Automatisierte Generation von Anki-basierten Karteikarten aus Markdownquellen

Lars Quentin
Beteuung: Prof. Dr. Carsten Damm

03.03.2023

Abstract

Karteikarten und selbstgeschriebene Zusammenfassungen sind zwei wertvolle Ressourcen zur Vorbereitung von Klausuren und dem generellen Lernen neuer Themenbereiche. Während Karteikarten sich am besten für wissensintensives Auswendiglernen eignen sind Zusammenfassungen besser zum Verstehen komplexer, stark zusammenhängender Themen. Das Ziel dieser Arbeit liegt darin, diese beiden Ansätze zu vereinen. Durch die Konzipierung einer Markdownerweiterung mit Karteikartensyntax wurde die Möglichkeit geschaffen, aus Zusammenfassungen Karteikartendecks zu extrahieren, welche dann isoliert mit dem Spaced-Repitition-System Anki gelernt werden können. Des weiteren wurden für GitHub sowie GitLab kollaborative Workflows entwickelt, bei welchem automatisiert nach jedem Commit durch die Nutzung von Continuous Integration (CI) automatisiert ein neues Kartenarchiv erstellt wird.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Das Problem des Lernens, insbesondere in universitären Bereich, lässt sich in zwei distinkte Kategorien aufteilen: Das verständnisorienterte Lernen komplexer Themen und das faktenbasierte Lernen spezifischer Details. Obwohl die meisten Klausuren eine Kombination beider Kategorien darstellen, erfordert jede Art unterschiedliche Vorgehensweisen:

- Im Falle verständnisbasierter komplexer Themen können Lernzettel geschrieben werden, in dem Informationen zusammengefasst und umformuliert werden. Dies resultiert in immer noch großen komplexen Texten oder Stichpunktlisten, in welchen die einzelnen Informationen sehr zusammenhängend sind. Die einzelnen Themen sind weniger isoliert; das erwartete Detailwissen ist geringer. Diese Klasse an Themen ist typisch für logisch-orientierte Studiengänge wie z.B. Mathematik, Philosophie, Informatik oder Physik.
- Im Gegensatz dazu erfordert das akribische Lernen von faktenbasierten Details viel Auswendiglernen und wird meistens über Karteikarten praktiziert. Dieser Ansatz ist typisch für Studiengänge mit einem hohen Lernvolumen, wie z.B. Medizin, Biologie oder Sprachwissenschaften. Allgemein findet sich diese Problemart oft in komplexen, nicht-menschengemachten Systemen wieder.

Im digitalen Bereich gibt es für diese beide Arten des Klausurlernens kanonische Lösungen:

- **Lernzettel:** Insbesondere in der Informatik werden für Lernzettel immer häufiger Markdown [1] verwendet. Markdown ist eine einfache, aber leistungsstarke Markup-Sprache die einige Vorteile bietet: Zum einen kann sie ohne spezielle Software direkt von Menschen geschrieben werden, zum anderen ist sie einfach parsebar, wodurch sie mit vielen Tools genutzt werden kann. Zudem kann Markdown durch die simple, zeilenbasierte Struktur mit Git versioniert werden. Es unterstützt simple Medien wie Tabellen, Bilder und LaTeX-Formeln, was es für die meisten Anwendungsfälle expressiv genug macht. Es gibt auch Tools wie Hedgedoc [2] zum kollaborativen Live-Editieren, welches unter anderem auch als Instanz von der GWDG gehosted wird [3]. Es ist einfach, Markdown durch Applikationen wie Pandoc [4] in HTML oder RevealJS-Presentationsfolien [5] via Hedgedoc zu exportieren.
- **Karteikarten:** Digitale Karteikarten werden häufig mit sogenannten Spaced-Repitition-Systemen (SRS) gelernt. Ein SRS ist ein Lernsystem, das Lernaufwand minimiert, indem es die optimale Zeit zur nächsten Wiederholung der Frage auf Basis der bisherigen Kartenhistorie sowie der Selbsteinschätzung der Schwierigkeit approximiert. Dies bedeutet das Fragen, die man leicht beantworten kann, seltener wiederholt werden als schwierigere Fragen, mit welchen man auch historisch bereits Probleme hatte.
  Das bei weitem beliebteste SRS ist Anki [6], ein Open-Source-Programm [7], das durch eine iOS-App [8] finanziert wird. Es basiert auf Chromium [9] und hat somit einen hohen Mediensupport. Anki bietet einen kostenlosen Cloud-Sync [10], der es ermöglicht, Karteikarten auf verschiedenen Geräten zu synchronisieren. Darüber hinaus gibt es eine kostenlose, von der Community erstellte Androidversion. Anki hat auch eine große Bibliothek von Decks, die von der Community geteilt werden, sowie Plug-Ins-Support [11], die es Benutzern ermöglicht, die Funktionalität von Anki zu erweitern.

