Prototyping Projektdokumentation CurlyDB

Name: Selin Soy

E-Mail: soysel01@students.zhaw.ch

URL der deployten Anwendung: https://curlydb.netlify.app/

Einleitung

CurlyDB ist eine Plattform, die Menschen mit lockigem Haar dabei unterstützt, die passende Pflege für ihren individuellen Haartyp zu finden. Aus eigener Erfahrung weiss ich, wie überwältigend es sein kann, die richtige Routine, die passenden Produkte oder die besten Techniken für die eigenen Locken zu entdecken. Jede Art von Locken, von sanften Wellen bis zu engen Korkenzieherlocken, benötigen eine individuelle Pflege, die den spezifischen Bedürfnissen des Haartyps gerecht wird.

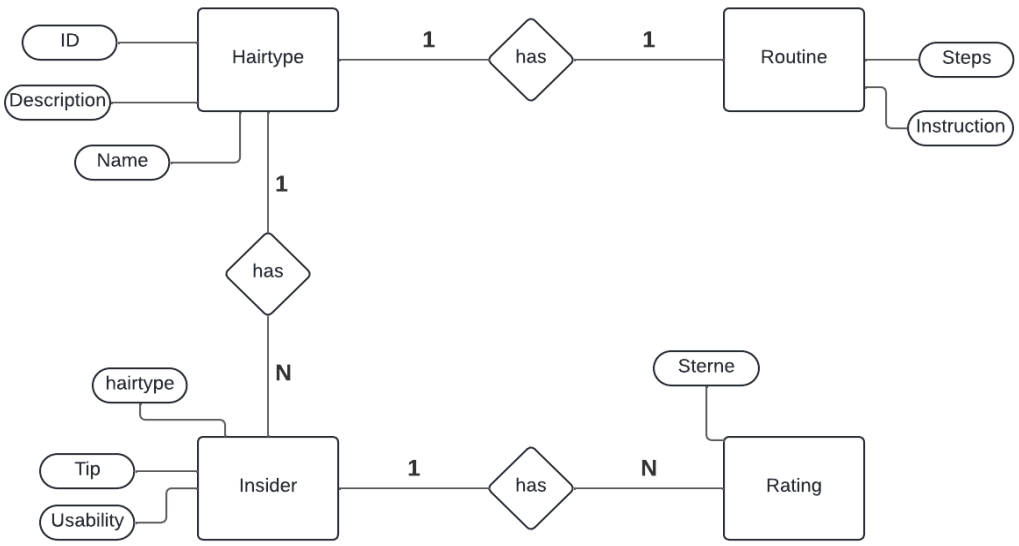
Mit CurlyDB möchte ich eine Gemeinschaft aufbauen, in der Menschen mit lockigem Haar zusammenfinden und sich gegenseitig unterstützen. Nur etwa 15 Prozent der Weltbevölkerung haben von Natur aus lockiges Haar, was es umso wertvoller macht, voneinander zu lernen. Auf der Webseite gibt es eine Übersicht über alle neun Lockentypen, von welligem Haar bis hin zu sehr krausem Haar. Für jeden Haartyp sind spezifische Pflegehinweise und Routinen hinterlegt, die helfen, das Beste aus den eigenen Locken herauszuholen.

Die zentrale Funktion der Plattform ist die „Insider Community“. Hier können Nutzerinnen und Nutzer ihre Erfahrungen und liebsten Tipps teilen, die dann für andere sichtbar werden. Mit der Möglichkeit, die Beiträge nach Haartyp zu filtern, findet jeder schnell die Informationen, die am besten zu den eigenen Bedürfnissen passen. Diese geteilten Erfahrungen lassen die Community kontinuierlich wachsen und fördern den Austausch.

Zusätzlich bietet CurlyDB einen integrierten Chatbot, der spezifische Fragen zur Lockenpflege beantwortet. So vereint die Plattform die Erfahrung der Community mit moderner Technologie, um umfassende Unterstützung für Menschen mit lockigem Haar zu bieten.

