig interessanter aussieht, wählten wir ein spezielles ś.

### KW 36 (05.09 – 11.09) - Planungsbeginn

André, Ramon & Timo

Wir waren immer noch mit dem Projektauftrag beschäftigt. Da es das erste Mal war, das wir so ein Projektauftrag schrieben, dauerte es etwas länger, wir wurden aber in dieser Woche fertig damit. Wir haben auch schon die Projektstruktur erstellt, also ein Server und ein Client Ordner mit den dazugehörigen Frameworks.

### KW 37 (12.09 – 18.09) – Planung beenden

Ramon

Zur Planung gehören ebenfalls Mockups, welche eine Design Vorlage geben. Ich wollte mir hier mehr Zeit nehmen als sonst, damit ich nachher, während dem Realisieren nicht noch mittendrin das GUI ändern muss.

Timo

Nach jener Planungsphase fing bei mir die Planung erst richtig an. Ich durfte mir viele Gedanken darüber machen, wie denn nun unser Backend aussehen soll. Ich entschied mich zusammen mit Ramon & André dafür, dass wir die Routes in „Skills“ und „User“ einteilen.

André

Ich habe mich heute ans Klassendiagramm gesetzt und mir bereits zu der Darstellung des Gesamtsystem in Modellform Gedanken gemacht.

### KW 38 (19.09 – 25.09) – Realisierungsbeginn

Ramon

Ich habe mit Svelte die Homepage und die Navigation erstellt. Allerdings ist mir dann aufgefallen, dass Svelte gar kein Routing System besitzt. Daraufhin habe ich das Projekt mit SvelteKit neu erstellt. Das ist zwar ein wenig anders, was die Filestruktur anbelangt, aber hat auch viel mehr Features. Das ärgerte mich natürlich ein wenig, aber es hat auch was Gutes. Denn wäre das nicht passiert hätte ich alles CSS selbst gemacht, was sehr schwer gewesen wäre, vor allem wenn die Webseite noch responsive sein muss. Nun habe ich mich für TailwindCSS als CSS Framework entschieden, für das es auch viele Component Libraries gibt.

Timo

Es war seit Anfang an ziemlich klar, dass wir das Backend in Rust machen werden. Keiner von uns hat auch nur annähernd etwas damit gemacht, weshalb dies ein Sprung ins kalte Wasser wird – was wir uns auch bewusst sind. Nach ein bisschen Googeln fand ich das Framework „rocket-rs“, dass man für API’s verwenden kann, für welches ich mich direkt entschied, da ihre Homepage sehr ansprechend aussah ☺. Danach baute ich die Grundstruktur des Backends mal auf, unterteilte die verschiedenen Teilbereiche (DB-Anbindung, API Endpoints, Helper – Funktionen, usw.) in dedizierte Ordner, um mir das Ganze einfacher zu machen.

André

Bei mir gabs den Startschuss für die Dokumentation und ich habe mich darum gekümmert, mal einen Grundgerüst für diese auf die Beine zu stellen. Dafür habe ich ähnlich wie beim Projektauftrag auch schon den Projektauftrag erneut durchgelesen und die Anforderungen festgehalten. Ich hatte ausserdem etwas mit GitHub und SSH Keys zu kämpfen, da ich mehrere Accounts auf demselben Laptop nutze und auch bisher primär mit GitLab gearbeitet hatte, wodurch ich dann Verbindungsprobleme hatte.

### KW 39 (26.09 – 02.10) – Realisierung

Ramon

Ich gebe zu, in dieser Woche habe ich eher weniger gemacht. Wir haben schliesslich noch andere Fächer in der Schule und mit einer Lernzielvereinbarung sollte man schauen, dass die Noten auch in den BM Fächern passen.

Timo

Wie auch Ramon habe ich in dieser Woche nicht allzu viel gemacht, da wir in den anderen Fächern ziemlich Druck hatten.

André

Das GitLab Problem ließ sich nun so halb lösen, zumindest konnte ich mittlerweile das Repo korrekt klonen und habe die Daten nun auch lokal verfügbar. Zudem habe ich mit dem Festhalten des I von Iperka in der Dokumentation begonnen und dabei auch auf die Aussagen der anderen zurückgegriffen.

### KW 40 (03.10 – 09.10) – Realisierung

Ramon

Da ich letzte Woche nichts gemacht habe, muss diese Woche mehr fertig werden als geplant. Ich habe also die Homepage mit samt Navigation neu gemacht. Dieses Mal mit TailwindCSS und der Component Library Flowbite-Svelte. Das erleichterte die Design Aufgabe sehr.

Timo

Nach einer „relativ“ gemütlichen Woche setzte ich mich als erstes an die Datenbank – Anbindung zu MongoDB. Dies stellte sich als mühsamer und schwieriger heraus als ursprünglich gedacht, da es zu den meisten MongoDB – bezogenen Libraries (Crates) keine / bzw. wenig Dokumentation gibt. Nach langem ausprobieren und herumspielen bei welchem ich über die Library [r2d2](https://docs.rs/r2d2/latest/r2d2/) (cooler Name, aber in unserem Projekt ziemlich nutzlos) gestolpert bin, fand ich dann doch eine Lösung, die sich für unser Projekt eignete.

André

Da das Klassendiagramm bereits erstellt wurde, konnte ich mich heute auf das E von IPERKA fokussieren. Zudem habe ich mich selbst auch noch mit Svelte auseinandergesetzt, um nicht den Anschluss zu verlieren.

### KW 41 (10.10 – 16.10) – Realisierung

Ramon

In dieser Woche setzte ich mich an die Skill Seite, also dort wo man am Schluss die Kompetenzen einsehen und abschliessen kann. Das ist eine eher grosse Aufgabe, daher bin ich noch nicht ganz fertig geworden. Die Logik für das Einlesen und Darstellen der Daten fehlt noch.

Timo

Hoffnung ist meistens nicht alles – und auch so war es bei meinem Backend. Die Lösung von letzter Woche funktionierte bei weitem nicht komplett, was einiges an Zeit und Aufwand kostete, um sie mit viel Gebastel wieder zum Laufen zu bringen. Anscheinend war die Lösung nur für die Verwendung im aktuellen File gemacht, und nicht für ein Ordner- und Fileüberstreckendes System wie unseres. Ich fand dann heraus, dass man Variabeln mit diesem Framework für alle anderen Files auch verfügbar machen kann, was dann das Problem löste.

André

Heute war bei mir eher wenig los, da der Dokumentationsteil vor allem in den späteren Stufen sehr auf Implementation von Frontend und Backend basiert. Ich habe mir jedoch schon erste Testfälle für das manuelle Testing ausgedacht.

### KW 42 (17.10 – 23.10) – Realisierung

Ramon

Die Skill seite ist nun fertig und auch die Profil Seite ist gemacht. Dort kann man als unangemeldeter User sehen, wer welche skills schon fertig gemacht hat.

Timo

Diese Woche stellte ich die User Routes für das Login, Logout, Erstellen eines Users und Löschen eines Users fertig.

André

Da das Frontend und Backend heute gut vorangekommen ist, konnte ich auch gut mit der Doku weiterfahren und auch schon erste Dinge im Realisationsteil festhalten. Zudem halte ich auch den Zeitplan sowie Arbeitsjournal mehr oder weniger auf dem Laufenden.

### KW 43 (24.10 – 30.10) – Realisierung

Ramon

Alle Unterseiten waren Fertig bis auf das Admin Panel. Man will schliesslich nicht in der Datenbank die User manuell hinzufügen. Diese Seite erweist sich als die Komplexeste. Ich muss sehr viele Untermenus machen und Popups. Im Mockup habe ich leider ein Wichtiger Teil vergessen. Und zwar, wo sieht man die Bereits registrierten Benutzer? Das musste ich mit einem Popup lösen, in dem eine Tabelle angezeigt wird mit den bestehenden Usern. Auch hier bin ich aber noch nicht ganz fertig, denn bis jetzt ist erst das User Management vorhanden und die Skills sind noch nicht erstellbar. Das muss dann nächste Woche fertig werden.

Timo

Alle User – bezogenen Routes (Login, Logout, Create, Delete, Update) sind nun fertig, jedoch habe ich bei „Update“ noch kleine Probleme, die ich nächste Woche beheben werde.

André

Mein GitLab Problem ist nun endgültig behoben und ich konnte mir den neusten Stand ziehen, einige kleine Anpassungen machen und anhand des Standes die Dokumentation aktualisieren.

### KW 44 (31.10 – 06.11) – Realisierung

Ramon

Gesagt getan, das Admin Panel ist nun fertig. Es war mehr Aufwand als zuerst angenommen aber jetzt das es fertig ist bin ich sehr zufrieden mit dem Ergebnis.

Timo

Nach einem Hinweis von Ramon – dass man ja das User Objekt komplett löschen und angepasst wieder einfügen konnte (jedoch die ObjectId, equivalent zum PRIMARY KEY in SQL, behält) – machte das die Update Route nicht nur effizienter, sondern auch schöner und verständlicher. Dieses Prinzip übernahm ich direkt in die Update Route für die Skills, da es sich dort um die genau gleiche Logik handelte.

André

Das Admin Panel ist fertig geworden und es war recht umständlich, wodurch auch die Dokumentation dieses Teils etwas aufwändiger wurde.

### KW 45 (07.11 – 13.11) – Realisierung

Ramon

Ich habe angefangen das Frontend mit dem Backend zu verbinden. Bisher hat es nur Mock Daten angezeigt und noch nichts gespeichert. Ich dachte die Verknüpfung zwischen Front- und Backend wäre einfach, da wir uns laufend über die Änderungen im Datenmodell informiert haben, aber es gab noch viele Hürden, besonders mit der Speicherung der Cookies.

Timo

Diese Woche startete – und beendete ich den Grossteil aller Skill Routes. Da es bei den meisten keine Unterschiede abgesehen vom Datenmodell gab, konnte ich sie einfach rüberkopieren. An die „Mark Skill“ & „Complete Skill“ Routes setze ich mich nächste Woche.

André

Nach Absprache mit FE und BE Devs gab es noch eine Sektion der Realisierung bezüglich Verbindung zwischen BE und FE, da sich das anscheinend schwerer gestaltete als anfangs angenommen.

### KW 46 (14.11 – 20.11) – Realisierung

Ramon

Ja, die Cookies, das ist so eine Sache. Ich habe im Backend Sachen geändert, die es vielleicht fixen – negativ, stundenlang auf Stack Overflow gesucht und schliesslich aufgegeben. Das ist allerdings ein Problem, denn bei jedem Page Reload ist man wieder ausgeloggt. Aber ich machte einfach mal weiter und habe alle /user/ Routes vom Backend ins Frontend übernommen. Diese funktionieren auch prächtig.

Timo

Nach ein bisschen herumprobieren mit den $pull & $push Funktionen von MongoDB (zum Löschen bzw. Hinzufügen von Elementen) funktionieren nun auch die Complete & Mark Routes.

André

Für heute hatte ich ehrlich gesagt vergessen, mein Arbeitsjournal zu schreiben. Jedoch war sicher der Hauptteil die Weiterarbeit an der Dokumentation.

### KW 47 (21.11 – 27.11) – Realisierung

Ramon

Die Cookies sind immer noch ein Problem und die Deadline rückt näher. Ich habe ungelogen die ganze Woche daran gesessen bis ich irgendwann im Frontend die Route Authentifizierung reworked habe und danach ist es ohne Probleme gegangen. Manchmal liegt das Problem nicht dort, wo man es vermutet.

Timo

Wie sich wieder einmal herausstellte, funktionierten die Routes von letzter Woche doch nicht ganz so wie erhofft. Nach langem Debuggen und Problemsuchen fand ich dann heraus, dass ich lediglich $pull mit $push vertauscht hatte. Nach der Behebung des Problems setzte ich mich an die Unit Tests, die wir bis in zwei Wochen auch fertig haben müssen.

André

Da sich das Projekt langsam, aber sicher dem Ende neigt, habe ich mich in dieser Woche dem Auswerten gewidmet und bereits erste Feedbacks der anderen eingeholt.