Trotz der Vorteile, die sowohl Lernzettel als auch Karteikarten bieten, haben sie jeweils auch ihre Nachteile. Lernzettel sind ineffizient zum Details lernen, da sie oft sehr verbos sind und eine geringere Informationsdichte haben. Darüber hinaus kann man keinen expliziten Fokus auf Themen setzen, welche einem besonders schwerfallen, da der Text sich nicht so einfach in atomare Fakten aufteilen lässt. Karteikarten jedoch verlieren schnell ihre Reihenfolge. Zudem kann man zwischen Karteikarten keinen Kontext erhalten, welcher gegebenenfalls benötigt wird um das größere Konzept zu verstehen.

In diesem Praktikumsbericht wird eine Lösung vorgestellt, Lernzettel mit Markdown und und Karteikarten mit Anki logisch zu verbinden. Die Idee basiert auf dem Konzept von Literate Programming.

Literate Programming ist ein Konzept, das von Donald E. Knuth entwickelt wurde [12]. Es kombiniert Programmcode mit dessen Dokumentation, so dass beide in einer einzelnen Datei vereint werden. Die Funktionen sind innerhalb der Dokumentation integriert und die gesamte Dokumentation lässt sich ausführen. Das ursprüngliche Konzept von Literate Programming verwendet eine eigene Sprache namens "WEB", welche von Knuth selbst entwickelt wurde. WEB-Quelldateien kompilierten entweder zu Pascal oder TEX[13].

Moderne Umsetzungen von Literate Programming sind Jupyter Notebooks [14]. Jupyter Notebooks sind interaktive Dokumente, die Code, Markdowntext und Medien durch sogenannte Zellen in einem Dokument vereinigen. Da Jupyter Notebooks ohne explizite Programmierumgebung gehosted in einem Webbrowser, zum Beispiel via GWDG Jupyter Hub [15] oder Google Colab [16], benutzbar sind haben sie eine geringe Einstiegshürde. Hierdurch sind sie interdisziplinär in der Forschung omnipräsent. Sie implementieren das Konzept von Literate Programming, indem man den Code als Teil des Dokumentes betrachtet statt als isolierte Einheit.

1.2 Ziele und Beiträge

Das Ziel dieser Arbeit ist, analog zum Konzept des Literate Programming, Karteikarten in Markdownzusammenfassungen zu integrieren. Hierdurch wird es möglich, das gesamte Markdown-Ökosystem zum Erstellen, Bearbeiten und Versionieren der Karteikarten zu nutzen. Einerseits ermöglicht dies, dass Karteikarten den Kontext und die Reihenfolge behalten, welche für das Lernen komplexer Themen erforderlich ist. Andererseits soll ein automatischer Anki-Export ermöglicht werden, der die Nutzung von Spaced Repitition für ein effizienteres Lernen von Fakten ermöglicht.

Im Rahmen dieses Praktikums wurden mehrere Beiträge geleistet, um die gesteckten Ziele zu erreichen. Zunächst wurde ein neuer Dateistandard für `.anki.md` Dateien entwickelt, um die Integration von Karteikarten in Markdownzusammenfassungen zu ermöglichen.