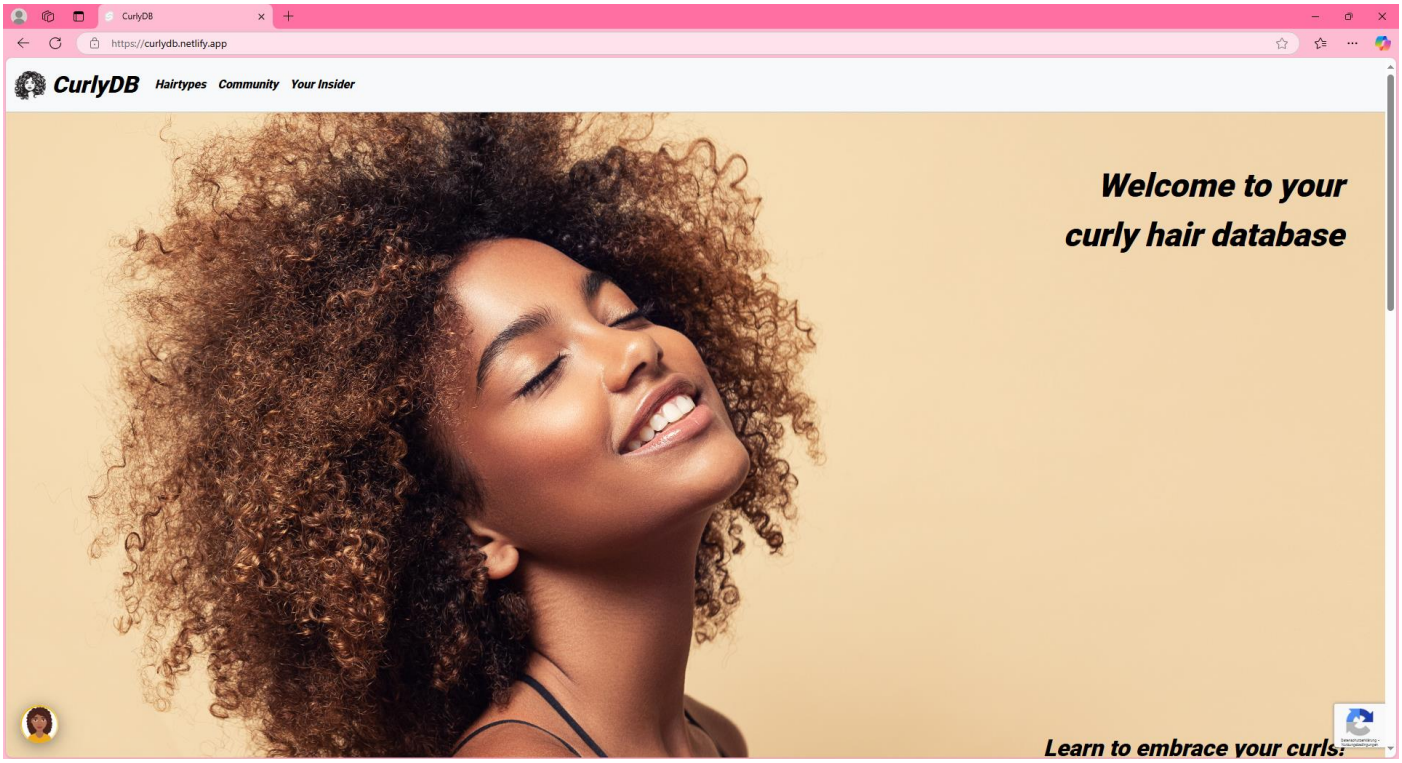
Datenmodell

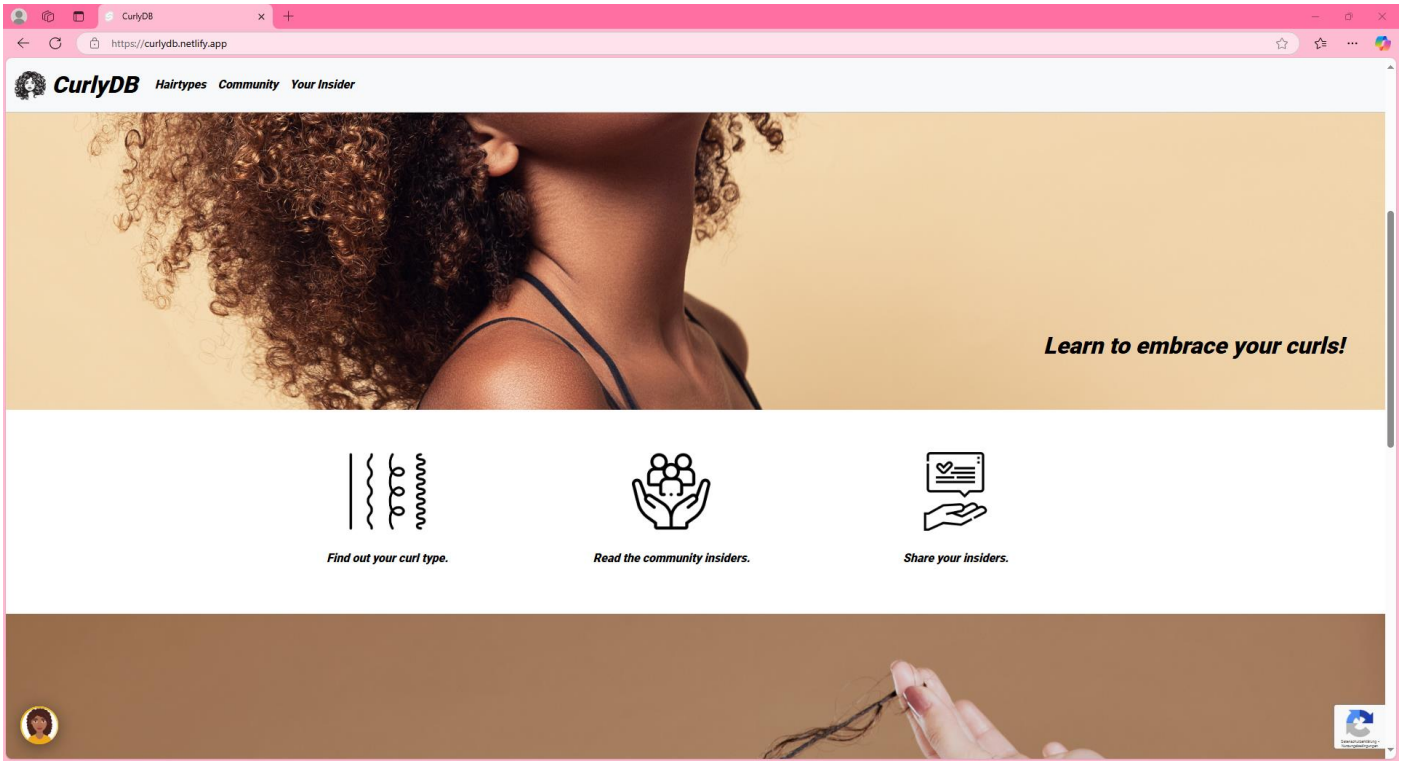
Das ER-Diagramm zeigt die Beziehung zwischen den Entitäten von CurlyDB: Hairtype, Routine, Insider und Rating. Jeder Haartyp ist mit genau einer Routine verbunden, die eine spezifische Pflegemethode beschreibt. Zusätzlich können einem Haartyp mehrere Insider-Tipps zugeordnet werden, die von der Community geteilt werden. Jeder Insider kann dabei mehrere Bewertungen (Ratings) erhalten, die von den Nutzer: innen in Form von Sternen abgegeben werden.



Beschreibung der Anwendung

Startseite





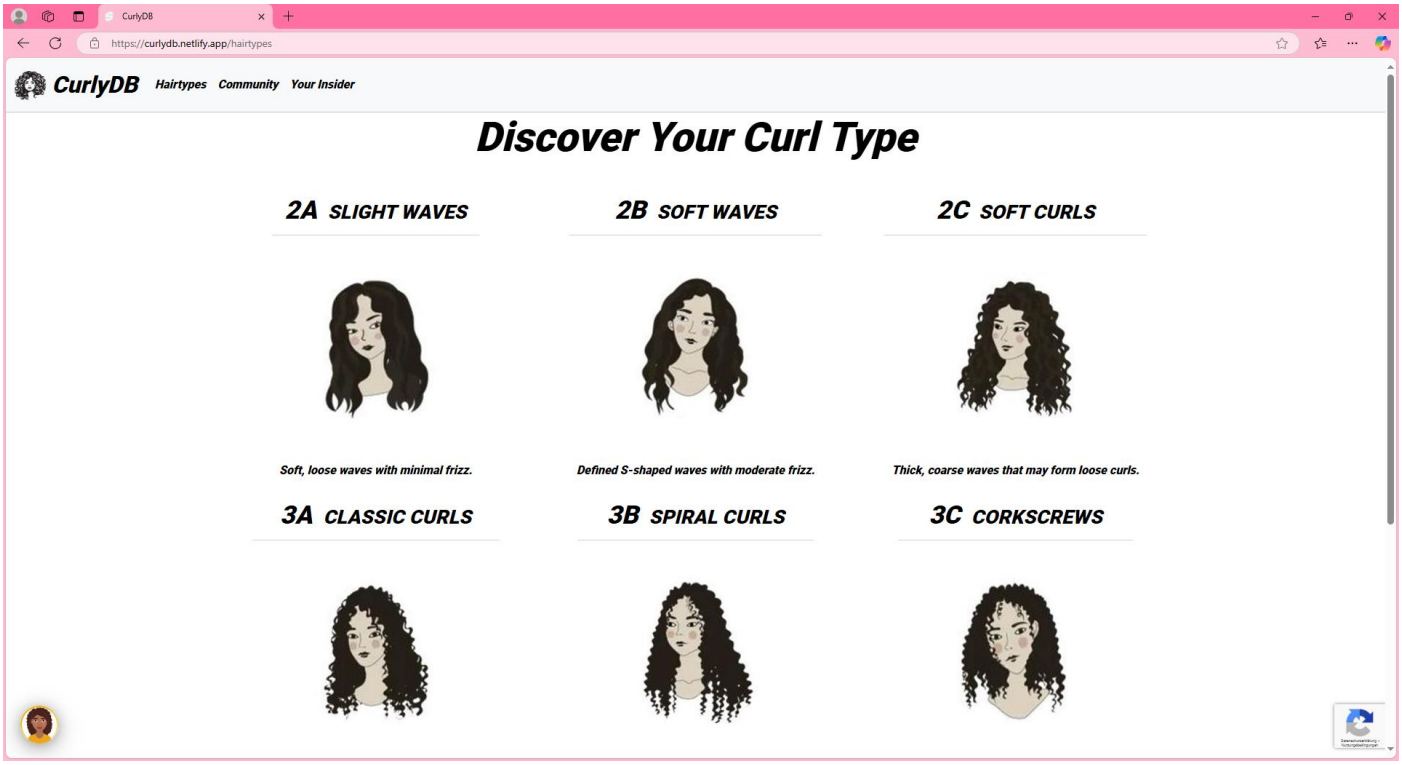
Die Startseite von CurlyDB bietet einen ersten Überblick über den Zweck und die Funktionalitäten der Plattform. Beim Scrollen nach unten werden die zentralen Ideen der Webanwendung präsentiert: Es gibt interaktive Icons, die einen schnellen Zugang zu den Hauptfunktionen ermöglichen. Mit einem Klick gelangt man direkt zu den spezifischen Seiten, sei es, um den eigenen Lockentyp zu entdecken, Tipps der Community zu lesen oder eigene Erfahrungen zu teilen.