### KW 48 (28.11 – 04.12) – Realisierungs Schluss

Ramon

In dieser Woche habe ich noch die übrigen Routes ins Frontend eingebunden. Das waren eigentlich alle /skill/ Routes. Es gibt allerdings noch ein paar Bugs. Die behebe ich nächste Woche.

Timo

Diese Woche machte ich weiter an den Unit Tests, sowie ein bisschen „Aufräumen“ im Repo. Ich entfernte unnötige Dependencies, redundante Codestellen und sortierte die verschiedenen Routes in einer sinnvollen Reihenfolge.

André

Diese Woche war Vervollständigung und Testing angesagt: Viele kleinere Dinge wurden noch in der Doku ergänzt und ich habe gemerkt, dass doch noch etwas mehr fehlt als anfangs gedacht. Deshalb habe ich diese Woche auch zuhause noch etwas weitergearbeitet.

### KW 49 (05.12 – 11.12) – Finalisierung

Ramon

Das Frontend ist nun endgültig abgeschlossen, alle Bugs sind behoben und es funktioniert perfekt mit dem Backend zusammen.

Timo

Gute News – das Backend ist nun komplett. Sämtliche Bugs wurden (hoffentlich) behoben, und läuft nun sauber mit dem Frontend zusammen.

André

Ich musste noch die ein oder andere Überstunde einlegen, aber mit etwas Verspätung und finalen Touches ist die Dokumentation nun auch fertig.

### KW 50 (12.12 – 18.12) – Abgabe

Abgabe

# PRAXIS

## 1 Einleitung

Wir haben uns entschieden, diesen Projektauftrag in Form einer Webapplikation umzusetzen. Es folgt nun die Dokumentation der Umsetzung dieser Webapplikation.

### 1.1 Ausgangslage

Für das Modul 326 gilt es, ein Projekt nach dem Ablauf einer IPA durchzuführen mitsamt Dokumentation und Implementation. Die Situation ist folgende: Es wird eine digitale Lösung zur Kompetenzerfassung der Lehrpersonen des GIBZ gesucht. Momentan gibt es noch keine allgemeine Lösung, dies soll nun geändert werden.

### 1.2 Umsetzung (Zusammenfassung IPERKA)

Das Projekt wurde mit der Projektmethode IPERKA umgesetzt.

I – Zuerst fand die Informationsphase statt, bei der die Anforderungen definiert wurden und wir uns Gedanken dazu machten, was alles benötigt wird. ++

+++ Umsetzung (Zusammenfassung IPERKA), evtl Ergebnis

### 1.3 Versionierung

Das ganze Projekt mit Dokumentation und seine jeweiligen früheren Versionen ist auf GitHub abgelegt. Das Repository sollte öffentlich sein und ist unter folgendem Link erreichbar:

<https://github.com/seiemens/gibz-fiesta>

#### 1.3.1 CI/CD

Zudem wurde bei GitHub noch ein AutoDeploy aktiviert, wodurch eine Pipeline bei jedem Commit automatisch durchläuft und den Code prüft. Leider muss hier angemerkt werden, dass diese Pipeline eigentlich immer failed, da sich Rust und die Pipeline aus uns unbekannten Gründen nicht vertragen.

## 2 Informieren

*Beim Informieren geht es um die Beschaffung, Ordnung und Bewertung von Informationen*

Um was geht es? Aus einem Set definierter Kompetenzen sollen Lehrpersonen per Selbstdeklaration die individuelle Erreichung unterschiedlicher Kompetenzen erfassen und planen. Zur Unterstützung der Lehrpersonen können für die Kompetenzen unterschiedliche Ressourcen bereitgestellt werden.

Was wird benötigt? Um eine Webanwendung realisieren zu können, sind Kenntnisse und ein grundlegendes Verständnis in den verwendeten Programmiersprachen vonnöten oder zumindest sehr vorteilhaft. Deshalb ist es wichtig, hier eine Lösung zu finden, die allen passt. Auch für die DB braucht es eine Entscheidung, da es dort verschiedenste Ausführungen gibt.

Wozu dient das Programm? Im Grunde soll es für Lehrpersonen des GIBZ eine (Web-)lösung geben, bei welcher Sie ihre eigenen Kompetenzen ähnlich einer To-Do-Liste nach Können bzw. Nicht-Können bewerten können.

Wie lange haben wir Zeit? Start des Projektes war am 2.9.2022 und voraussichtlicher Abschluss beziehungsweise Abgabe der Arbeit wird am 16.12.2022 stattfinden.

Zusätzlich zu den oben vorgegebenen Anforderungen (Mindestanforderungen & Erweiterte Anforderungen) wurden vom Projektteam eigene, ergänzende Anforderungen definiert:

**Muss**

* **Dark-Mode Switch**: Um die Applikation auch visuell ansprechend zu gestalten und dabei die Augen trotzdem etwas zu schonen, soll eine Funktion eingebaut werden, die zwischen einem hellen und einem dunklen Design der Website wechseln lässt.
* **Responsive UI Design / Layout**: Damit die App auch auf mobilen Geräten seinen Charme behält und um die Benutzerfreundlichkeit zu steigern, soll das Design / Layout responsive umgesetzt werden, d.h. es passt sich jeweils entsprechend an die Grösse des Endgeräts an.
* **Authentifizierung für API-Requests:** Auch die Sicherheit der App soll nicht missachtet werden – So ist vorgesehen, dass API-Anfragen nur mit entsprechender Verifizierung gemacht werden können. Dies dient dazu, um unerwünschte Eindringlinge von der API fernzuhalten.
* **Passwort-Hash:** Datenschutz ist ebenfalls ein wichtiges Thema. So steht es ausser Frage, dass die Passwörter der Benutzer jeweils «gehasht» werden sollen, so dass diese nicht nur für allfällige Eindringlinge, sondern auch für die Systemadministratoren uneinsehbar sind.

**Kann**

* **Login-Sperre:** Bei einer dreimalig falschen Eingabe des Passworts soll der jeweilige Besucher für eine gewisse Dauer (3’) von weiteren Loginversuchen blockiert werden. Dies hilft unter Anderem gegen Missbrauch in Form von Brute Force Angriffen.
* **Erweiterte Logfunktionen:** Die Administration darf ebenfalls nicht zu kurz kommen – so soll die Dokumentation der Aktivitäten aktiviert werden, so dass für den Admin ersichtlich ist, wann sich ein Benutzer einloggt / ausloggt. Zudem sollen dabei jeweils die IP’s des benutzen Geräts ebenfalls festgehalten werden.

**Weiteres / Allgemeines**

* Objektorientierte Programmiersprache
* Testkonzept inkl. Beschreibung aller Testfälle

## 3 Planen

*In der Planungsphase werden Lösungswege gesucht und Modelle erstellt*

Bei der Wahl der Programmiersprache wurde uns die Entscheidung bedingt abgenommen, da wir zur Gruppenfindung bereits unsere präferierten Sprachen nennen mussten. Bei der Diskussion während der Teamfindung kamen einige davon auf:

Für das Frontend war die Debatte vor allem zwischen Svelte, einem JS-Framework und AngularJS. Beim Backend wollten wir grundsätzlich alle etwas Neues ausprobieren, da wir zu diesem Zeitpunkt mit C# (und teilweise auch Java) bereits einige Projekte realisiert hatten.

### 3.1 Mockups

Um später eine einfachere Arbeit beim Implementieren des Frontends zu haben oder auch nur schon zur Visualisierung von unseren Ideen und als Orientierungshilfe sind Mockups sehr hilfreich. Dabei sei gesagt, dass Mockups in den meisten Fällen nicht das finale Design darstellen. Nachfolgend sind die Mockups, anhand welcher das User Interface implementiert wurde.

Schön ersichtlich bei allen ist der Header, der auf allen Unterseiten einheitlich gestaltet wird und eine Navigation beinhaltet. Zudem wird das Ganze durch ein Logo auf der linken Seite ergänzt. Hier sei gesagt, dass das Logo nicht von uns entworfen wurde, sondern völlig fair und legal gestohlen wurde.

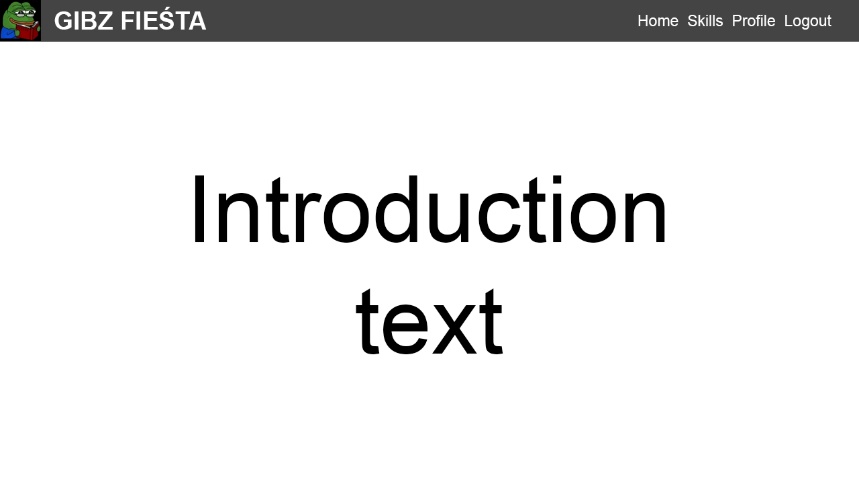
Die Homepage, die auch als Landing Page fungiert und grundsätzlich immer das erste ist, das der Benutzer sieht, soll aus diesem Grund nicht zu überladen sein. Daher bleibt sie neben einem gut ersichtlichen Begrüssungstext / Slogan praktisch leer. Dafür wird der Text in den Mittelpunkt gestellt und gewinnt umso mehr an Bedeutung.

Abbildung 1 Mockup Homepage

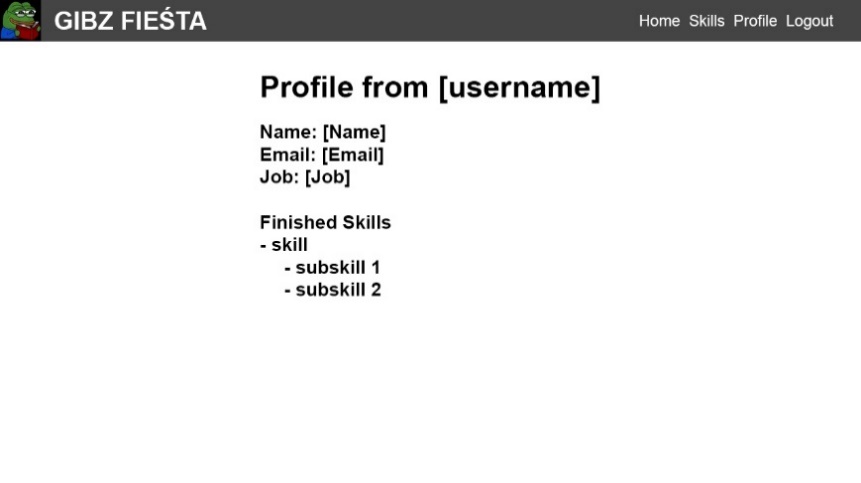
Sofern der Nutzer sich dann korrekt angemeldet hat, gelangt er über die Navigation oben wahlweise auf eine Übersicht zu seinem Profil mit Namen, E-Mail, Job sowie bereits erreichten Skills Kompetenzen.

Abbildung 2 Mockup Profil

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 3 Mockup Skills Übersicht

Auf der Unterseite «Skills» sind alle für den Benutzer verfügbaren Skills aufgelistet und der Benutzer hat die Option, einzelne «Subskills» jeweils auszuklappen und dort die jeweiligen Skill-spezifischen Ressourcen zu betrachten. Zudem soll es möglich sein, einzelne Skills sowie ganze Skillsets als erledigt zu markieren.