Darüber hinaus wurde ein Tool erstellt, das automatisch aus einem Ordner mit Markdown-Dateien ein Ankideck erstellen kann [17]. Hierbei wird die Ordnerstruktur auf die Struktur der Ankidecks übertragen, und es wird sämtliche Markdown Syntax sowie das Rendering von LATEX-Formeln unterstützt. Das Tool kann auch mit der Mobilversion von Anki genutzt werden. Zudem kann mit dem Ankideck offline gelernt werden, da alle verlinkten Bilder in das Deckarchiv gespeichert werden.

Zuletzt wurden Git-Repository-Vorlagen für GitLab [18] und GitHub [19] konzipiert und implementiert, mit welchen automatisch via Continuous Integration nach jedem Commit ein aktualisiertes Ankideckarchiv erstellt wird. Dies ermöglicht die Erschaffung von Ankikarten ohne lokal installierte oder technischem Verständnis des Konvertierungsprozesses.

2 Softwarekonzeptionierung und Methodologie

2.1 Entwurf einer Markdownerweiterung

Markdown ist zwar sehr verbreitet, jedoch gibt es keine universale Markdownspezifikation. Stattdessen gibt es viele verschiedene Varianten, welche aber leicht voneinander abweichen. Somit, bevor man einen Markdownstandard erweitern kann, muss ein Grundstandard spezifiziert werden.

Für den Anwendungsfall dieses Praktikums wird als Basis der am weitesten verbreite CommonMark [20] Standard genommen. Dieser ist sehr simpel, jedoch von den meisten Implementationen unterstützt. Insbesondere ist er zu den meisten üblichen Standards kompatibel.

Desweiteren soll der hier entwickelte Karteikarten-Standard auch mit folgenden Markdownstandards kompatibel sein:

- **GitHub Flavored Markdown (GFM) [21]:** Benötigt für die automatische Kartengenerierung über GitHubrepositories.
- **GitLab Flavored Markdown (GLFM) [22] :** Benötigt für die automatische Kartengenerierung über GitLabrepositories.
- **HedgeDoc Flavored Markdown 2.0 (HFM 2.0) [23]:** Benötigt zum kollaborativen Erstellen von Karteikarten über die HedgeDoc-Infrastruktur der GWDG [3].

Somit, aufbauend auf CommonMark, wird im Rahmen dieses Praktikums ein Standard für `.anki.md`-Dateien definiert. Der Standard hat folgende Ziele:

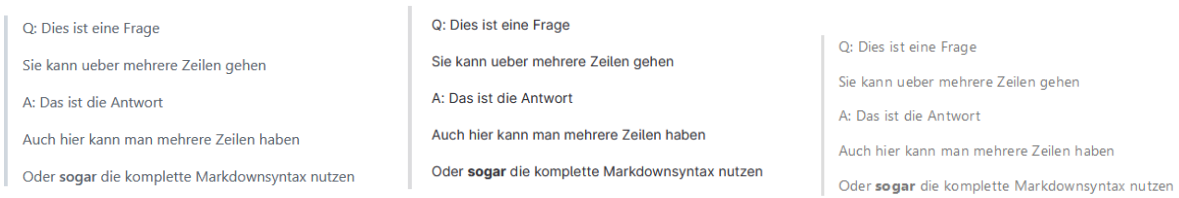
- Eindeutige Parsebarkeit.
- Renderunterstützung in GitHub, GitLab sowie HedgeDoc.
- Einfache Syntax, welche sich manuell mit wenig Extraaufwand schreiben lässt.
- Gute Lesbarkeit in Codeform.
- Keine Kollisionen mit normalem Text; keine falschen positiven Erkennungen.