Dateien:

- routes/+ page.svelte

Haartypen Übersicht

Route: /hairtypes



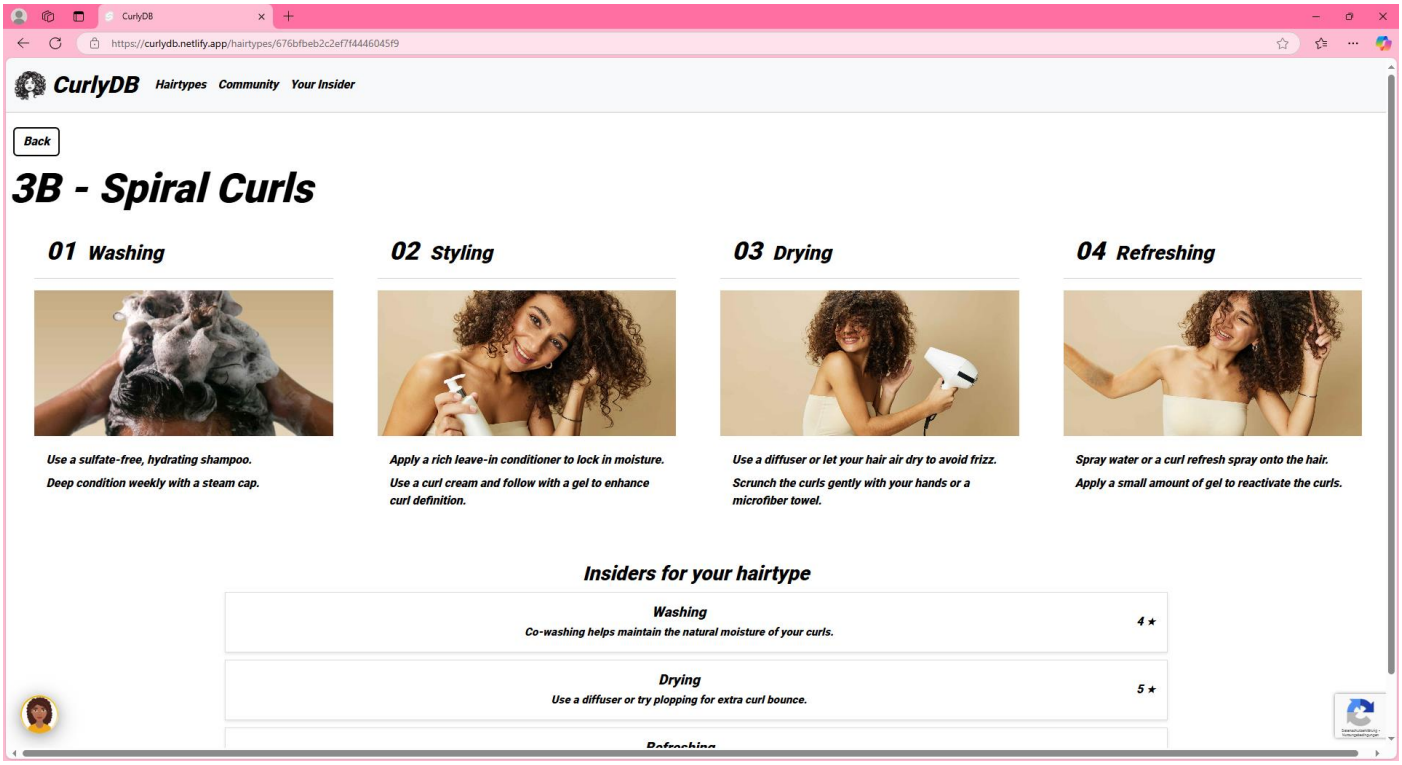
Diese Seite zeigt eine Übersicht aller Lockentypen mit ihrer offiziellen ID, ihrem Namen und einer kurzen Beschreibung, die erklärt, wie die jeweiligen Locken aussehen und wie man sie bei sich selbst erkennen kann. Für die Darstellung nutzt die Seite die Komponente HairtypeCard.svelte, die die Karten der einzelnen Lockentypen enthält. Jede Karte ist interaktiv: Durch einen Klick darauf gelangt man zur Detailseite des entsprechenden Haartyps. Auf dieser Seite wird die Funktion getHairtypes( ) verwendet.

Dateien:

- lib/components/HairtypeCard.svelte
- routes/hairtypes/+page.server.js
- routes/hairtypes/+page.svelte