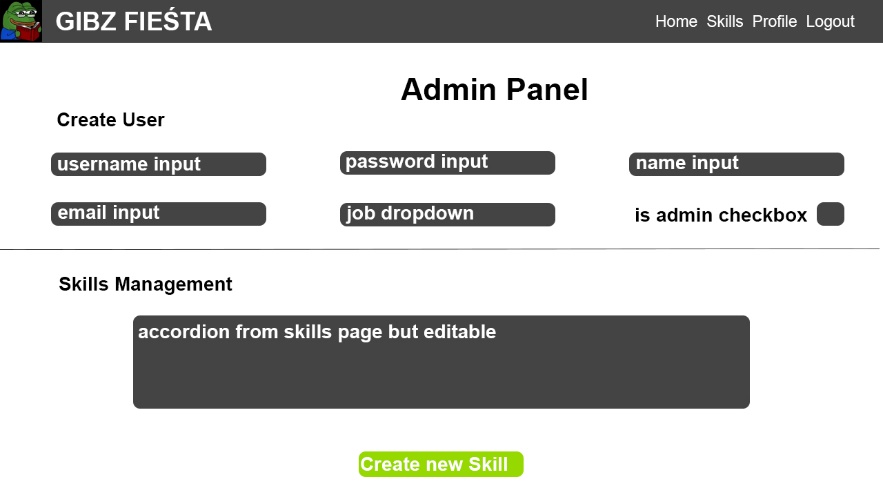


Abbildung 4 Mockup Admin Panel

Nicht ersichtlich für den normalen Benutzer aber trotzdem geplant ist zudem ein Administrations-Panel, in welchem der entsprechende Webmaster dann einerseits neue Benutzer erstellen kann, aber auch Bestehende bearbeiten und die existierenden Kompetenzen bearbeiten und erweitern.

### 

### 3.2 Klassendiagramm

Die Applikation basiert grundsätzlich auf zwei Objekten – Dem User-Objekt und dem Skill-Objekt. Das User-Objekt hat dabei nicht ein Array von Skills, wie das vielleicht in einem relationalen Modell der Fall wäre, sondern eine gehashte Liste mit ObjectIDs und dazugehörigen Levels, welche jeweils einen Skill und das abgeschlossene Level an Subskills referenziert. Die Skills setzen sich ausserdem aus einer diversen Anzahl Subskills zusammen, welche wiederum eine diverse Anzahl an Ressourcen referenzieren können. Da unsere Datenbank, **MongoDB**, eine nicht relationale Datenbank ist gibt es keine herkömmliche Relation zwischen User und Skill, da MongoDB intern nicht mit Tabellen arbeitet. So ergibt sich also folgendes Klassendiagramm:

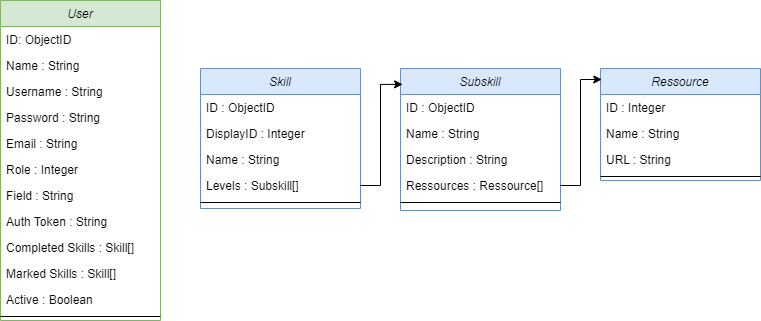


Abbildung 5 Klassendiagramm

### 3.3 Gesamtsystem

Zusammen mit der zuvor erwähnten Datenbank MongoDB ergibt sich so ein System bestehend aus Frontend, Backend, Datenbank und einer API in Form von einem Webserver. Das Frontend, welches mit SvelteKit realisiert wurde, sendet über das http Protokoll Anfragen an die (REST) API, die auf Rocket.rs, einem Rust Framework (vergleichbar mit express.js) das als Webserver dient, läuft. Die API fragt dann das Backend, welches mit Rust implementiert wurde an und dieses holt wiederum die Daten aus der DB, welche dann auf demselben Weg in Form eines JSONs zurück an das Frontend geliefert wird.

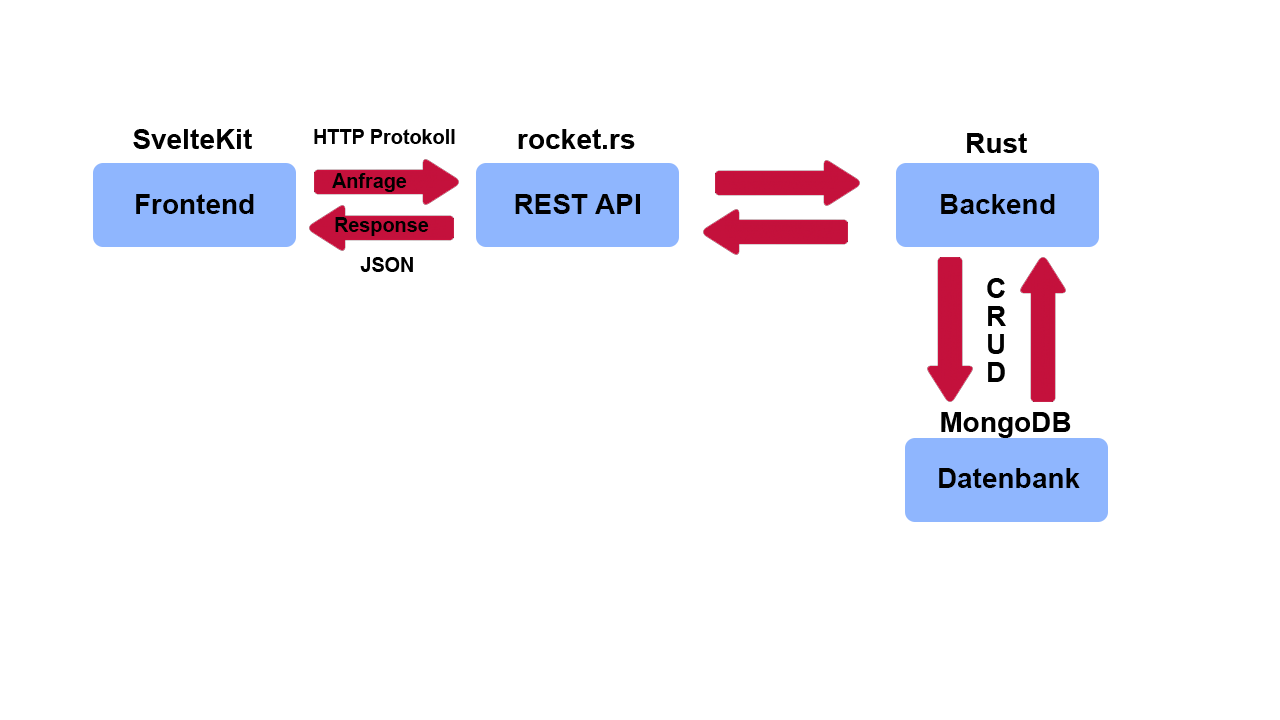


Abbildung 6 Modell des Gesamtsystems

## 4 Entscheiden

*Beim Entscheiden werden die Varianten verglichen und schlussendlich eine Entscheidung gefällt*

### 4.1 Programmiersprachen

Im Backend fiel der einstimmige Konsensus schlussendlich auf **Rust**, da dies für uns alle relativ Neuland war und wir trotzdem schon mal davon gehört hatten. So haben wir die Chance, innerhalb dieses Projektes eine komplette neue Programmiersprache kennenzulernen und das Gelernte direkt in die Tat umzusetzen. Und da wir alle schon von der Existenz von Rust Bescheid wussten, standen die Chancen hoch, dass es sich dabei um eine etablierte Sprache handelt, die bereits (für unser Projekt) ausreichend dokumentiert wurde.

Nach etwas Recherche zeigte sich ausserdem, dass Rust im Vergleich zu vielen anderen Programmiersprachen in Sachen Sicherheit weit vorne mit dabei ist. Zudem hat es keinen handelsüblichen Garbage Collector, dafür aber ein eigenes Typsystem. Dadurch kann die Performance erhöht werden, was jedoch auch bedeutet, dass man dieses Typsystem auch beherrschen muss. Ein weiterer Vorteil liegt in der Gleichzeitigkeit, die z.B. in C/C++ nicht so einfach umzusetzen ist, wie das in Rust möglich ist.

Für das UI und generell das Frontend haben wir uns relativ schnell auf Svelte geeinigt, da es einerseits einfach zu verstehen und vielseitig einsetzbar, während andererseits Ramon schon erste Erfahrungen mit Svelte gesammelt hat. Zusätzlich ist es moderner als viele der «grossen bekannten» Frameworks

Bei der Datenbank fiel die Entscheidung schlussendlich auf MongoDB. Obwohl die Applikation sich eigentlich für eine relationale Datenbank eignen würde, haben wir uns trotzdem dagegen entschieden, weil:

- Schutz vor SQL - Injections (Mongo - Injections gibts auch, aber viel seltener)

- Responses von Queries sind direkt in JSON - Format (keine Umwandlung nötig)

- Integrierte Library für div. Sprachen zur einfachen Verwendung von Queries

- Skalierbarer als MySQL (Resourcenfreundlicher)

- Bessere Übersicht für Admins

- Schnellere Responses bei simplen Queries

### 4.2 Skills VS User

Wenn es um verschiedene Skills geht, die mit verschiedenen Benutzern in Verbindung stehen, stellt sich schnell die Frage, wie diese Verbindung hergestellt werden soll. Bei einer Diskussion sind wir auf zwei unterschiedliche Varianten gekommen. Nachfolgend die Entscheidung, welche hierfür schlauer ist.

Es ist klar, dass wir jeweils ein Objekt für Benutzer und eines für Skills haben werden. Die erste Variante besteht aus einem Skill-Objekt, welches selbst mittels einem User-Array abspeichert, welche Nutzer alle schon diesen spezifischen Skill abgeschlossen haben. Bei der zweiten Variante handelt es sich um das Gegenteil: Auf dem User-Objekt wird stattdessen ein Skill-Array gespeichert, welches alle Skills enthält, die dieser jeweilige Benutzer bereits abgeschlossen hat.

Schlussendlich sind wir zum Schluss gekommen, dass die Variante 2 in Form eines Skill-Arrays auf dem User-Objekt mehr Sinn macht, da dann die Darstellung der erreichten Kompetenzen auf der «Skills»-Unterseite relativ simpel mit einer Abfrage dieses Skill-Arrays realisiert werden kann. Dagegen kann das User-Array aus Version 1 in dieser Form allerhöchstens für administrative Zwecke verwendet werden, für alles andere muss zuerst einmal gefiltert werden. Diesen Schritt sparen wir uns bei Variante 2.

### 4.3 Admin Berechtigungen

Etwas Kleines, das aber trotzdem erwähnt werden sollte, ist die Überlegung, die wir uns bei den Berechtigungen gemacht haben: Sie sind in unseren Augen logisch und doch erwähnungswert. Es ging dabei primär darum, inwiefern der Admin Zugriff auf Passwörter hat. Zum einen war klar (und es wurde im Zwischengespräch auch nochmal angesprochen), dass die Passwörter nur gehashed vorhanden sind. Doch sollte der Admin trotzdem eine Möglichkeit haben, einem User ein Passwort zu setzen. Deshalb soll es also möglich sein, einem User als Admin ein neues Passwort zu setzen und dabei das alte zu überschreiben, ohne dass das alte ersichtlich ist.

### 4.4 Anzeige der Skills / Subskills

Wir haben uns bei der Anzeige von Skills für ein Accordion-Element entschieden. Die andere Variante wäre ein Grid gewesen, jedoch sehen wir dort die folgenden Nachteile im Vergleich zum Accordion: Jeder Skill kann eine dynamische Anzahl Subskills haben, so kann einer beispielsweise 3 haben und ein anderer 4. Bei einem Grid würde das schnell zu Darstellungsproblemen bzw. zu einer unschönen Anzeige führen, da beim Grid je nach Anzahl Skills Leerräume entstehen können.