Die unsterstützte Karteikartensyntax funktioniert wiefolgt:

```
> Q: Dies ist eine Frage
>
> Sie kann ueber mehrere Zeilen gehen
>
> A: Das ist die Antwort
>
> Auch hier kann man mehrere Zeilen haben
>
> Oder **sogar** die komplette Markdownsyntax nutzen
```

Abbildung 1: Ein Beispiel der genutzten Karteikartensyntax

In den verschiedenen Umgebungen sieht es wiefolgt aus:



GitHub GitLab HedgeDoc

Abbildung 2: Rendering der in Abb. 1 definierten Syntax

Als Alternativansätze wurden folgende Syntaxmöglichkeiten betrachtet:

Durch Horizontallinien getrennte Karteikartenseiten:
Diese Syntax würde wiefolgt aussehen:

```
---
Q: Dies ist eine Frage

Sie kann ueber mehrere Zeilen gehen
---
A: Das ist eine Antwort

Auch hier kann man mehrere Zeilen haben

Oder **sogar** die komplette Markdownsyntax nutzen
---
```

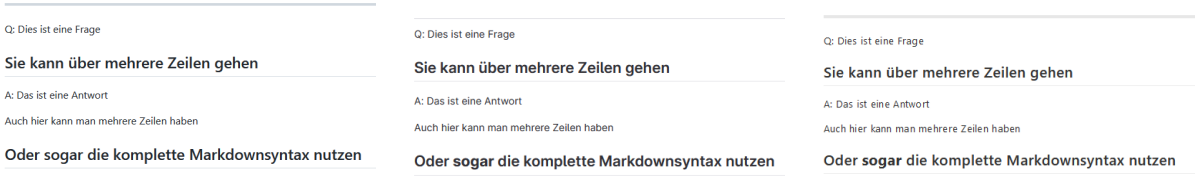
Abbildung 3: Ein Beispiel einer alternativen Karteikartensyntax

Während diese Syntax sehr leserlich in Markdownform und sehr einfach zu schreiben ist, hat sie mehrere Probleme.

Die Syntax kann nicht in der ersten Zeile benutzt werden, da sie Kollisionen mit der sogenannten Front Matter, ein auf YAML basierendes Metadatenformat innerhalb der Markdowndatei, hat. Front Matter wird sowohl von GFM, GLFM als auch HFM unterstützt.

Zudem ist es sehr einfach syntaktische Fehler zu machen, indem der Nutzer die letzte Horizontallinie weglässt.

Zuletzt ist das visuelle Rendering sehr unintuitiv, insbesondere da ohne freie Zeile die horizontale Linie in eine Überschrift umgewandelt wird



GitHub GitLab HedgeDoc

Abbildung 4: Rendering der in Abb. 3 definierten Syntax

Karteikarten in Tabellenform:
Diese Syntax würde wiefolgt aussehen:

```
| Question                          | Answer                                |
| --------------------------------- | --------------------------------------|
| Wofuer steht DFA?                 | Deterministischer endlicher Automat   |
| Welche Laufzeit hat Quicksort durchschnittlich? | O(n log n)            |
```

Abbildung 5: Beispiel für eine tabellenbasierte Karteikartensyntax

Diese Syntax sieht gerendert sehr lesbar aus.

Question	Answer
Wofuer steht DFA?	Deterministischer endlicher Automat
Welche Laufzeit hat Quicksort durchschnittlich?	O(n log n)

GitHub

Question	Answer
Wofuer steht DFA?	Deterministischer endlicher Automat
Welche Laufzeit hat Quicksort durchschnittlich?	O(n log n)

GitLab

Question	Answer
Wofuer steht DFA?	Deterministischer endlicher Automat
Welche Laufzeit hat Quicksort durchschnittlich?	O(n log n)

HedgeDoc

Abbildung 6: Rendering der in Abb. 5 definierten Syntax

Jedoch hat auch diese Syntax einige Nachteile. Zu erst eignet sich diese Syntax nicht für die Anwendung durch literate Programming, da hierdurch nur viele einzeilige Tabellen zwischen dem Paragraphen entstehen würden. Zudem ist diese Art von Syntax sehr aufwändig für den Endnutzer, sofern dieser kein WYSIWYG-Editor nutzt. Zuletzt ist es weder möglich, eine Antwort über mehrere Zeilen zu definieren, noch sinnvoll Bilder einzubinden.