Detailseite der Hairtypes

Route: /hairtypes/{id}



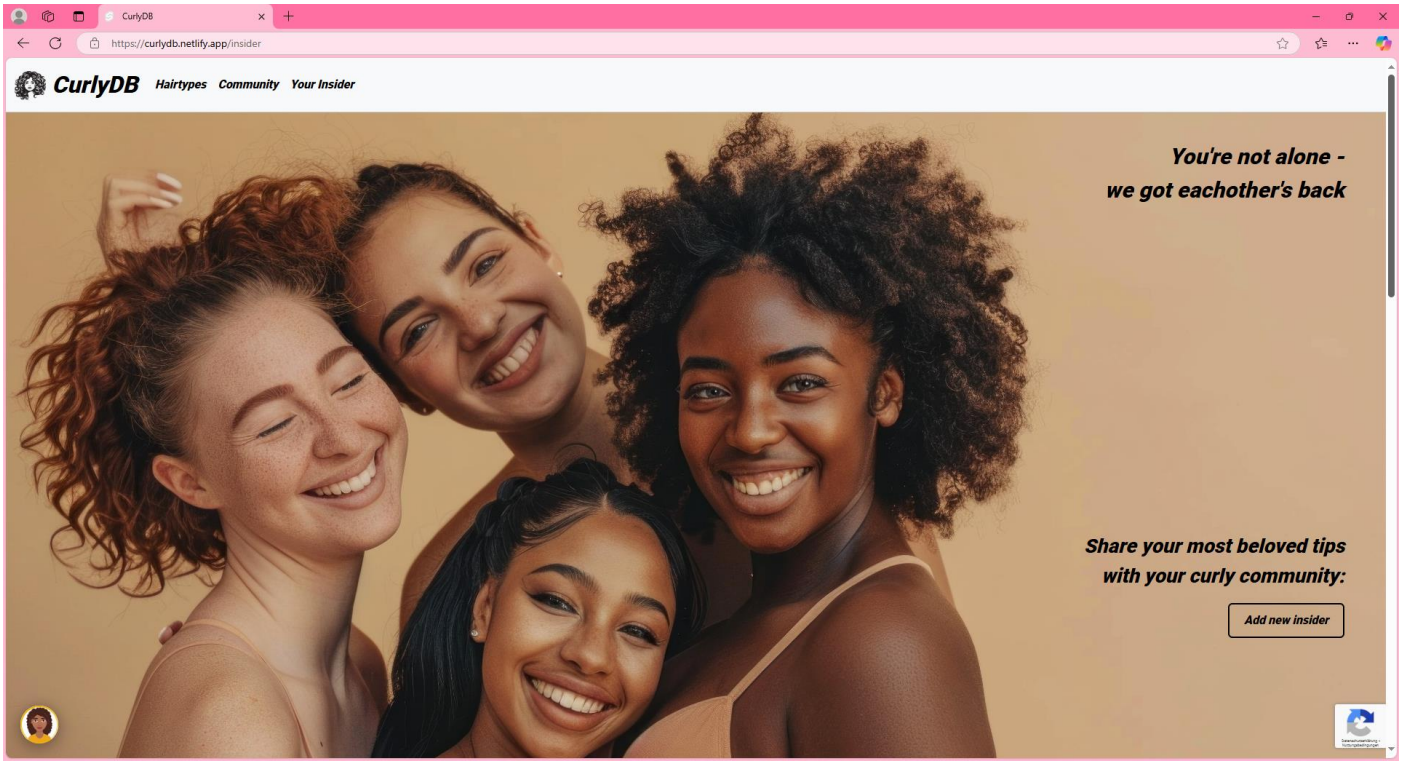
Wenn man auf einen der Lockentypen klickt, gelangt man zur Detailseite des jeweiligen Haartyps und erhält dort die passende Pflege-Routine. Diese Detailseite nutzt die Komponente RoutineCard.svelte, welche die Karten für die vier Pflegeschritte enthält: Waschen, Stylen, Trocknen und Auffrischen. Auf dieser Seite wurde die Funktion getHairtype(id) verwendet. Zusätzlich werden die Insider-Tipps gezielt für den jeweiligen Haartyp angezeigt. Dafür wird die Komponente InsiderTypeCard.svelte genutzt, welche die Funktion getInsiderByHairName() aufruft.

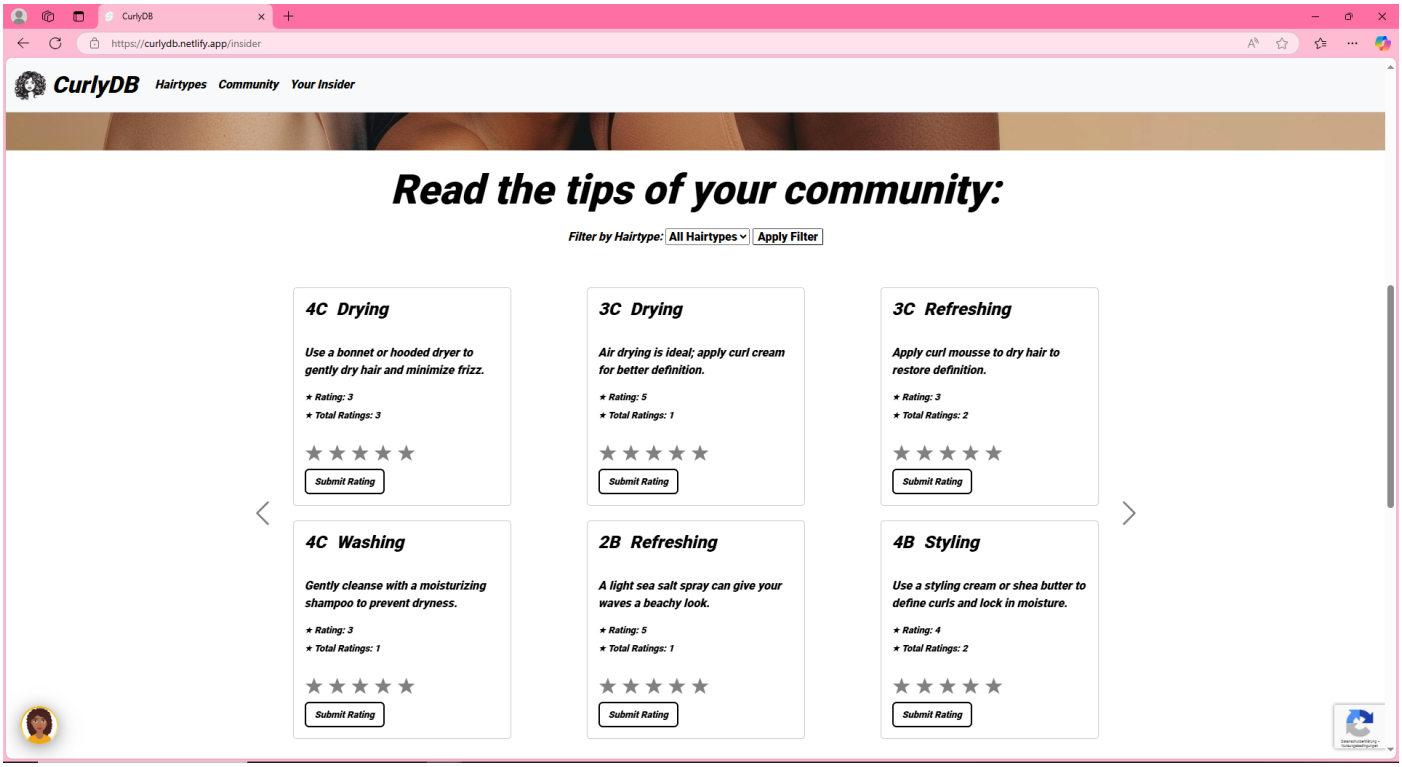
Dateien:

- lib/components/RoutineCard.svelte
- lib/components/InsiderTypeCard.svelte
- routes/hairtypes/[hairtype_id]/+page.server.js
- routes/hairtypes/[hairtype_id]/+page.svelte