## 5 Realisieren

*Bei der Realisierung geht es hauptsächlich um die tatsächliche Umsetzung des zuvor Geplanten.*

### 5.0 Lernphase

Um eine Grundbasis für die Umsetzung zu schaffen, mussten erst unsere Kenntnisse in den gewählten Sprachen gestärkt beziehungsweise erlernt werden. Für Rust heisst das grundsätzlich ein Start bei null.

Bei Rust konnten wir auf die *offizielle Rust-Dokumentation* [[Link 1]](https://www.rust-lang.org/learn) zurückgreifen und uns dort einen ersten Überblick verschaffen. Mithilfe dieser und der Übersicht auf der *offiziellen Website* [[Link 2]](https://doc.rust-lang.org/stable/) konnten wir uns die nötigen Kenntnisse so weit aneignen. Was jedoch eher aufwändig war und viel mit Trial und Error erarbeitet wurde, war die Verbindung zwischen Rust und MongoDB. Da Rust doch noch eher zu den neuen Programmiersprachen gehört, fand sich wenig Dokumentation zur Library, mit welcher die Anbindung gemacht wurde.

Da Ramon schon die Basics von Svelte kannte, musste er nur die Grundkenntnisse nur nochmals kurz repetieren. Diese funktionierte gut mit der *offiziellen Svelte-Dokumentation* [[Link 3]](https://svelte.dev/docs). Ein – zwei Mal kam es zu einem Problem, an dem er anstand, welches er dann mithilfe von Stack Overflow & Co schlussendlich lösen konnte. Zum Teil dauerte die Suche zwar länger, als man sich vielleicht für bekanntere Frameworks wie Angular gewohnt ist, da Svelte immer noch relativ neu ist und noch nicht von so vielen Developern genutzt wird. Trotzdem fand sich die Lösung aber fast immer, manchmal eben in GitHub Issues auf der 2. Seite und nicht auf Stack Overflow.

Nun geht es darum, das zuvor Geplante in die Tat umzusetzen. Hier wurde zwischen Frontend und Backend unterschieden, wobei es vor allem bei der Verbindung der Beiden zu der ein oder anderen Überschneidung führte.

### 5.1 Allgemeines

Beim Klassendiagram war ersichtlich, dass der User jeweils für die erreichten Skills einen Hash erhält, der eine ObjectID und Levels referenziert. Im Unterschied dazu gibt es die markierten Skills, die ebenfalls mit der ObjectID abgespeichert werden. Im Gegenteil zu den erreichten Skills werden dort jedoch keine Levels mitgegeben, da jeweils nur die Skills selbst, also eigentlich die Überkategorien der Subskills markiert werden können. Ein mögliches (befülltes) User-Objekt könnte also folgendermassen aussehen:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 7 Beispiel eines User-Objekts

### 5.2 Frontend

Das Design des Frontends wurde möglichst getreuu den Mockups realisiert, wobei das natürlich nicht immer zu hundert Prozent möglich war. Bei der Farbwahl haben wir durch unseren Theme-Switcher (Dark Mode / Light Mode) zwei verschiedene Varianten, einmal die helle Version und einmal die dunkle. Aus Gründen der besseren Visualität wird in den Bildern hier primär die dunkle Variante verwendet, die helle Variante existiert aber auch.

Als Primärfarbe des Dark Modes haben wir ein dunkles Blau gewählt, das von einem hellen Blau sowie grün und rot für die Knöpfe ergänzt wird.

#### 5.2.1 Homepage / Landing Page

Die Homepage sieht final praktisch 1 zu 1 wie das MockUp dazu aus. Einzig etwas Weiteres wurde ergänzt, dafür aber direkt auf allen Seiten: Der «Theme-Switcher» - das kleine Icon oben rechts, mit welchem man zwischen dem Dark Mode und dem Light Mode wechseln kann. Es gehört zu unseren persönlichen Anforderungen und wurde deshalb entsprechend ergänzt.

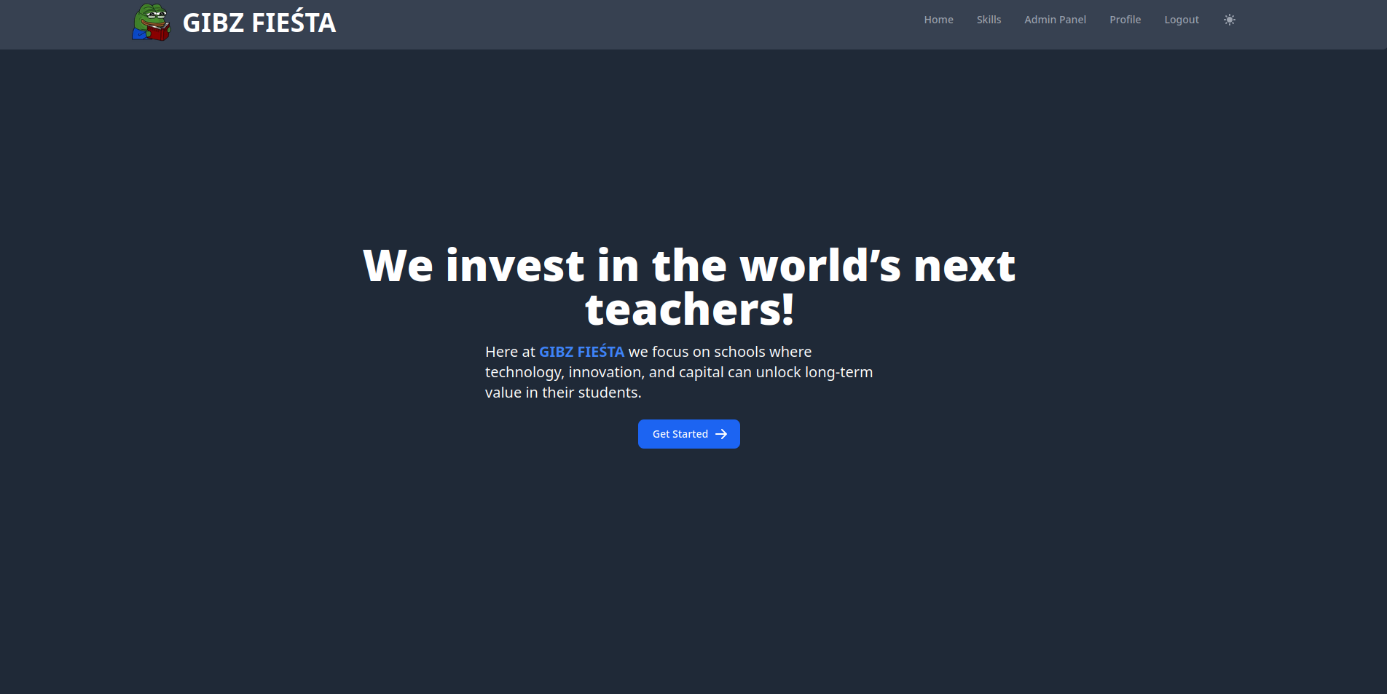


Abbildung 8 Umsetzung Homepage

Vergleich zum Light Mode:

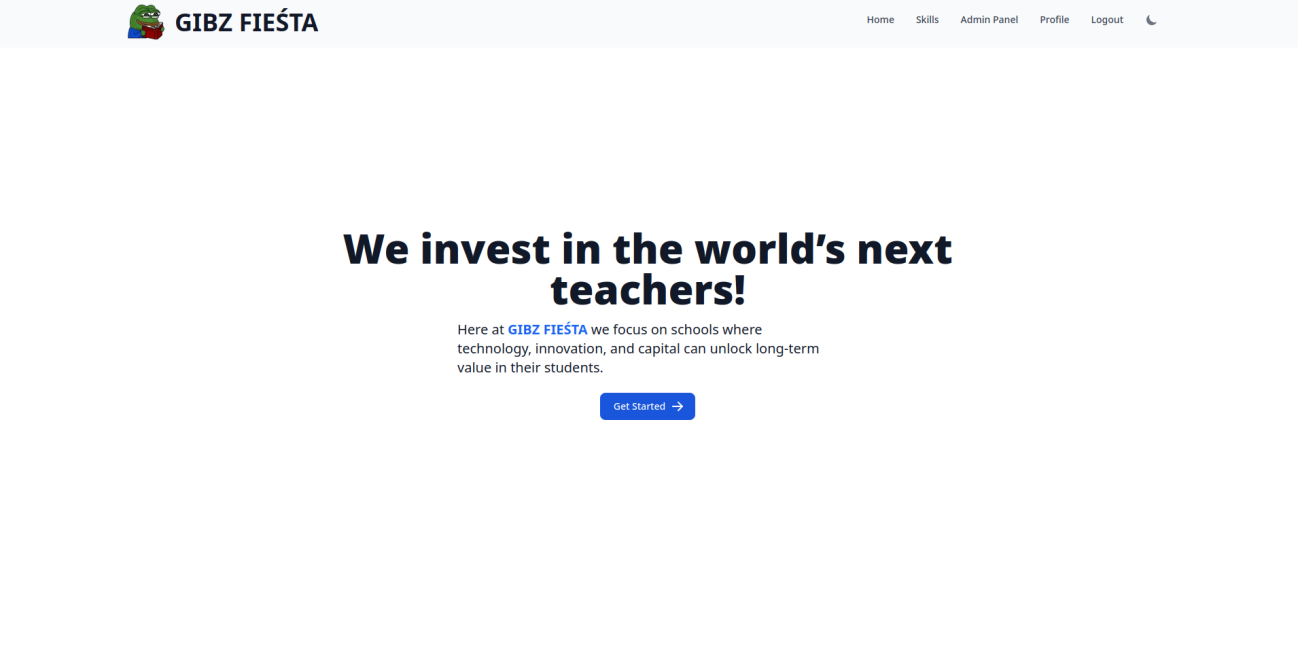


Abbildung 9 Umsetzung Homepage (Light Mode)

#### 5.2.2 Profil

Das Profil wurde ebenfalls fast komplett nach Vorgabe implementiert, wobei es hier durchaus noch Potenzial für das ein oder andere Styling gehabt hätte. Jedoch sind die Angaben klar ersichtlich und es ist auf einen Blick zu erkennen, welche Skillkategorien / Skills bereits abgeschlossen wurden:

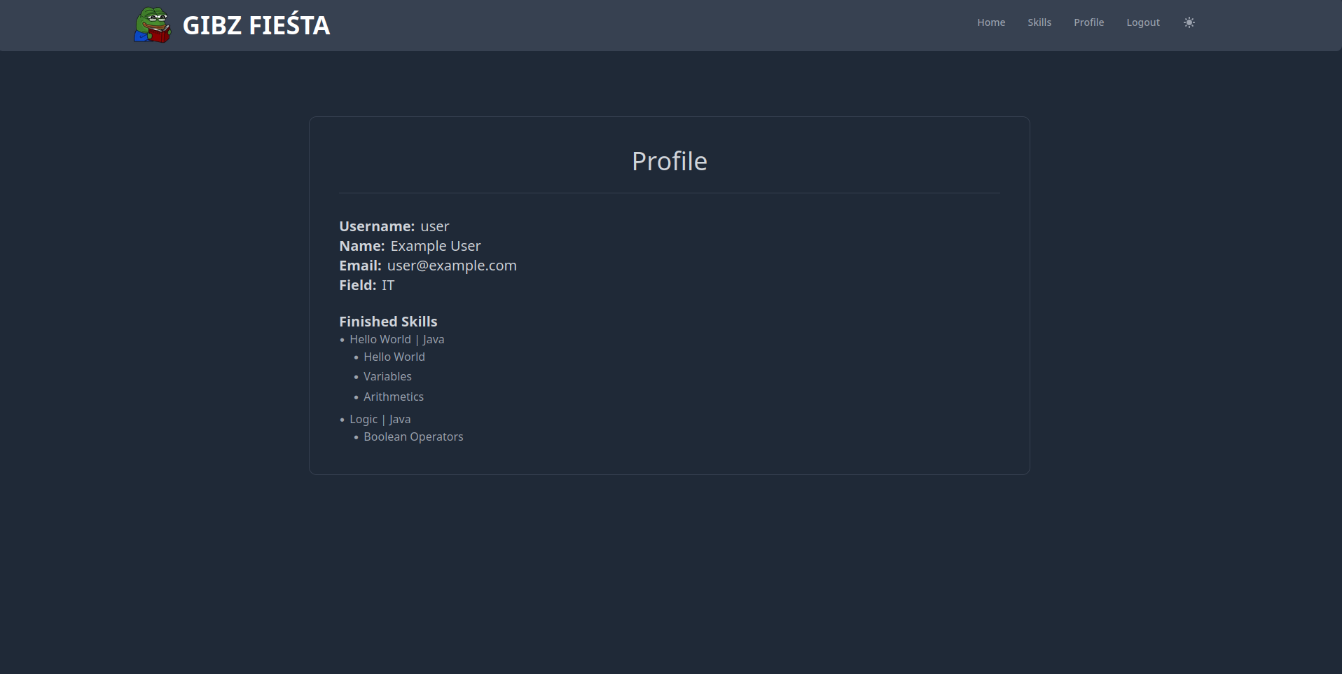


Abbildung 10 Umsetzung Profil Seite

Das Designen und Entwickeln der Profil Seite lief verhältnismässig schnell, wenn man es mit der Skill und Admin Page vergleicht. Durch das Mockup hatte man bereits eine klare Vorstellung und auch die Logik mit dem Darstellen der Daten war bereits auf den beiden Seiten vorhanden und musste nur noch angepasst werden.