Karteikarten durch XML Objekte:
Diese Syntax würde wiefolgt aussehen:

```
<anki>
  <q>
    Dies ist eine Frage

    Sie kann ueber mehrere Zeilen gehen
  </q>
  <a>
    Dies ist die Antwort

    Auch hier kann man mehrere Zeilen haben

    Oder sogar die komplette Markdownsyntax nutzen
  </a>
</anki>
```

Abbildung 7: Beispiel für eine XML-basierte Karteikartensyntax

Der Inhalt hiervon wird leider von GitHub gar nicht gerendered, weswegen es für unseren Anwendungsfall nicht nutzbar ist. In GitLab und HedgeDoc werden die Tags nicht angezeigt, jedoch ist der Inhalt noch sichtbar.

2.2 Unterstützung von LaTeX-Formeln

Die Desktopversion von Anki ermöglicht bereits die Darstellung von LaTeX-Formeln in Karteikarten. Hier wird beim ersten Anzeigen einer Karteikarte ein Bild der Formel mit einer lokal installierten LaTeX-Engine gerendert und in der Anki-eigenen Mediendatenbank gespeichert. Die Formel in der Karteikarte wird durch das gerenderte Bild ersetzt. Somit muss jede Formel nur einmal gerendert werden. Insbesondere wird die Mediendatenbank über die Cloud synchronisiert. Somit reicht es, dass das erste Gerät eine LaTeX-Intallation hat.

Die Mobilappliaktionen wie AnkiDroid oder die iOS-Version haben jedoch keine Unterstützung für LaTeX. Dies stellt normalerweise kein Problem dar, da bei der Karteikartenerstellung über die Desktopapplikation bereits die Formel gerendered wird. Da der Karteikartengenerator hier jedoch eigenständig zu Anki funktioniert, muss in Rahmen dieser Applikation auch die Formelgeneration übernommen werden.

Die Formelumwandlung ist jedoch nicht trivial. Das Problem hierbei sind die unterstützten Ausgabeformate:

- **Portable Document Format (PDF):** PDF ist ein sehr komplexes Dateiformat. Es hat viele, sehr umfangreiche Standards und wird von nur wenig Applikationen korrekt implementiert. Somit ist die Rasterisierung von PDF von Applikationen sehr fehleranfällig.
- **Device independent file format (DVI):** DVI ist ein selten genutztes, obskures Seitenformat. LaTeXist die einzige große Applikation, welche noch DVI als Ausgabeformat unterstützt, da Knuth es als formell unterstütztes Ausgabeformat von TeXdefinierte. Dementsprechend wenig Applikationen unterstützen DVI.
- **Encapsulated PostScript (EPS):** EPS ist ein für Drucker entwickeltes Vektorgrafikformat. Somit unterscheidet es sich sowohl von der Funktionalität als auch von der Nutzerschaft stark von pixelbasierten Bildformaten wie PNG.

Im folgenden wird eine Taxonomie der Konvertierungmöglichkeiten präsentiert:

- **ImageMagick:** ImageMagick [24] ist ein Open-Source-Programm zur Bearbeitung von Raster- und Vektorgrafiken. Dieser Ansatz ist auch die etablierte Lösung des TeX-Stackexchange [25]. Sie ist simpel, hat jedoch zwei Nachteile. Einerseits benötigt man sowohl LaTeXals auch ImageMagick als externe Abhängigkeit; andererseits benötigt sie manuelle Anpassungen der Sicherheitsrichtlinien, was root voraussetzt.
- **PDFium:** PDFium [26] ist die C++ PDF Bibliothek von Chromium [27]. Da man hierfür jedoch den kompletten Chromium-Browser benötigt, ist der Aufwand zu hoch. Die Kompilation von Chromium benötigt mindestens 8GB RAM sowie 100GB Festplattenspeicher zur Kompilation und braucht auf älterer Hardware mehrere Stunden.
- **KaTeX:** KaTeX [28] ist eine in Javascript geschriebene LaTeX-Implementation, welche auf Webseiten eingebunden werden kann um Formeln darzustellen. Sie wird unter anderem auch von GitHub und GitLab genutzt. Jedoch würde dies eine LaTeX-Engine wie Deno [29] oder Node.js [30] vorraussetzen, welche sehr komplex einzubinden wäre.
- **dvisvgm:** Zuletzt gibt es `dvisvgm` [31], welches PDF, DVI und EPS-Dokumente in SVG-Vektorgrafiken umwandeln kann. SVG ist das häufigst genutzte Vektorgrafikformat, für welches viele triviale PNG Konvertierungsmöglichkeiten existieren.