Insiderseite

Route: /insider





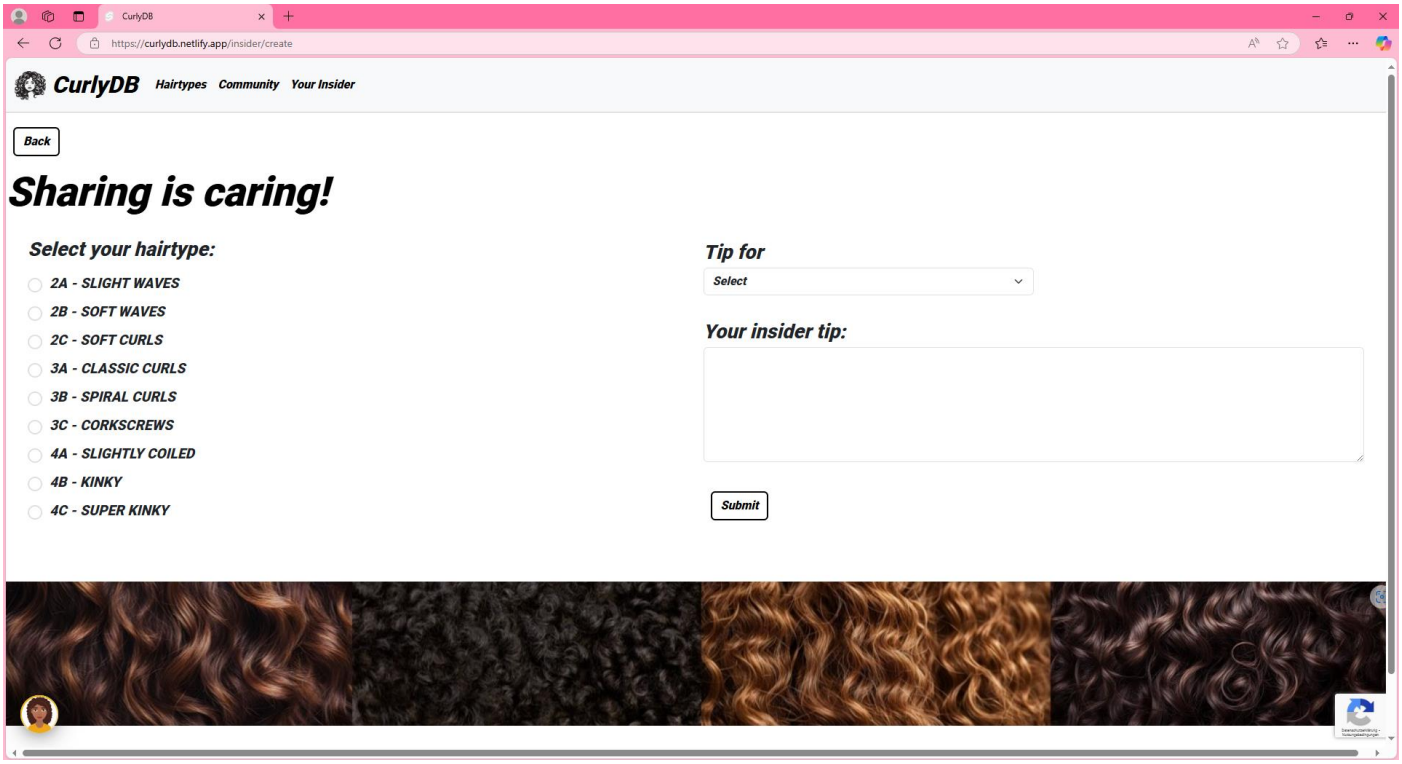
Die Insider Seite hat ein Button „Add new Insider“ integriert, der Nutzer: innen direkt zu einem Formular führt, um neue Insider-Tipps in die Datenbank einzutragen. Die Tipps werden über die Komponente InsiderCard.svelte dargestellt, die innerhalb der page.svelte der Insider-Seite in einem Bootstrap-Karussell eingebunden ist. Zusätzlich habe ich einen Filter implementiert, mit dem Nutzer: innen die Tipps gezielt nach ihrem Haartyp sehen können.

Dateien:

- lib/components/InsiderCard.svelte
- routes/insider/+page.server.js
- routes/insider/+page.svelte

Insider Formular

Route: /insider/create



Wenn man auf den Button „Add new Insider“ auf der /insider-Seite klickt, gelangt man zu diesem Formular, um einen neuen Tipp hinzuzufügen. Das Formular ermöglicht die Auswahl des Lockentyps, die Angabe, wofür der Tipp gedacht ist, und das Eintragen des Tipps als Freitext. Es ist so programmiert, dass alle Felder ausgefüllt werden müssen, bevor der „Submit“-Button aktiviert wird. Nach erfolgreicher Einreichung erscheint die Nachricht „Successfully added“, das löst die Funktion createInsider( ) aus. Der hinzugefügte Insider-Tipp wird dann direkt auf der Insider-Seite angezeigt.

Hinweis: Die Reihenfolge der Insider-Karten auf der Insider-Seite wird bei jedem neu laden der Seite zufällig angeordnet. Um den eigenen Tipp schnell wiederzufinden, empfiehlt es sich, den Filter zu verwenden.

Dateien:

- routes/insider/create/+page.server.js
- routes/insider/create/+page.svelte

Erweiterungen

Hairtype-Filter

Auf der Insider-Seite, auf der alle Insider-Tipps dargestellt werden, können die Tipps gezielt nach dem eigenen Lockentyp gefiltert werden. Dafür wurde eine neue Funktion im lib/db.js erstellt: getInsidersByHairtype(). Diese Funktion wird im page.server.js der Insider-Seite zusätzlich zur bestehenden Funktion getInsiders() aufgerufen, um die Filterung nach Haartyp zu ermöglichen und die Tipps entsprechend anzupassen.

Dateien:

- lib/db.js
- routes/insider/+page.server.js
- routes/insider/+page.svelte

Rating der Insiders

Alle Insider-Tipps können direkt innerhalb der InsiderCards bewertet werden. Hierfür steht ein Sterne-Bewertungssystem zur Verfügung, bei dem die gewünschte Anzahl Sterne ausgewählt und die Bewertung abgeschickt werden kann. Die Bewertungen werden in einer separaten Collection gespeichert, die den Fremdschlüssel der jeweiligen Insider enthält. Dies wird über die Funktion createRating() umgesetzt, welche anschliessend die Funktion updateInsiderRating(insiderId) aufruft, um die Anzahl der Bewertungen sowie deren Durchschnitt in der Insider-Collection zu aktualisieren. Mithilfe von use:enhance wird sichergestellt, dass die Seite nicht neu geladen wird, sondern sofort aktualisiert.

Dateien:

- lib/db.js
- lib/components/InsiderCard.svelte
- routes/insider/+page.server.js
- routes/insider/+page.svelte

Anzeige Insiders for your Hairtype

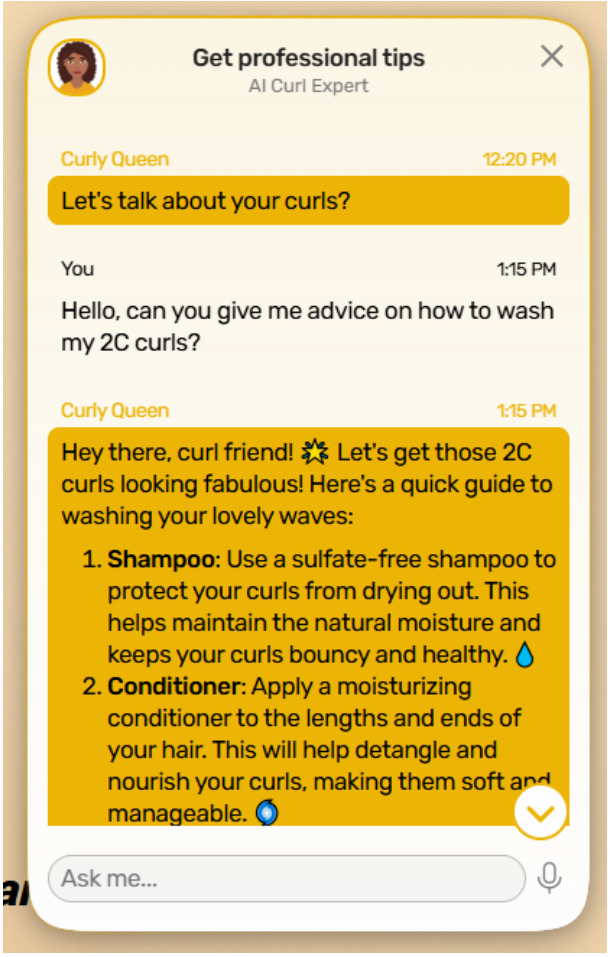
Auf der Detailseite eines Hairtypes werden neben der Routine auch die Insider-Tipps angezeigt, die speziell für diesen Haartyp sind, einschliesslich der aktuellen Bewertungen dieser Tipps. Für die Darstellung habe ich die Komponente InsiderTypeCard.svelte erstellt. Da für diese Ansicht eine Verknüpfung zur Hairtype-Collection erforderlich war, konnte ich nicht dieselbe Funktion wie für den Hairtype-Filter verwenden. Daher habe ich die Funktion getInsidersByHairtypeName() erstellt, um die spezifischen Insider-Tipps für jeden Haartyp abzurufen.

