MEDIA
TECH LAB
by Media Lab Bayern

1. Milestone Review

1. Aktueller Status:

- *Projektfortschritt: Wo befindet sich dein Projekt momentan im Entwicklungsprozess?*

    - Bis jetzt wurde bereits der Stand eines MVP erreicht. Eine erste Beta-Version mit den grundlegenden Features wurde vor zwei Wochen veröffentlicht, eine zweite Beta-Version, in die bereits User Feedback aus mehreren User Interviews eingeflossen ist, erscheint voraussichtlich am Di., 18.06. Die These, dass RedakTool für viele Redakteur:innen nützlich sein könnte, wurde im Rahmen des Media Lab Innovation Festival am 11.06. und 12.06. durch umfangreiches positives und konstruktives Feedback unterstützt. Es wurden bereits konkrete Pläne zur Zusammenarbeit/Kooperation und gewünschte Features mit einzelnen Redaktionen (national und international) sowie mit unabhängigen Journalist:innen vereinbart (u.A. Augsburger Allgemeine, Süddeutsche Zeitung, Social Media Watchblog, PrimeNews.ch, Campact, Dekoder, whatif.zone und weitere). Außerdem haben sich konkrete Integrationsprojekte mit anderen KI-Startups (Neuraforge, Gistable) bereits angebahnt.

- *Implementierte Schlüsselfunktionen: Welche Schlüsselfunktionen wurden erfolgreich implementiert?*
    - Die Website RedakTool.ai mit Beta/E-Mail/Newsletter-Features
    - Funktionsfähigkeit der Browser-Extension auf jeder Website
    - Verfügbarkeit in (aktuell) zwei Sprachen (DE, EN)
    - Dark/Light Modus
    - Zoom-Funktion für die UI; unabhängig von der Website
    - Content-Extraktions-„Picker“ (visuell Inhalte wählen)
    - Automatische Content-Extraktion (Autokorrelationsalgorithmus)
    - Markdown-Konvertierung
    - WYSIWYG-Editor-Integration (Markdown Format Konvertierung)
    - *Anbindung an sieben verschiedene KI-Provider per API* und insgesamt etwa 80.000 kommerzielle und freie Modelle/Modellvarianten wählbar. Realistisch werden im deutschen Sprachraum vor allem die Top 20 des *https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard* verwendet, wovon alle Modelle bis auf Reka- und Yi-Modelle aktuell bereits im Service Worker unterstützt werden (jedoch noch nicht über

UI).
Yi-Modelle haben einen starken asiatischen Bias und wurden zunächst noch nicht angebunden. Integriert sind:

- One-Shot und Streaming OpenAI (GPT 3.5, 4, 4-turbo, 4o etc.)
- One-Shot Anthropic (Opus, Haiku & Co.)
- One-Shot Google (Gemini & Co.)
- One-Shot und Streaming Perplexity (Fact Checking)
- One-Shot Audio Transcription API von OpenAI für Whisper
- One-Shot Cohere (Command R+ etc.)
- One-Shot und Streaming Anbindung an Ollama-Message API Inferface: Freie, lokale Modelle: *https://www.ollama.com/library* (> 90 Open Source Modelle)
- One-Shot und Streaming HuggingFace API: Freie Modelle, lokal und hosted: *https://www.huggingface.com* (> 80.000 Open Source Modelle; nur Sprachmodelle mit Inference Endpoints kompatibel mit Message API)