In unserem Fall haben wir uns für `dvisvgm` entschieden. Es ist eine Laufzeitabhängigkeit, jedoch in jeder kompletten TeXLive- sowie MikTeX-Installation vorhanden. Dies bedeutet, dass wir keine Abhängigkeiten haben, welche nicht auch von Anki vorrausgesetzt werden. Die von `dvisvgm` erstellen SVG-Grafiken werden dann durch `resvg` [32] zu PNG exportiert.

Als Syntax wurde sich `$$f(x)=x$$` für LaTeXentschieden[1]. Dies entspricht auch der KaTeX-Syntax, welche sowohl von HedgeDoc, als auch von GitHub oder GitLab genutzt wird.

2.3 Commitbasierte Kartengeneration durch Continuous Integration (CI)

Im Rahmen der Continuous Integration (CI) Softwarepraktik bieten alle großen Git-Repository-Hostinganbieter sogenannte Workflows an. Hierbei wird zu spezifischen Events wie Commits oder Pull Requests ein Container ausgeführt, welcher automatisiert vordefinierte Aufgaben erfüllen kann. Unter den typischen Aufgaben von CI zählen das Analysieren von Source Code nach Best-Practises, dem sogenannten Linting, der Ausführung automatisierter Tests sowie der Erstellung von Kompilaten.

In unserem Fall ermöglichen wir durch die Nutzung von CI-Workflows eine automatische Erstellung von Ankidecks aus Gitrepositories. Dieser Anwendungszweck ist so konzipiert das die Nutzerschaft gemeinsam

[1] Um das parsing simpel zu halten restriktieren wir diese Syntax vorerst nur auf einzelne Zeilen.

kollaborativ via Git eine Ordnerstruktur an Anki-Markdowndokumenten erstellt. Im Rahmen dieses Praktikums wurden sowohl für GitLab [18] als auch für GitHub [19] Repositoryvorlagen erstellt. Im folgenden wird ohne Beschränkung der Allgemeinheit die Funktionsweise anhand von Gitlab vorgestellt:

1. Ein Nutzer macht eine Änderung an dem Repository und pusht den Commit auf den GitLab-Server. Die GitLab-Instanz startet einen OCI-Ubuntucontainer welcher das Repository im current working directory (CWD) eingebunden hat.
2. Das System wird via `apt` aktualisiert und es werden folgende Abhängigkeiten installiert:
   - curl (Zur Installation von Rust via `rustup` benötigt)
   - git (Für den Download des Konvertierungsprogramms benötigt)
   - LaTeX(Zur Darstellung von Formeln zur Laufzeit benötigt)
   - C-Compiler via `build-essential` (Zur Kompilation benötigt)
   - pkg-config (Zur Kompilation benötigt)
   - OpenSSL (Zur Kompilation benötigt)
   - Rust-Compiler (Zur Kompilation benötigt)
3. Das Konvertierungsprogramm wird in das CWD via git gecloned und kompiliert.
4. Das kompilierte Programm wird auf den Ordner mit den Anki-Markdowndokumenten angewandt.
5. Das resultierende Karteikartenarchiv wird als Artifact den Nutzern bereitgestellt.