Die Profil Seite dient dazu, den eigenen Fortschritt und Daten auf einen Blick zu sehen. Ausserdem ermöglicht es dem User den Fortschritt über den Link, der in diesem format ist: /profile/<username>. Dieser kann auch **ohne Authentifizierung** abgerufen werden. Dementsprechend musste man im Backend eine eigene Route erstellen, damit der Auth Token nicht versehentlich mitgeschickt wird. Das wäre ein kritischer Sicherheitsfehler und würde es jedem Besucher erlauben, sich als den User einzuloggen, dem das Profil gehört

#### 5.2.3 Skill-Seite

Neben dem offensichtlichen Unterschied der Farbwahl wurde der Skill-Übersicht noch eine Suchfunktion hinzugefügt, um sich auch bei etwas mehr Skills noch immer zurechtzufinden. Zudem sind die Skillkategorien sowie einzelne Skills ein und ausklappbar, was ebenfalls einen besseren Überblick verschafft. Die Kategorien verfügen zudem entsprechend dem MockUp jeweils über eine «Mark»-Funktion, um Kompetenzen, die der User besonders hervorheben möchte, markieren zu können. Die einzelnen Skills können mit Klick auf den grünen «Complete» Button als erreicht abgestempelt werden. Achtung: Die Markierung mit «Mark» gilt nur visuell und hat nichts mit dem Erreichsstatus des Skills zu tun.

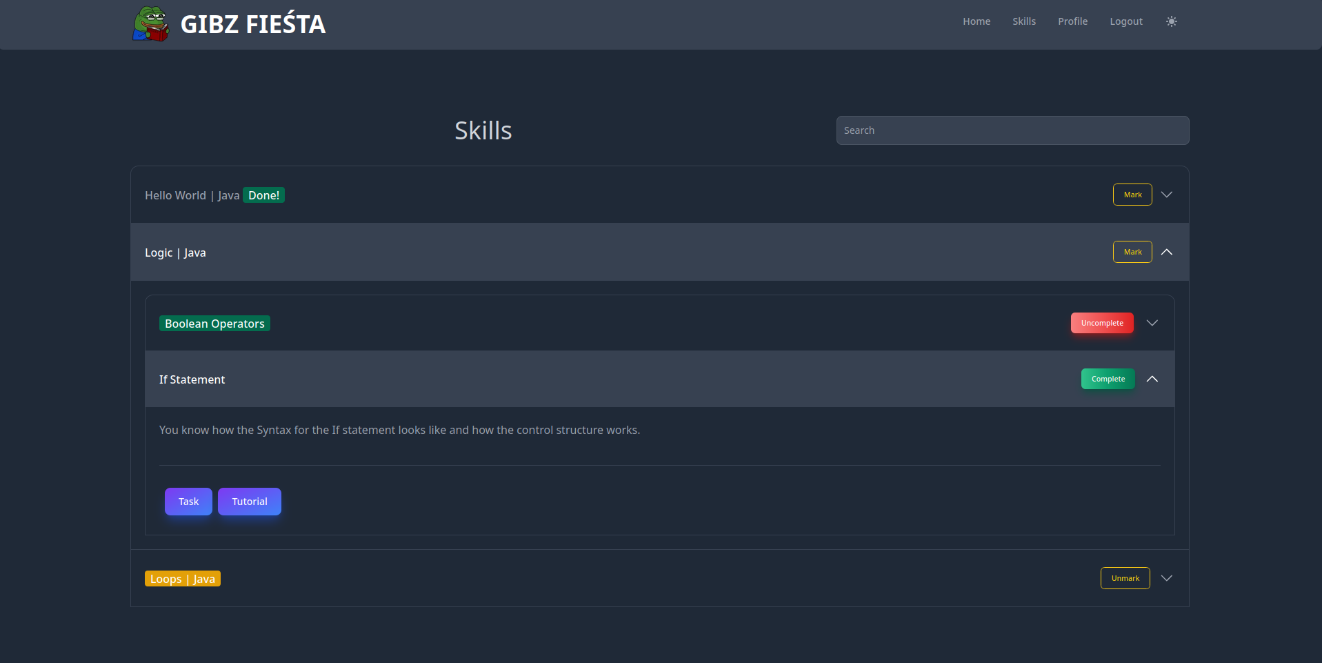


Abbildung 11 Umsetzung Skillseite

Vergleich zum Light Mode:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 12 Abbildung Skillseite (Light Mode)

#### 5.2.4 Admin-Panel

Das Admin Panel beanspruchte mit am meisten Zeit für das Design, da es aus den meisten Elementen besteht. Es kann grundsätzlich in zwei Teile geteilt werden: User Management und Skills Management. Bei der Nutzerverwaltung kann wahlweise ein neuer Nutzer (mit oder ohne Admin-Berechtigungen) erstellt oder aber die Liste aller Nutzer aufgerufen werden. Bei der Kompetenzverwaltung sind alle vorhandenen Skills aufgelistet und können je nach Wahl bearbeitet oder gelöscht werden. Zudem kann ebenfalls ein neuer Skill angelegt werden. Diese Ansicht ist jeweils nur für Administratoren ersichtlich.

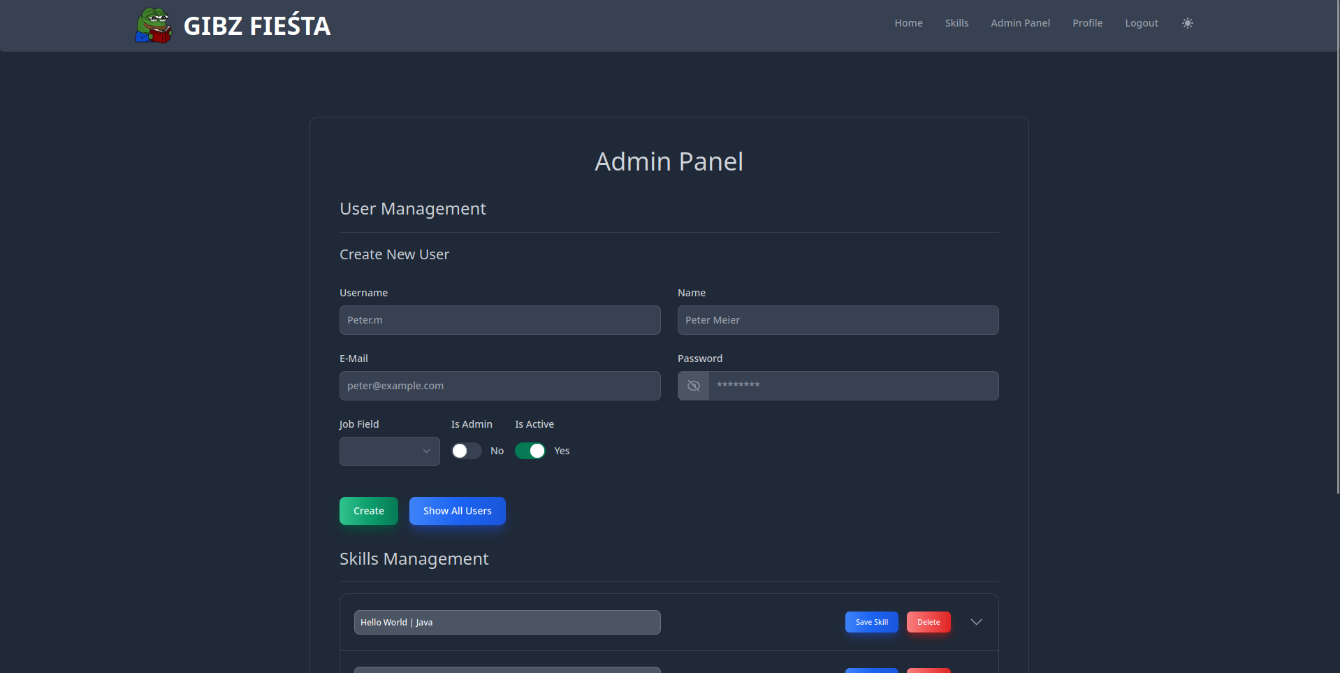


Abbildung 13 Umsetzung Admin Panel

Ein Bild, das Text, schwarz, Monitor, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung 14 Umsetzung Admin Panel Teil 2

Vergleich zum Light Mode:

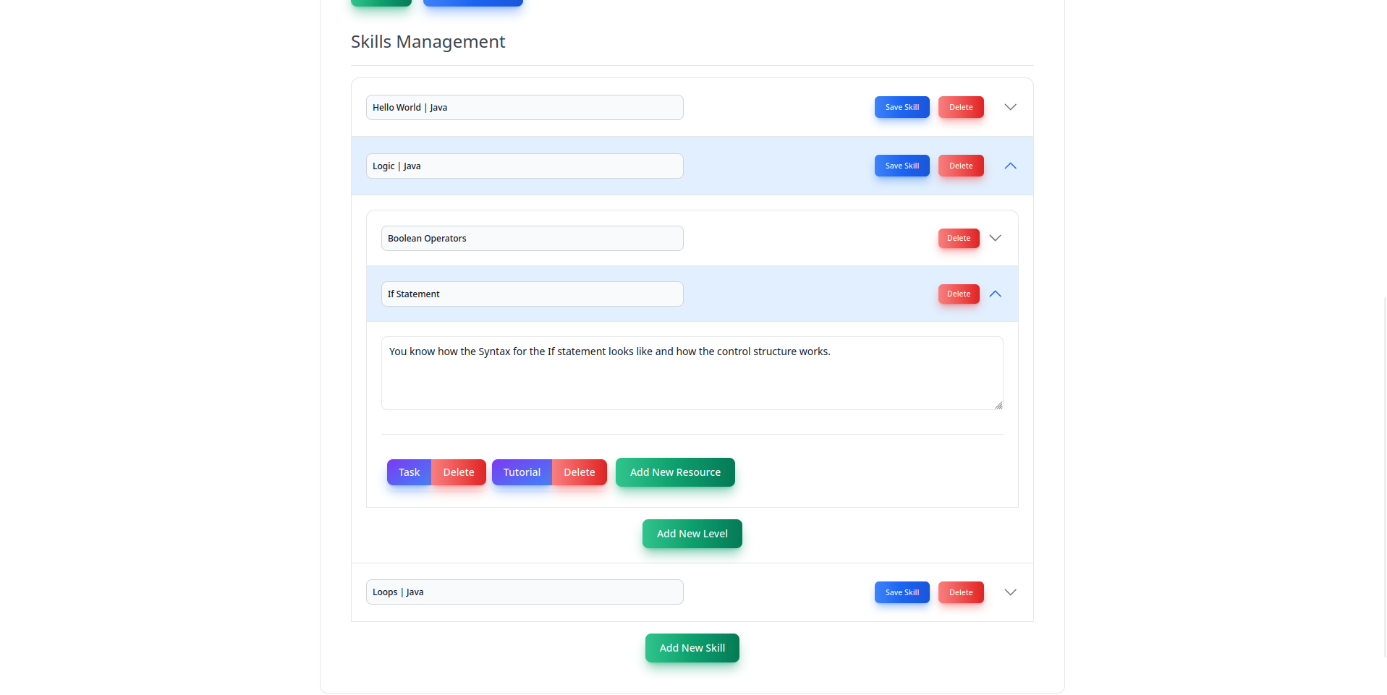


Abbildung 15 Umsetzung Admin Panel Teil 2 (Light Mode)