- UI/UX-Refinements auf Basis von insgesamt drei User Interview-Sessions, davon ein kompletter UI-Rework mit folgenden Hauptpunkten:
  - Parallele Sichtbarkeit der Editoren (links KI-Ergebnisse ./. rechts Entwurf für Skript).
  - Tabs beinhalten Editor und KI-Prompt-Bereich dediziert untereinander, ähnlich „ChatGPT“ und anderen Beispielen.
  - KI-Prompt-Bereich erhält Formularfelder für Nutzereingaben.
  - Formularfelder werden mit Defaults belegt (aus Einstellungen).
  - „Spezialwunsch“-Eingabefeld.
  - User Feedback für Prompt-Ausführung (Streaming API Anbindung [bisher partiell - nur OpenAI, wird noch erweitert]).
  - Einklappbare Navigation und alle Bereiche resizable.
  - Zeichenzähler (Entwurf).
  - Deaktivierung von Bildern/Medien („stört bei der Konzentration auf Text“).
- Implementierung der ersten Fassung eines dynamischen Prompts für alle Kernfeatures:
  - Übersetzung
  - Zusammenfassung
  - Content-Extraktion, Bereinigung und Datenformat-Umwandlung
  - Kreatives Schreiben („neuen Content entwickeln“)

    - Lektorat

- *Gesamtaufbau: Beschreibe kurz den Gesamtaufbau deines Projekts.*

    - Die Programmcodebasis implementiert das WebExtension API, Manifest Version 3. Grundsätzlich wird der Code mit wenigen Anpassungen in Google Chrome / Chromium, Microsoft Edge, Mozilla Firefox und Opera funktionieren. Apple Safari-Kompatibilität würde etwas mehr Aufwand erfordern, ist aber ebenfalls möglich (seit Safari 14+ von 2020). Zunächst konzentriert sich die Implementierung jedoch auf Google Chrome.

    - Der Programmcode teilt sich grob in zwei Domänen auf:

        - **Service Worker**: Dieser arbeitet im Hintergrund mit erweiterten Berechtigungen und unabhängiger Laufzeit. Hier finden lang-laufende Prozesse statt, die Speicherung und er Abruf von Daten in der IndexedDB, Verwendung von Browser-spezifischen API's, Ausführung der Vektordatenbank per WASM und die Kommunikation mit API-Endpunkten (lokal oder remote).

        - **User Interface**: Dieser wird im Kontext jeder Website, die besucht wird ausgeführt und hat Zugriff auf den DOM der besuchten Website. Der Programmcode „injiziert" dabei einen JavaScript-Code, der kontinuierlich dem DOM überwacht und analysiert. So können z. B. Audio- und Video-Elemente die nachträglich geladen werden (z. B. Nach Einblenden einer Werbung) für die Transkription zugänglich gemacht werden, indem der Audio-Stream abgegriffen wird. Auch können DOM-Inhalte hier per Autokorrelation erkannt und in Markdown gewandelt werden, um im WYSIWYG-Editor bearbeitbare gemacht zu werden. Die Oberfläche (UI) wird als Web Component gebündelt und in die Website geladen. Sie ist sodann per Tastenkombination oder Klick in der Erweiterungsverwaltung aufrufbar. Da sie im Shadow DOM der Website arbeitet, sind die Styles (CSS) und JavaScript-Ausführungskontext weitgehend isoliert - es kommt bis auf bei der Schriftgröße zu keinen Kompatibilitätsproblemen oder Beeinflussung zwischen Extension und Website. Die UI

kommuniziert per postMessage()-API bi-direktional und asynchron mit dem Service-Worker code (Message Broker-Architektur).

2. Abgleich mit dem Meilensteinplan Monat 1 & 2:

- *Erreichte Meilensteine:*
  Welche Ziele aus dem Meilensteinplan für die ersten beiden Monate wurden erreicht?

    - Alle Ziele wurden grundsätzlich erreicht. Es ergaben sich jedoch vier kleine Einschränkungen (siehe unten). Darüber hinaus wurde jedoch bereits weitere Funktionalität, die eigentlich für den Milestone 3 geplant war, in Teilen implementiert, da zur Reduktion des Risikos ein technischer Proof-of-Concept umgesetzt wurde:

    - Implementierung der Transkription von Audio/Video-Dateien und Live-Videos/Streams/Podcasts etc. (Youtube & Co.)
        - Erkennung und Analyse von Video und Audio-Elementen auf einer Website (*live!*).
        - DSP-Algorithmik zur Reduktion von Rauschen durch *BiQuad-EQ-Filter*.
        - Korrektes *Schneiden an Sprechpausen* durch Lautstärke-Autokorrelation im Audio-Datenstrom.
        - *Transkodierung von Audio-Inhalten* im Browser hin zu Opus als Format mit der besten Datenkompression.
        - PoC für den Schnitt nach Kontext-Größe (Whisper kann Inhalte mit mehreren Stunden Laufzeit nicht auf einmal verarbeiten).
        - Einzelnes Probehören und Transkribieren der Abschnitte mit Zeitmarken.