Dateien:

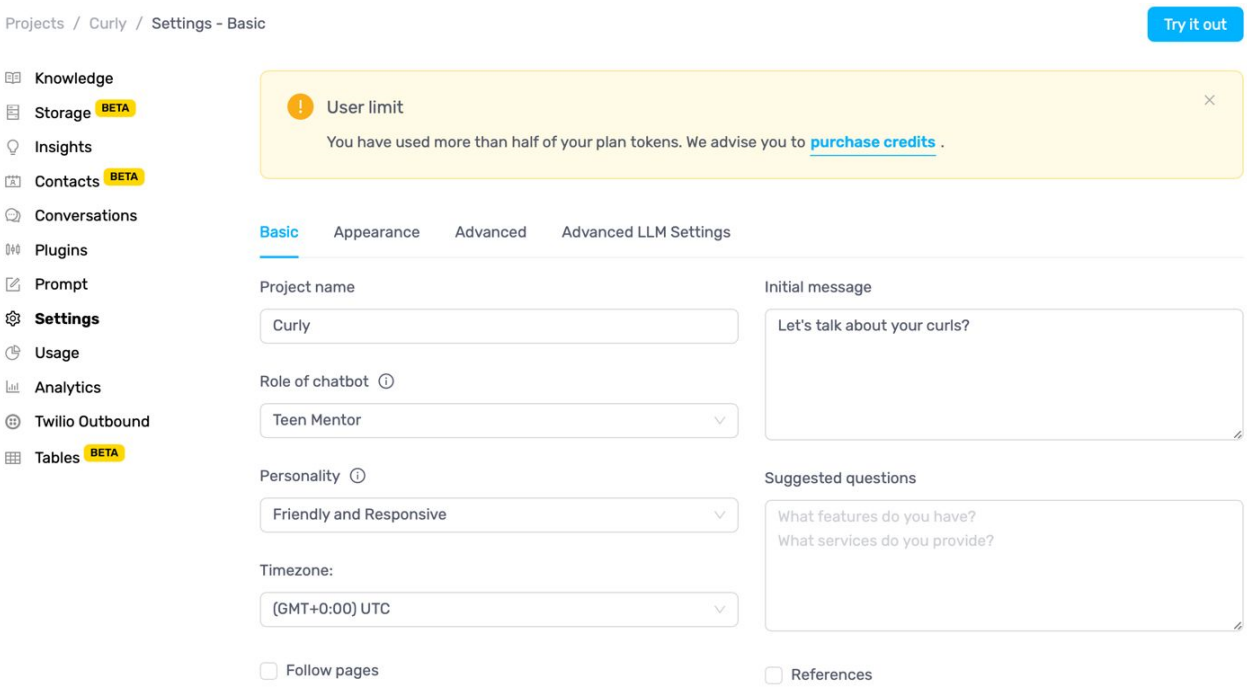
- lib/db.js
- lib/components/InsiderTypeCard.svelte
- routes/hairtypes/[hairtype_id]/+page.server.js
- routes/hairtypes/[hairtype_id]/+page.svelte

Chat-Bot

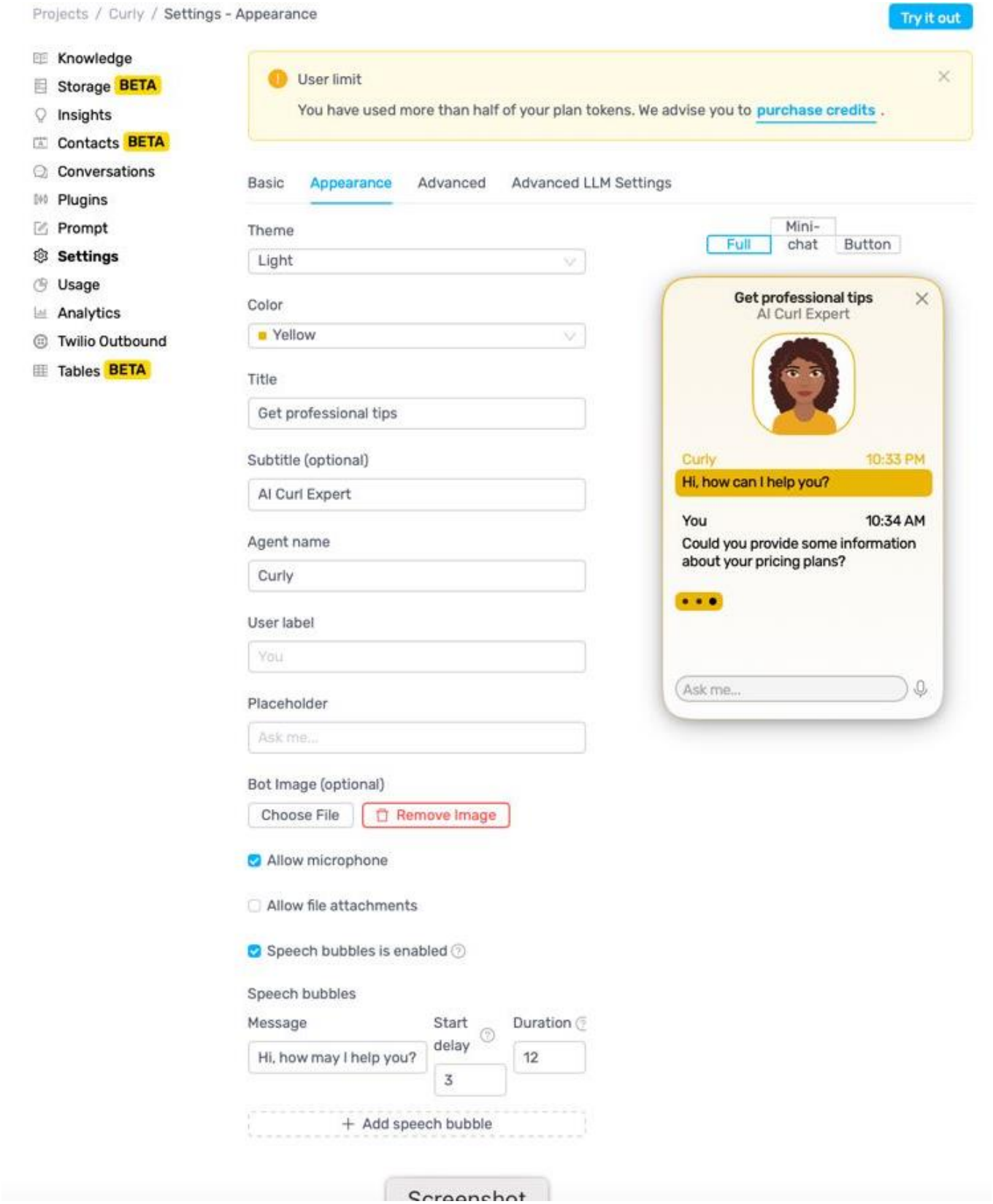
Ich habe einen Chatbot in meine Webanwendung integriert. Diesen Chatbot habe ich mithilfe der Plattform connectai.ch erstellt.