Beim Klicken auf «Show All Users» erscheint folgendes Fenster mit allen bestehenden Nutzern aufgelistet



Abbildung 16 Umsetzung User List

Sollte der Admin dann auf den Edit-Button (Icon eines Stifts) drücken, kann er in folgendem Fenster den User bearbeiten:

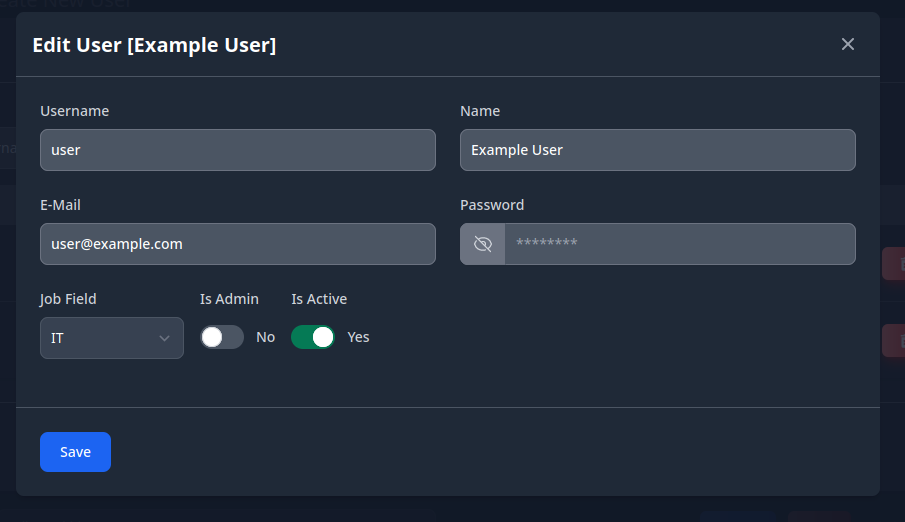


Abbildung 17 Umsetzung Userverwaltung

Wie bei der ganzen Seite mussten wir hier sehr aufpassen, dass die Seite responsive ist, da das eine unserer Zusatzanforderungen ist. Hier war das allerdings etwas schwieriger, da so viele Elemente dargestellt werden müssen. Das Skill Accordion war relativ einfach responsive darzustellen, da es schon Mobile freundlich out of the box ist. Das User Create Formular allerdings war mühsamer wegen den beiden Toggle Buttons, denn die verschiebten sich immer sehr. Die Lösung war dann, das Ganze in eine Flexbox zu packen und mit weiteren Flexboxen das Layout zu gestalten.

Das Elend ist aber noch nicht vorbei, denn die User Liste fehlt noch. Hier entschieden wir uns die Liste hinter einem Pop-Up zu verstecken, da sonst die ganze Seite noch voller aussieht, was den User möglicherweise überfordern könnte. Die Schwierigkeit hierbei war allerdings die Darstellung auf Smartphones. Es hat zu wenig Platz für eine Tabelle. Eine Horizontale Scrollbar behebt das Problem aber.

#### 5.2.5 Svelte Allgemein

Text

Description automatically generated

Abbildung 18 Implementation der User-Suche mit Svelte

Im Bild oben sieht man für die User Suche für die Admin Page. Das ist typisch Svelte gelöst, denn auffallend ist die Dollar $: Syntax. Mit dieser Syntax kann man einen Codeblock definieren, der immer ausgeführt wird, sobald eine Variabel darin den Wert ändert. Das eignet sich perfekt für eine Suchfunktion, denn jedes Mal wenn der User ein Zeichen Tippt, wird sofort die neue, gefilterte liste angezeigt. Das Reactive Statement ist eigentlich ein Label, also etwas, was man nicht verwenden sollte, da es schnell *Spagetti* *Code* verursacht.

Das reactive Statement ist aber nicht alles, was es für die Suchfunktion braucht. Die Variable *userSearchTerm* muss auf das Input Feld gebinded werden und über den Array muss im HTML drüber gelooped werden, normalerweise ginge das nicht. In Svelte sieht das ganze etwa so aus.

Text

Description automatically generated

Abbildung 19 HTML der User-Suche mit Svelte

Der Inhalt des {#each} Blocks wird für jedes Element gerendert und macht so dynamisches HTML sehr einfach zu erstellen.

Im Bild sieht man auch dass die beiden Buttons das onClick Event gebinded haben. Wenn also der erste Button, der User Edit Button, gedrückt wird, wird die *OpenEditUserModal* Funktion aufgerufen

### 5.3 Backend

Für die Umsetzung unserer Backend – Applikation haben wir uns wie schon erwähnt für Rust, eine low-level Programmiersprache entschieden. Der Webserver wurde mit Hilfe von [rocket.rs](https://rocket.rs/), einem neuen Framework (vergleichbar mit express.js) realisiert. MongoDB ist der Datenbankdienst unserer Wahl, da er sich aus vorherigen Erfahrungen als einfach und simpel zu bedienen herausstellt.

Da wir den normalen Fehlercode langweilig fanden, entschieden wir uns für sämtliche Error Responses den Code [418](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/418) (I’m a Teapot) zu verwenden.

#### 5.3.1 Ordnerstruktur

A picture containing graphical user interface

Description automatically generatedUm das Projekt so erweiterbar wie möglich zu machen, entschied ich mich für eine modulare Ordnerstruktur, welche mit MVC (Model – View – Controller) vergleichbar ist.

##### 5.3.1.1 API

Dieser Ordner beinhaltet sämtliche Endpoints die das Backend zur Verfügung stellt, wie zum Beispiel «/user/create», mit welcher ein neuer User in der Datenbank erstellt werden kann.

##### 5.3.1.2 Data

Im «data» Ordner befinden sich alle Methoden, die auf die Datenbank zugreifen. Die meisten dieser Methoden folgen dem CRUD-Schema, wobei es bei Einzelfällen (wie z.B. das Markieren eines Skills) eine eigene Methode benötigte.

##### 5.3.1.3 Helpers

Unter «helpers» verstehen sich Funktionen, die redundanten Code vermindern und einem auf gut Deutsch «Das Leben einfacher machen». Hier findet man zum Beispiel Funktionen wie «grandmas\_bakery.rs» ein Cookie Handler, welcher Cookies erstellt sowie die Daten aus einem Cookie rauslesen kann.

##### 5.3.1.4 Models

Der «models» Ordner enthält Datenmodelle (siehe [Klassendiagramme](#_3.2_Klassendiagramm)), mit welchen die Datenbank gefüllt wird. Im aktuellen Projektumfang sind dies «Skill» & «User», welche bei Bedarf natürlich erweitert werden können / dürfen.

##### 5.3.1.5 Tests

Im «tests» Ordner befinden sich die Unit Tests des Projekts. Diese sind im Moment bei einer ~61% Deckungsrate.

#### 5.3.2 Connector

Das Herzstück des Backends, der «Connector» (im «data» Ordner), ist vergleichbar mit dem Controller aus dem MVC – Modell. Er beinhaltet alle Methoden, die die Daten aus der Datenbank lesen, schreiben, modifizieren und löschen. Der Controller ist in folgende Kategorien unterteilt:

* «User Related Methods»: Alle Methoden, die die User Datenbank modifizieren.
* «Skill Related Methods»: Alle Methoden, die die Skill Datenbank modifizieren.
* «Auth Methods»: Methoden, die für die Authentifizierung des Users verantwortlich sind.

##### 5.3.2.1 User Related Methods

Die meisten Methoden in dieser Kategorie behandeln den Read, Create & Delete Teil, wie zum Beispiel «create\_user()», welche es dem Systemadmin ermöglicht, neue Benutzer zu erstellen. Da wir uns hier auf dem «Data Level» befinden, sind die Return Values meistens Booleans oder «Results» von MongoDB – Aktionen. Prozesse wie Authentifizierung o.Ä. befinden sich im «api» Ordner.

Text

Description automatically generatedSchwierigkeiten gab es hier ein paar. Die meisten haben mit dem «Update» Prozess von Daten zu tun, da dieser sehr schlecht dokumentiert und mühsam umzusetzen ist. Ramon ist jedoch auf die gute Idee gekommen, anstelle des Bearbeitens eines Objektes das Ganze einfach zu löschen – und dann wieder mit den aktuellen Daten einfügen; Jedoch die «ObjectId» (vergleichbar mit dem Primary Key aus SQL – Datenbanken) zu behalten. Durch das sind Objekte immer noch unter dem gleichen Key zu finden, egal was mit den restlichen Daten passiert.

Die «Mark» & «Complete» Methoden waren auch nicht ganz so einfach. Wir haben uns darauf geeinigt, das eine Methoden beide Use – Cases behandeln sollen, also ein Skill markieren / completen sowie entmarkieren / uncompleten. Dies musste in mehreren Schritten gemacht werden:

1. Den «completed\_skills» resp. «marked\_skills» array des Users auslesen
2. Überprüfen, ob der User diesen Skill (via ObjectId) markiert / vervollständigt hat
3. Falls dies so ist, den Skill vom Array entfernen bzw. hinzufügen
4. Das User Objekt speichern und einen gültigen Returnwert zurückgeben

Text

Description automatically generatedDer «skills\_vec» ist in diesem Falle der Array, der ausgelesen wird. Dies kann entweder der «completed\_skills» oder «marked\_skills» Array des Users sein.

Nach dem Auslesen wird über den Array iteriert und ein Skill gesucht, der die gleiche ObjectId hat wie den, den der User gerade versucht zu markieren / entmarkieren. Falls dies der Fall ist, wird via $pull der Skill aus dem Array entfernt resp. Mit $push hinzugefügt.

##### 5.3.2.2 Skill Related Methods

Zu den SRM zählen alle Methoden, die die «Skill» Objekte manipulieren. Hier gab es keine komplizierten Methoden, da es sich nur um Create, Read und Delete handelt. Die Updatemethode wurde wie beim Userobjekt via Delete & Insert gelöst.

##### 5.3.2.3 Auth Methods

Text

Description automatically generatedDiese Kategorie beinhaltet zwei Methoden: Eine für die Authentifizierung des «normalen» Users, und eine für Admins.

Die beiden Methoden funktionieren von der Logik her genau gleich, jedoch mit einem kleinen Unterschied. Bei «verify\_admin()» wird in der Datenbank nach einem User mit dem auth\_token & einer «role» von 1, also Admin, gesucht.

Bei der «normalen» Auth-Verifikation wird nur nach einem Token gesucht. Dies ist sicher, da nur ein Admin Benutzer erstellen kann, wodurch keine «Fremde» Einsicht haben können. (Ausser ein Login kommt ans Licht, dies wird in der Fachsprache «Osi Layer 8 Error» genannt. ☺)

#### Text Description automatically generated5.3.3 Endpoints

Die Endpoints, welche sich im «api» Ordner befinden, sind in die Files «skill.rs» & «user.rs» unterteilt und folgen alle einem ähnlichen Aufbau:

Sie beinhalten einen Auth-Checker (User / Admin), welcher dafür sorgt, dass nur zugelassene Personen auf den Endpoint zugreifen können. Danach folgt eine CRUD – Abfrage (oder Mark/Complete), welche dann bei Erfolg als Response einen Wert (Json oder Statuscode) zurückschickt.