- *Nicht erreichte Meilensteine:*

    - Alle Ziele wurden grundsätzlich erreicht, allerdings mit **vier kleinen Einschränkungen.**

    - **Funktional:**

- Die Anbindung an die KI-Modelle (Modellauswahl) kann noch nicht dynamisch in der Oberfläche konfiguriert werden. Hierfür muss der Bereich „Einstellungen“ erweitert werden.

- Außerdem erfordert das Authentication Framework von Google eigentlich eine Private Key Datei für den API-Zugriff. Hier ist Arbeit erforderlich, einen korrekten JWT Token über andere Wege zu konstruieren, damit die Kommunikation mit den Gemini-Modellen auch in einer Browser-Extension ohne Dateisystem-Zugriff möglich ist.

- Die Konvertierung von HTML zu Markdown hat sich nicht immer als eine reibungslose Thematik gezeigt. So werden beispielsweise Tabellen in Markdown immer mit Header-Zeile erwartet, doch nicht jede Tabelle in HTML hat auch einen Header. Links werden nicht erkannt, wenn sie Umbrüche enthalten, aber oft haben Link-Texte auf Webseiten eben Umbrüche im Format. Für die Bereinigung dieser kleinen Bugs ist ein Vorverarbeitumgsschritt als Optimierung für Milestone 3 und 4 geplant. Technisch ist bereits offensichtlich, wie dies gelöst werden sollte.

  Mittel- bis langfristig kann als Alternative auch eine Umstellung auf HTML als Datenformat für die Bearbeitung der Textinhalte angemacht werden, hier ist dann aber ebenfalls mit Problemen zu rechnen, da HTML deutlich komplexere Varianten von Elementen und Styling ermöglicht, die ebenfalls nicht immer völlig problemlos in einem WYSIWYG-Editor behandelt werden können. Für den Moment und insbesondere auch aus dem User Feedback heraus, dass komplexe Formatierung gar nicht gewünscht (da als störend wahrgenommen) ist, scheint mir die Entscheidung für Markdown und simple Formatierung, sowie eine Optimierung der Konvertier-Logik auf wenige, aber dafür gut funktionierende Stil-Elemente am sinnvollsten zu sein.

- **Nicht-funktionale Einschränkungen:**

  - Ein paar „Ecken und Kanten“ im UI/UX-Bereich sollten ferner in Milestone 3 und 4 noch geschliffen werden. Das Streaming der KI-Modellantworten sollte z.B. auch visuell sichtbar sein und

auch Ladeoperationen sollten entsprechend durch Lade-Icons den Nutzer:innen bewusst gemacht, und UI-Elemente deaktiviert werden.

3. Ausblick in die nächsten 2 Monate:

- *Machbarkeit der kommenden Meilensteine:*
  Wie schätzt du die Realisierbarkeit der geplanten Meilensteine für die nächsten zwei Monate ein?

    - In Monat 3 und 4 kommen zwei wichtige Features auf mich zu, die große technische Herausforderungen darstellen werden. Dazu gehört die Integration der lokalen Vektor-Suche, sowie die automatisierte Transkription. Beide Themen sind an der Grenze dessen, was technisch aktuell machbar ist. Die technischen Herausforderungen erscheinen mir im Wesentlichen jedoch weitestgehend bekannt/ offensichtlich, abschätzbar und lösbar.

- *Herausfordernde Ziele:*
  Gibt es spezielle Herausforderungen bei bestimmten Zielen, die du besonders hervorheben möchtest?