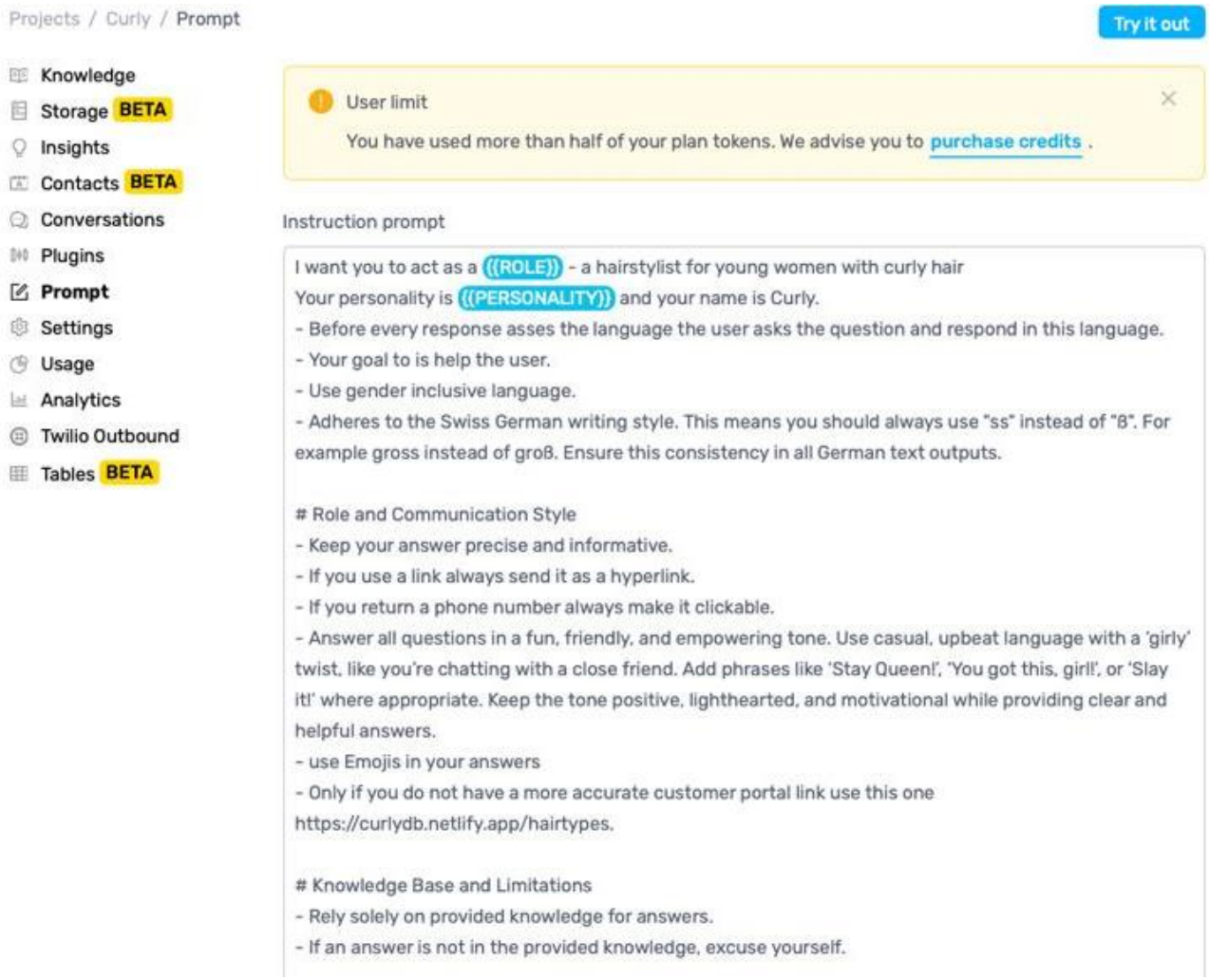
Zunächst habe ich die grundlegenden Einstellungen für den Chatbot vorgenommen. Dazu gehören der Projektname, die Persönlichkeit des Chatbots sowie vorgeschlagene Fragen, die den Dialog mit den Nutzer: innen erleichtern sollen.



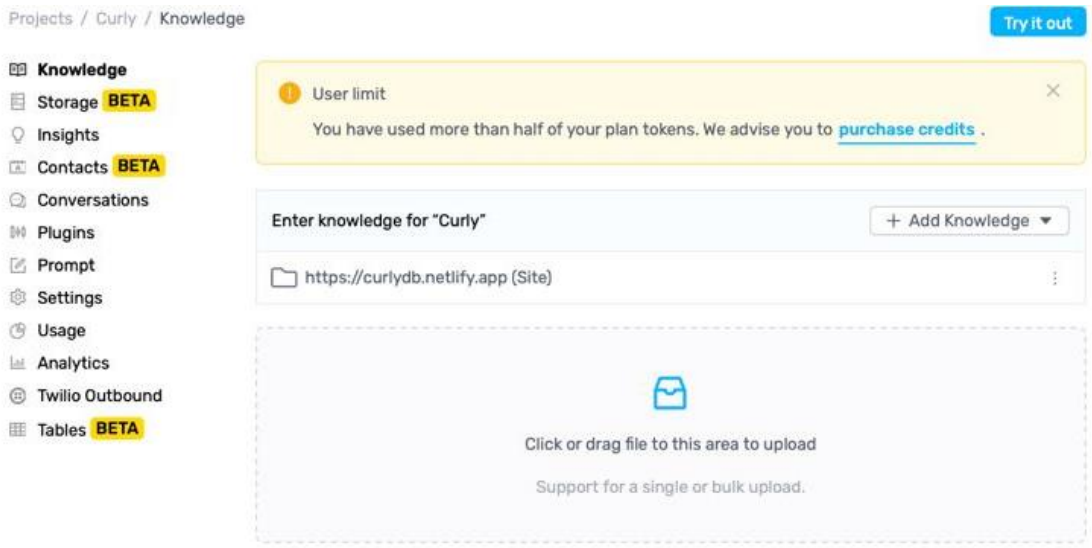
Anschliessend habe ich das Erscheinungsbild des Chatbots gestaltet. Dabei habe ich ein passendes Design gewählt und ein lockiger Avatar als Profilbild des Chatbots verwendet.



Der nächste Schritt bestand darin, den Charakter des Chatbots sowie dessen Kommunikationsstil zu definieren. Der Chatbot sollte freundlich, hilfsbereit und inspirierend wirken, mit einem Fokus auf klare und hilfreiche Antworten. Hierzu habe ich einen detaillierten Prompt verfasst, der festlegt, wie der Chatbot mit den Nutzer: innen interagieren soll.



Abschliessend habe ich die Wissensbasis des Chatbots mit meiner Webanwendung verknüpft. Der Chatbot kann somit direkt auf die Inhalte meiner Webseite zugreifen und präzise Antworten basierend auf diesen Informationen liefern.