Mit dem ‘#’ auf der ersten Zeile wird definiert, um welchen Requesttyp es sich handelt, um dann eine angemessene Antwort zu geben.

#### 5.3.4 Helpers

Folgende Helperfunktionen sind im aktuellen Projekt vorhanden:

* Grandmas Bakery (Cookie Parser & Creator)
* endecr ( Encrypt & Decrypt, gebraucht für das [hashen](#_5.4.1.1_Hashing) von Passwörtern)
* token (Generierung von neuen auth\_tokens)

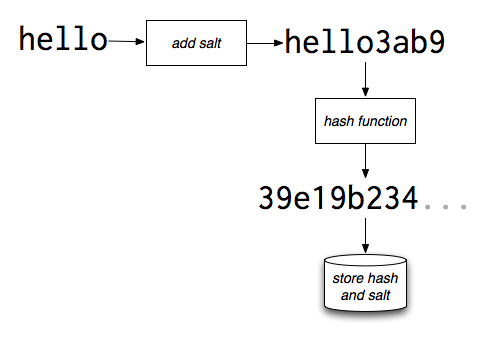
### 5.4 Sicherheit & Rollensystem

Die Sicherheit einer Applikation ist Das A & O in der heutigen Zeit. Um die Sicherheit unserer Applikation zu gewährleisten, haben wir folgende Technologien implementiert:

* Password Hashing
* Rollensystem

#### 5.4.1 Sicherheit & Verschlüsselung

##### 5.4.1.1 Hashing

Für das hashing der Passwörter verwenden wir einen Algorithmus namens «Argon2», einer der sichersten und zuverlässigsten Algorithmen, welcher nicht wie SHA & MD5 bereits gecrackt wurde. Argon2 verwendet ein bekanntes Prinzip beim Hashing, Salt & Pepper.

Dies sind zwei Strings, die vor resp. nach dem eigentlichen Passwort eingefügt werden, um die Sicherheit zu erhöhen. Diese Funktion ist in im «endecr» helper zu finden, bei welcher der Salt-wert über ein .env File gesetzt wird. Durch das muss das Salt nicht in der Datenbank gespeichert werden, was Passwörter auch bei einem Angriff sehr schwer zu entziffern macht.

#### 5.4.2 Rollensystem

Das Rollensystem ist wie folgt aufgebaut:

* 0: Normaler User (Lehrer)
* 1: Admin (Kann Skills & User erfassen, bearbeiten & löschen)
* Gast: kein Account

Um maximale Sicherheit zu gewährleisten, sind sämtliche Routes für Non-User nicht zugänglich, mit Ausnahme der «get\_user\_profile()» Methode, die Profile von bestehenden Usern für Gäste einsichtlich macht.

### 5.4 Verbindung Backend – Frontend

Ursprünglich hatte das Frontend nur mit einem statischen JSON File die Daten dargestellt, um trotzdem Daten zu haben. Das Backend war schliesslich immer noch in Arbeit. Wir haben uns konstant ausgetauscht über die Felder, die das JSON für die Kompetenzen und User Models beinhalten soll. Das hatte den Vorteil, dass das Backend und Frontend abgekoppelt voneinander entwickelt werden konnten. Erst am Schluss wurden die beiden Komponenten zusammengefügt. Das lief ganz einfach, in dem wir die URL fürs Backend von den Testdaten zur URL des Backends änderten.

Zumindest lautete so der Plan. Denn beim tatsächlichen Verbinden merkten wir, dass es doch nicht ganz so einfach war. Als Testdaten hatten wir zwei einfache Files, das tatsächliche Backend allerdings hatte mehr als 12 Routes. Wir mussten also im Frontend die API Calls mehr oder weniger neu schreiben und aufteilen. Das ist aber noch gegangen im Vergleich zum Problem mit den Cookies.

#### 5.4.1 Cookies, die nicht gespeichert werden

Im Arbeitsjournal wurden die Cookies schon ein wenig angeschnitten. Im Grunde ging es darum, dass wenn man die Seite neu geladen hat mit F5, man ausgeloggt wurde. Uns war klar, dass es etwas mit den Cookies zu tun hatte, denn dort sollte eigentlich der Authentifizierungs Token gespeichert werden. Wenn man aber in die Dev-Tools des Browsers ging, sah man dort weder ein Cookie, noch ein Token.

Das Einzige, was man sah, war beim Request in den Dev-Tools, das dort "Request Cookies" gesetzt und sogar gespeichert worden sind. Diese kann man aber nicht editieren und sogar auch nicht löschen. Das hat uns ein wenig verwirrt. Wir haben daraufhin viele kleine Dinge versucht zu ändern, ohne wirklich zu wissen, woran das lag, dass die Cookies nicht im normalen Ort, also dort wo auch LocalStorage liegt, gespeichert wird. Zwischenzeitlich dachten wir auch, dass das Problem im Backend liegt und haben dort bei der Response versucht die Cookie Properties zu ändern. Ohne Erfolg, unser Problem zu lösen leider.

Da wir nun schon viel Zeit daran verschwendet haben, ignorierten wir vorerst das Cookie Problem und machten uns weiter daran, die restlichen Routes ins Frontend zu übernehmen. Irgendwann war dann die Auth Route dran. Diese wird bei jedem Page Load aufgerufen und überprüft den Authentifizierungs Token, der in den Cookies gespeichert ist, denn der kann vom User modifiziert werden. Da für die Implementation dieser Route allerdings die Clientseitige Authentifizierung der gerade aufgerufenen Seite überarbeitet werden musste, dauerte das auch ein wenig länger.

Nur kurz noch erwähnt, was mit "Clientseitige Authentifizierung" gemeint ist. Darunter verstehen wir, die Überprüfung, ob ein User auch wirklich die Berechtigungen dazu hat, eine bestimmte Seite, also z.B /skill oder /admin aufzurufen. Logischerweise sollten normale User keinen Zugriff auf das Admin Panel bekommen. Wenn sie es trotzdem Aufrufen wollen werden sie zurückgeschickt. Mit dem Rework dieses Systems wurde auf magische Weise auch das Problem mit den Cookies gelöst. Uns war zu dem Zeitpunkt aber schon bewusst, dass sie tatsächlich gespeichert werden, wir konnten aber nicht rausfinden, wieso wir nach einem Refresh der Seite ausgeloggt waren. (edited)

## 6 Kontrollieren

*Zur Kontrolle wird in dieser Phase die Applikation getestet*

Wir haben uns auch selbst vorgenommen, ein automatisches Test-System zu implementieren, weshalb hier nun eine Auflistung von diesen Tests folgt. Um das Ganze nicht nur einer Maschine zu überlassen, haben wir ausserdem noch manuelle White-Box Tests durchgeführt. Was hier noch fehlen würde, sind Black-Box Tests. Diese haben wir aus dem Grund weggelassen, da es bei diesen darum geht, das Programm von einer dem Projekt unvertrauten Person testen zu lassen. Da wir drei aber alle sehr involviert sind, konnte das keiner von uns übernehmen.

### 6.1 Unit Tests

Im Code beziehungsweise spezifischer im Unterordner *Fiesta-Server/src/tests* befinden sich alle Unit Tests, die für diese Applikation erstellt wurden. Es folgen nun einige Screenshots dieser Tests, spezifisch zum User-Objekt:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Für das Skill Objekt wurden ebenfalls Tests umgesetzt:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### 6.2 Whitebox Tests

Als Testuser hatten wir im Vorhinein den Admin Benutzer „admin“ gespeichert mit dem Passwort „123456“. Dieser wurde auch schon beim Entwickeln benutzt, nun aber via das MongoDB Webinterface zurückgesetzt, sodass keine Skills mehr abgeschlossen oder markiert waren. Dann gab es noch einen 2. Normalen, also ein Benutzer ohne Administratorrechte mit dem Usernamen „user“ und Passwort „123456“, ebenfalls zurückgesetzt. Als Skills haben wir die 3 von den Screenshots beim Realisieren Teil genommen.

Daten des Systems, auf dem getestet wurde:

Ryzen 7 3700x

32GB RAM

RTX 3080

4TB SSD

Bildschirme: 1x FHD, 1x UWQHD

|  |  |
| --- | --- |
| Frontend | 1 – Login mit korrekten Daten |
| Beschreibung | Mit richtigem Username und Passwort via Frontend anmelden |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „admin“ eingeben 4. Als Passwort „123456“ eingeben 5. Login erfolgreich, man landet auf der Skills Seite. |
| Erwartetes Resultat | Login erfolgreich, man landet auf der Skills Seite. |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

|  |  |
| --- | --- |
| Frontend | 2 – Login mit inkorrekten Daten |
| Beschreibung | Mit falschem Username und Passwort via Frontend anmelden |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „Admin“ eingeben 4. Als Passwort „123456“ eingeben 5. Login nicht erfolgreich, man bleibt auf der Seite und es wird eine Fehlermeldung bzgl. Falschem Username/Passwort angezeigt. |
| Erwartetes Resultat | Login nicht erfolgreich, man bleibt auf der Seite und es wird eine Fehlermeldung bzgl. falschem Username/Passwort angezeigt. |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

|  |  |
| --- | --- |
| Frontend | 3 – Skill Erstellen |
| Beschreibung | Über das Admin Panel ein neuer, leerer Skill erstellen. |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „admin“ eingeben 4. Als Passwort „123456“ eingeben 5. Login erfolgreich, man landet auf der Skills Seite. 6. Admin Panel öffnen 7. Auf „Add New Skill“ klicken 8. Daneben auf „save skill“ klicken 9. Seite neu laden |
| Erwartetes Resultat | Ein neuer leerer Skill wird unten am Accordion hinzugefügt und mittels Page Reload überprüft |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

|  |  |
| --- | --- |
| Frontend | 4 – Skill Markieren und Abhaken |
| Beschreibung | Als normaler User (kein Admin), Skills auf der /skills seite markieren und löschen |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „user“ eingeben 4. Als Passwort „123456“ eingeben 5. Login erfolgreich, man landet auf der Skills Seite. 6. Den ersten Skill Markieren 7. Den ersten Skill aufklappen 8. Das 2. Level abschliessen. 9. Seite neu laden |
| Erwartetes Resultat | Skill ist nach dem neu laden immer noch markiert und das 2. Level ist als abgeschlossen markiert. |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

|  |  |
| --- | --- |
| Frontend | 5 – User Löschen |
| Beschreibung | Über das Admin Panel der User „user“ löschen. |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „admin“ eingeben 4. Als Passwort „123456“ eingeben 5. Login erfolgreich, man landet auf der Skills Seite. 6. Admin Panel öffnen 7. Auf „Show All Users“ klicken 8. Den User mit dem Username „user“ finden und ganz rechts über den Button mit dem Mülleimer löschen. |
| Erwartetes Resultat | User verschwindet von der Liste |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

|  |  |
| --- | --- |
| Frontend | 5 – User Löschen |
| Beschreibung | Über das Admin Panel der User „user“ löschen. |
| Durchführung | 1. Seite öffnen 2. Login Page öffnen 3. Als Username „admin“ eingeben 4. Als Passwort „123456“ eingeben 5. Login erfolgreich, man landet auf der Skills Seite. 6. Admin Panel öffnen 7. Auf „Show All Users“ klicken 8. Den user mit dem Username „user“ finden und ganz rechts über den Button mit dem Mülleimer löschen. |
| Erwartetes Resultat | User verschwindet von der Liste |
| Erhaltenes Resultat | Erwartetes Resultat wurde bestätigt. |
| Status | Bestanden |
| Datum | 16.12.2022 |
| Getestet von | Ramon Camenzind |

## 7 Auswerten

*In der Auswertung wird auf das Projekt als Ganzes zurückgeblickt und ausgewertet*

### 7.2 Probleme

#### 7.2.1 Cookies

Es wäre wohl eine Untertreibung zu sagen, dass die Einbindung von Cookies keine leichte Aufgabe war. Das Setzen des Cookies war vergleichsweise simpel, problematisch wurde es jedoch bei einem Reload der Website. Denn dann war es anfangs so, dass das Cookie verworfen wurde und der Benutzer nach jedem Neuladen wieder abgemeldet war. Dies war nicht zuletzt aufgrund der Blockierung vom Set-Cookie Header seitens der Browser:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 20 Auszug aus den MDN Web Docs

Eine Übergangslösung bestand darin, das Cookie manuell für 7 Tage lang zu setzen. Da das jedoch sehr anfällig für Angriffe ist und wir das Sicherheitsrisiko möglichst minimieren wollen, reicht dies natürlich nicht für die finale Applikation.