    - Für die automatisierte Transkription per Whisper und mit Live-streaming, DSP-Filterung, Transkodierung und Schnitt an Sprechpausen gibt es ein *Paper*. Meinen Ansatz habe ich unabhängig davon entwickelt und fand das Paper erst nachträglich. Für RedakTool wird der Audio-Inhalt einfach fortlaufend im Browser geschnitten und mit der Transkription des jeweils letzten Abschnitts das nachfolgende gebiased. Dadurch funktioniert mein Ansatz für jedes Transktiptionsmodell. So könnte später auch z.B. das Deepgram Voice AI Modell und zukünftig auch weitere Transkriptions- und Übersetzungsmodelle per API angebunden werden.

      Die technische Machbarkeit hierfür ist bereits in der Beta-Version demonstriert. Die Implementierung muss jedoch stabilisiert, automatisiert und optimiert werden. So sollte das DSP-Handling im Service Worker erfolgen und nicht den UI-Thread blockieren.

Generell erwarte ich dazu noch einige Hürden bei der Transkodierung (fehlende/inkompatible Audio/Video-Codecs), Stream-Abbruchbehandlung, Asynchonität und generell Fehlerbehandlung in der Retry-Logik (komplizierte Stückelung; hier muss ich einen Stack mit Index-Sortierung implementieren).

- Beim Thema in-browser Vektor-Datenbank habe ich bereits in Milestone 1 *einige Fortschritte* gemacht, die zeigen, dass die Vektor-Suche auch lokal und im Browser mittels *HNSW-Algorithmus* und WebAssembly (SIMD, CPU vector instructions) beschleunigt werden kann.

  Durch das äußerst kleine Text-Embedding-Modell Nomic Embed Text v1 lässt sich mittels *Transformers.js* ein Embedding in hinreichender Qualität auch im Browser ausführbar machen. So lassen sich Textinhalte für Suche und Klassifizierung mittels eines Vectorstores *indexieren* und *wieder abrufen*.

  Dafür muss jedoch erst Vectorstore und die Vektor-Suche korrekt implementiert werden. Auch an dieser Stelle gibt es keine Lösung die direkt eingesetzt werden kann. Die dafür nötige Implementierung habe ich als Open-Source-Projekt *Vectorstore* unter MIT-Lizenz veröffentlicht.

  Alle Komponenten für Indexierung, Suche und Klassifizierung müssen anschließend perfekt aufeinander abgestimmt und miteinander integriert werden. Hier sind noch weitere Tests und Optimierung nötig.

  Ferner muss die Performance in der Anwendung soweit optimiert werden, dass eine Nutzung auf einem durchschnittlichen modernen Laptop mit durchschnittlicher CPU- und Speicherausstattung Sinn ergibt. Im Zweifel müssen Teile der Ausführung in den Service Worker und somit in den Hintergrund ausgelagert werden (delayed processing).

  Als Persistenz-Layer für den Vectorstore muss ein Low-Level Adapter zur Speicherung und Abruf der Vektor-Embedding-Daten als Blob implementiert werden. Das scheint mit der einfachste Part zu werden.

- *Fokussierte Schwerpunkte:*
  Worauf legst du in den kommenden Monaten den Hauptfokus, um die Projektziele zu erreichen?

    - **Optimierung**: Erst die Fertigstellung und Optimierung der Prompts und der bestehenden Funktionalität, sodass sie grundsätzlich sinnvoll eingesetzt werden kann, dann die Optimierung des bestehenden Transkriptions-Feature hin zur vollkommen automatischen Transkription.

    - **Persistenz, Vektor-Datenbank, Suche:** Sobald die Optimierung abgeschlossen ist, können Inhalte Extrahiert und sinnvoll mit Hilfe der KI-Features bearbeitet werden. Anschließend sollen sie auch gespeichert und durchsuchbar gemacht werden. Der Stand der Bearbeitung soll über Webseiten hinweg erhalten bleiben und Entwürfe im Archiv gespeichert / per Vektor-Suche wiedergefunden werden können.

Zusätzliche Anmerkungen oder Herausforderungen:

Hier können Themen hinzugefügt werden, die nicht direkt durch die vorherigen Fragen abgedeckt wurden.