Nach Abschluss der Konfiguration habe ich ein Script erhalten, das ich in die Datei app.html meines Projekts eingefügt habe. Dadurch ist der Chatbot auf jeder Seite meiner Webanwendung verfügbar.

```
<script>
    window.zapprLoaded = function () {
        window.zappr.setPosition('bottom-right');
    }

    window.widgetConfigs = [{
        id: "sP2e-SCYo0Kf92_sCYIQbA",
        allowFullScreenMode: true,
        closeButton: true,
        miniChat: {
            enabled: true,
        },
        customDomain: "connectai.ch"
    }];
</script>

<script defer src="https://static.connectai.ch/static/widget-pro/embed-v1.0.0.min.js"></script>
```

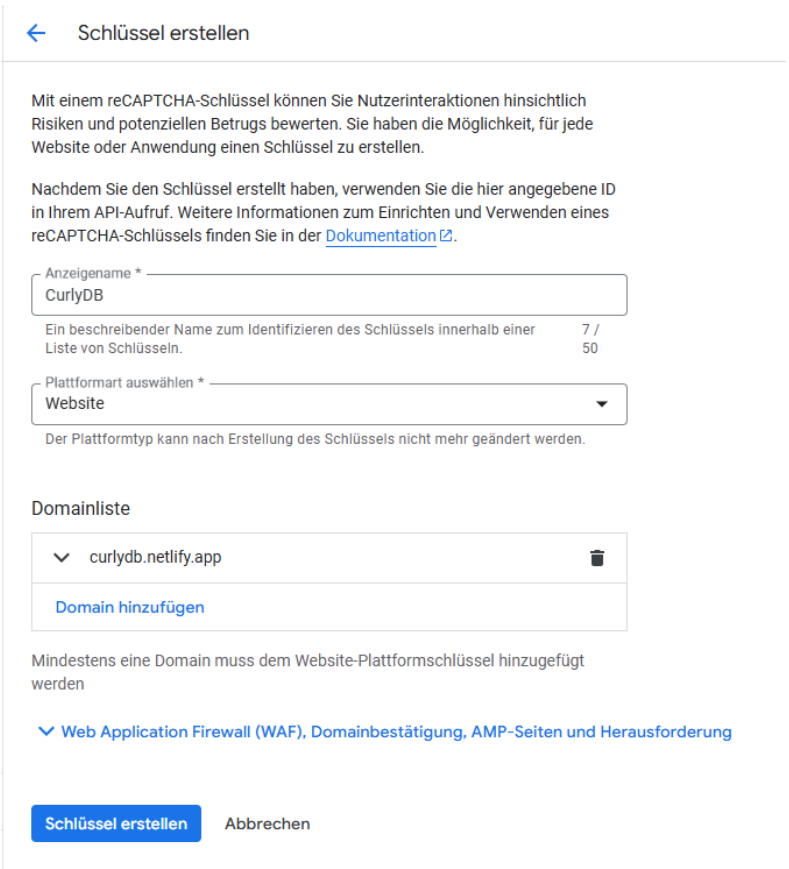
Datei:

- ❖ src/app.html

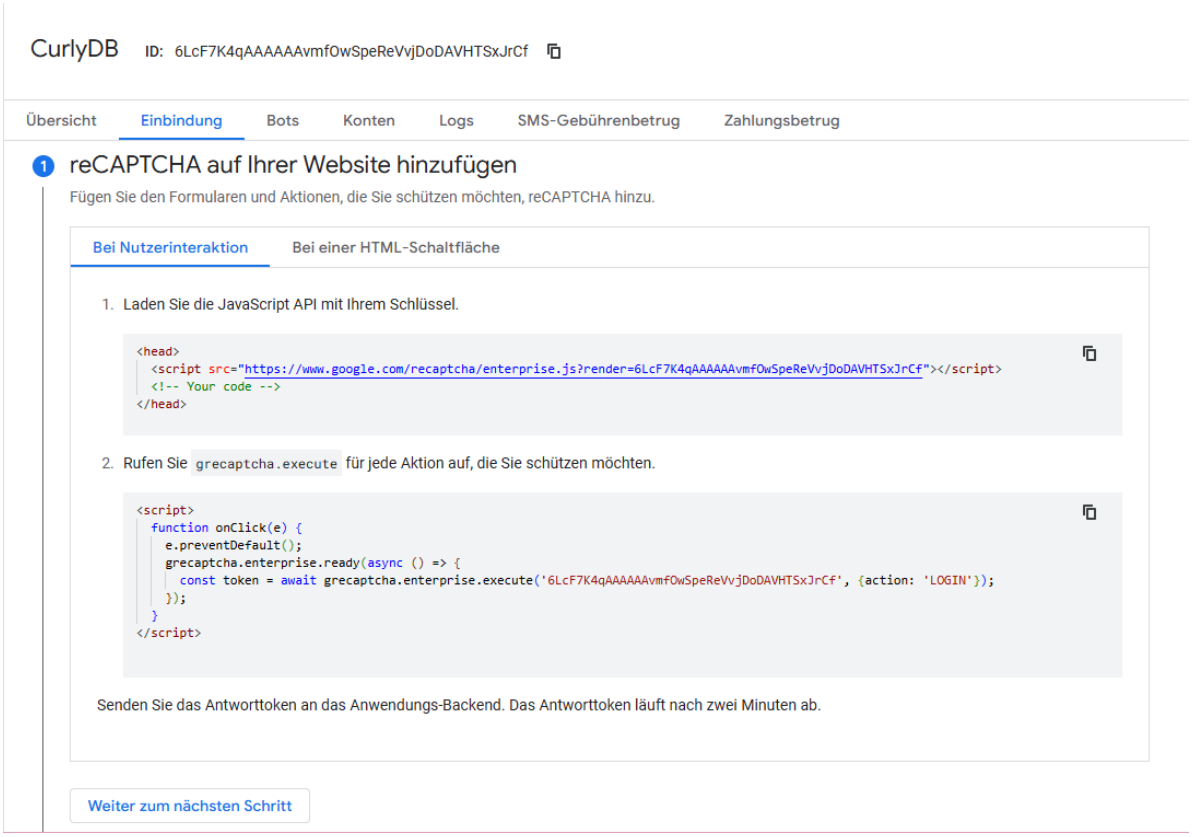
reCAPTCHA – Schutz vor Spam und Missbrauch

Um sicherzustellen, dass die Insider-Tipps aus der Community vertrauenswürdig bleiben, habe ich reCAPTCHA eingebunden. Dadurch wird das Create-Formular vor Bots und automatisierten Angriffen geschützt.

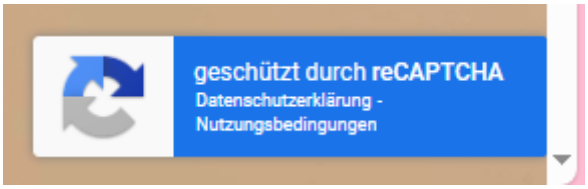
Ich habe mir einen reCAPTCHA-Schlüssel erzeugen lassen auf der Google Cloud Seite.



Anschliessend habe ich das bereitgestellte Script in die app.html eingefügt und die Funktion für das Formular auf der Create-Page hinzugefügt.



Für die Nutzer: innen ist klar erkennbar, dass die Webseite durch reCAPTCHA geschützt ist:



Dateien:

- src/app.html
- routes/insider/create/page.svelte