Schlussendlich funktionierte es dann mit dem Ergänzen des folgenden kleinen Teils:

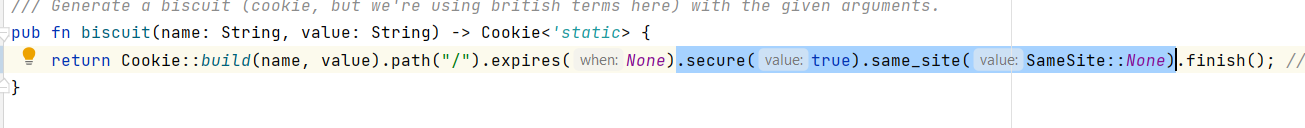


Abbildung 21 Funktion zur Cookiegenerierung

Auch wenn es vermutlich noch bessere Lösungsansätze für diese Problem gäbe, ist dies auf jeden Fall schon einmal eine sicherere Variante als das manuelle Setzen für 7 Tage. Zudem funktioniert Sie auch, wenn die Seite neu geladen wird.

#### 7.2.2 Aktualisierung des UI bei Änderungen

Bei der Implementation des Frontends kam es kurzzeitig zum Problem, dass das User Interface beim Abändern des Codes nicht mitaktualisiert wurde und diese Änderungen nicht visuell sichtbar waren, obwohl sie eigentlich gepusht wurden. Spezifischer geschah das beim Hinzufügen von Elementen zu einem Array, was in diesem Fall eine doch sehr wichtige Funktionalität ist. *array.push(element)* war nicht ausreichend, damit das Array auch im UI aktualisiert wurde. Nach einiger Recherche ist dann herausgekommen, dass das Array nach der Änderung zuerst mit array = array aktualisiert werden musste, bevor es dann auch im UI wieder korrekt angezeigt wurde. Auf diese Lösung sind wir nicht alleine gekommen sondern mithilfe von ein *Github Issues-Kommentar* [[Link 4]](https://github.com/sveltejs/svelte/issues/4285#issuecomment-575984317).

#### 7.2.3 Accordions auf Admin Page

Ein weiteres Problem war, dass bei jedem Seitenwechsel die Accordions noch einen kurzen Moment sichtbar waren und der Inhalt der neuen Seite über den Accordions erschien. Wir nahmen an, dass das eine Limitation von SvelteKit ist. Das Problem haben wir allerdings ganz einfach gelöst, indem das div, das den ganzen Inhalt der Seite umfasst, unsichtbar gemacht wird, wenn die Unterseite gewechselt wird. Die Implementation davon sieht man unten.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### 7.3 Verbesserungen

André

Eine Applikation ist nie perfekt und es kann immer noch mehr verbessert werden. Trotzdem würde ich behaupten, dass uns die Umsetzung gut gelungen ist.

Was ich in Zukunft vermutlich anders machen würde, ist die Aufteilung. Bei uns gab es mehr oder weniger eine Trennung zwischen FE, BE und Dokumentation. Ich denke es wäre für ein nächstes Projekt schlauer, wenn alle Teammitglieder überall ein wenig mithelfen. Zudem ist mir zum UI noch etwas in den Sinn gekommen, dass wir bei der Implementierung missachtet haben: Unsere Buttons sind grösstenteils Rot und Grün, was für die Barrierefreiheit nicht hilfreich ist, da es Nutzer mit Rot-Grün-Schwäche benachteiligt.

Etwas vermeintlich kleines, was jedoch einen grossen Unterschied macht, sind Fehlermeldungen. Uns ist leider noch entgangen, diese dem Benutzer ebenfalls anzuzeigen. In der jetzigen Version ist es so, dass nur beim Login eine Fehlermeldung angezeigt wird, wenn etwas falsch gemacht wurde.

Ramon

Shape

Description automatically generatedGrundsätzlich bin ich sehr zufrieden mit dem Endprodukt. Während der Entwicklung habe ich schon einige dinge verbessert, die mich gestört haben, allerdings nicht alles. Zum einen weil ich nicht genau weiss wie und zum anderen weil das keine wirkliche Priorität hatte.

Abbildung 22: SVG Fluid Shape

Ein so ein Issue wäre zum Beispiel, wenn man als unauthentifizierter User aufs Admin Panel will, lädt es zuerst immer noch eine Sekunde und der Redirect tritt erst danach ein. Es wäre gut, wenn der Redirect so schnell kommt, dass die Seite gar nicht erst lädt.

Ein anderes Problem ist, dass die Landing Page sehr leer aussieht. Ich hatte vor noch SVG „bubbles“ (siehe Bild) in den Hintergrund zu packen, es dann aber mehr oder weniger vergessen, muss ich gestehen.

### 7.4 Fazit

André

Alles in allem war das Projekt in meinen Augen ein voller Erfolg. Nicht nur konnten wir uns in eine für uns neue Programmiersprache einarbeiten, sondern haben wir auch schon erste wichtige Erfahrungen mit einem IPA-ähnlichen Projekt gemacht. Zudem ist das Produkt eine funktionierende Webapplikation, die als solche mit nur minimalen Anpassungen eingesetzt werden könnte. Es gab natürlich nicht nur Lichtblicke, aber komplett blockierende Probleme hatten wir zum Glück keine. Ich habe ausserdem gemerkt, dass Rust nicht unbedingt meine Lieblingssprache ist, da Sie eher auf das Manipulieren von Bytes o.Ä. abzielt und ich weniger in diesem Bereich unterwegs bin. Svelte war dagegen eine willkommene Alternative zu meiner bisherigen Methode von normalem CSS ohne grossartige Frameworks.

Ramon

Ich fand das Projekt lief für mich Persönlich sehr gut, zwischendurch gab es ein paar Komplikationen, die wir angesprochen haben, aber im grossen Ganzen war es ein Erfolg. Ich konnte mich mehr in SvelteKit vertiefen und werde mit Sicherheit auch wieder dieses Framework für mein nächstes Webprojekt verwenden. Was neu dazugekommen ist, ist TailwindCSS. Ich wusste, dass das CSS Framework sehr beliebt bei Entwicklern ist aber ich wusste noch nicht wieso. Nun ja, jetzt weiss ich es, es macht das Design sehr viel einfacher. Allerdings bin ich mir nicht sicher ob ich es auch in Zukunft verwenden werde, denn ich bin der Meinung, dass alle Webseiten immer ähnlicher aussehen und da hilft es nicht, wenn alle dieselbe Design Grundlage verwenden.

Die Realisierungsphase hat mir definitiv auch viel mehr Spass gemacht als die Planungsphase. Ich finde Diagramme und Mockups erstellen nicht gerade die spannendste Aufgabe. Zugegeben aber, die Design Mockups haben sogar ein wenig geholfen bei der Realisierung, auch wenn ich eigentlich schon seit Anfang an das Design der Webseite im Kopf hatte.

Timo

Meiner Meinung nach verlief das Projekt besser, als ich am Anfang erwartet habe. Da Rust für mich eine komplett neue Sprache war wusste ich nicht genau, auf was ich mich einliess. Jedoch waren diese Sorgen umsonst, da es sich bei Rust um eine (sehr) angenehme Sprache handelt. Die VSCode Extension «rust-analyzer» mit dem integrierten «borrow checker» machen das Entwickeln und Debuggen um einiges leichter als man es sich von anderen Low-Level Sprachen (wie z.B. C) gewohnt ist.

Auch durfte ich viel im Bereich Planning bzw. Softwarearchitektur dazu lernen. Das Designen von Klassen und Strukturen durfte ich bisher nur selten in Modulen, da ich mich normalerweise eher mit dem Frontend beschäftigte. Auch das Datenhandling sowie die «intelligente» Aufteilung von Routes bzw. Endpoints war mir neu, jedoch machte es mir enorm Spass, auch mal an der Logik herumzubasteln.

## 8 Anhang

### Linkverzeichnis

* Link 1 – *Offizielle Rust Website* - <https://www.rust-lang.org/learn>
* Link 2 – *Offizielle Rust Dokumentation* - <https://doc.rust-lang.org/stable/>
* Link 3 – *Offizielle Svelte Dokumentation* - <https://svelte.dev/docs>
* Link 4 – *Github Issues Kommentar -* <https://github.com/sveltejs/svelte/issues/4285#issuecomment-575984317>
* Link 5 – *Offizielle SvelteKit Dokumentation -* <https://kit.svelte.dev/docs/introduction>

Honorable Mention, da bestimmt noch der ein oder andere Link vergessen wurde: <http://stackoverflow.com/>

### 8.2 Abbildungsverzeichnis

[Abbildung 1 Mockup Homepage 13](file:///C:\Users\Andrew\Documents\GIBZ\M326%20oder%20so\gibz-fiesta\Documentation\M326DokumentationGIBZFIESTA.docx#_Toc122122203)

[Abbildung 2 Mockup Profil 13](file:///C:\Users\Andrew\Documents\GIBZ\M326%20oder%20so\gibz-fiesta\Documentation\M326DokumentationGIBZFIESTA.docx#_Toc122122204)

[Abbildung 3 Mockup Skills Übersicht 13](file:///C:\Users\Andrew\Documents\GIBZ\M326%20oder%20so\gibz-fiesta\Documentation\M326DokumentationGIBZFIESTA.docx#_Toc122122205)

[Abbildung 4 Mockup Admin Panel 14](file:///C:\Users\Andrew\Documents\GIBZ\M326%20oder%20so\gibz-fiesta\Documentation\M326DokumentationGIBZFIESTA.docx#_Toc122122206)

[Abbildung 5 Klassendiagramm 14](#_Toc122122207)

[Abbildung 6 Modell des Gesamtsystems 15](#_Toc122122208)

[Abbildung 7 Beispiel eines User-Objekts 17](#_Toc122122209)

[Abbildung 8 Umsetzung Homepage 18](#_Toc122122210)

[Abbildung 9 Umsetzung Homepage (Light Mode) 18](#_Toc122122211)

[Abbildung 10 Umsetzung Profil Seite 19](#_Toc122122212)

[Abbildung 11 Umsetzung Skillseite 20](#_Toc122122213)

[Abbildung 12 Abbildung Skillseite (Light Mode) 20](#_Toc122122214)

[Abbildung 13 Umsetzung Admin Panel 21](#_Toc122122215)

[Abbildung 14 Umsetzung Admin Panel Teil 2 21](#_Toc122122216)

[Abbildung 15 Umsetzung Admin Panel Teil 2 (Light Mode) 22](#_Toc122122217)

[Abbildung 16 Umsetzung User List 22](#_Toc122122218)

[Abbildung 17 Umsetzung Userverwaltung 23](#_Toc122122219)

[Abbildung 18 Implementation der User-Suche mit Svelte 23](#_Toc122122220)

[Abbildung 19 HTML der User-Suche mit Svelte 24](#_Toc122122221)

[Abbildung 20 Auszug aus den MDN Web Docs 32](#_Toc122122222)

[Abbildung 21 Funktion zur Cookiegenerierung 32](#_Toc122122223)

[Abbildung 22: SVG Fluid Shape 33](file:///C:\Users\Andrew\Documents\GIBZ\M326%20oder%20so\gibz-fiesta\Documentation\M326DokumentationGIBZFIESTA.docx#_Toc122122224)

**TODO**

* **Layout**
* **Realisierung Teil Backend**
* **Zeitplan importieren**
* **Zwüschegspräch (höchstens im Arbeitsjournal)**