3 Funktionsweise

Die Funktionweise ist in mehrere, disjunkte Schritte aufgeteilt, welche im folgenden betrachtet werden:

1. **Markdownkarteikarten einlesen:** Innerhalb des per Programmargument übergebenen Pfades werden rekursiv nach allen Dateien gesucht, welche eine `.anki.md` Dateiendung haben. Alle gefundenen Dateien werden eingelesen und Karteikarten werden alle nach vorherig definierter Syntax geschriebenen extrahiert.
2. **Formeln durch Bilder ersetzen:** Um die Mobilunterstützung zu sichern werden bereits zur Erstellung der Decks alle LaTeX-Formeln zu PNG-Bildern umgewandelt, die dann in normaler Markdownsyntax als Ersatz der Formel eingesetzt werden.
3. **Bilder lokal speichern:** Um die Möglichkeit der Offlinenutzung zu ermöglichen, werden sämtliche extern referenzierten Bilder lokal gespeichert.
4. **Markdownkarteikarten in HTML umwandeln:** Mit der `comrak`-Bibliothek [33] werden die Markdownkarteikarten in äquivalente HTML-Dokumente umgewandelt.
5. **HTML-Karteikarten und Bilder in Anki-Archiv speichern:** Die generierten Karteikarten werden nun als Ankideckarchiv gespeichert. Die interne Struktur entspricht hierbei der genutzten Ordnerstruktur.

4 Diskussion

4.1 Audio- und Videounterstützung

Diese Version unterstützt weder Audio- noch Videoeinbettung. Hierfür gibt es zwei Gründe:

1. **Schlechte Anki-Unterstützung:** Offiziell ist die Einbettung von Audio- und Videodateien in Anki-Karteikarten möglich, jedoch ist der Support dafür nicht ausreichend. Wenn Audiodateien ohne weitere Anpassungen eingebunden werden, starten sie direkt nach dem initialen Laden der Karte, ohne das Steuerungsmöglichkeiten zum Pausieren oder zur Lautstärkeeinstellung vorhanden sind. Bei Videodateien öffnet sich ein zusätzliches Pop-up-Fenster, das einen mpv-Player startet und das Video automatisch abspielt. Diese Probleme kann man jeodoch umgehen, in dem man manuell die HTML-Elemente in die Karte integriert und die Medien nur indirekt referenziert. Im Rahmen dieser Arbeit wurde auch ein Proof of Concept diesbezüglich erstellt [34].

2. **Keine native Markdownunterstützung:** Weder CommonMark noch GFM[2] oder GLFM definieren einen offiziellen Weg Audio- und Videodateien einzubinden. HedgeDoc bietet nur die Möglichkeit für YouTube- oder Vimeovideos [23]. Keiner der genannten Standards unterstützen Audiodateien. Somit gibt es einen kanonischen Weg Video oder Audio syntaktisch sinnvoll einzubinden.

4.2 Geplante Weiterentwicklung und Einsatz der Software

Obwohl die erste Version erfolgreich entwickelt wurde, gibt es noch einige geplante Folgeentwicklungen:

- Die Anki Markdown-Spezifikation sollte erweitert werden, um mehr Medientypen auf eine kompatible Weise abzudecken. Dies sollte in einem offenen, RFC-ähnlichen Prozess geschehen, wo alle Interessen- und Nutzergruppen einbezogen werden.

- Die Anwendung sollte auch für Nicht-Informatikstudenten zugänglicher werden. Dies bedeutet die Entwicklung einer Benutzeroberfläche und die Erschaffung einer insgesamt einfacheren Benutzererfahrung. Außerdem sollte eine angemessene Dokumentation erstellt werden, worin viele WYSIWYG-Tools für die Erstellung von Flashcards vorgestellt werden sollten.

Neben diesem Punkten ist geplant, sowohl mit der Fachgruppe Informatik sowie einer Anki-Projektgruppe eines Medizinstudentenvereins Testläufe zu gestalten, in welchen iterativ die Software und die dazugehörigen Arbeitsprozesse verbessert werden.

[2] GFM unterstützt Videos, jedoch nur in Pull Requests und Issues [35].

5 Literaturverzeichnis

Literatur

[1] John Gruber. *Daring Fireball: Markdown*. URL: https://daringfireball.net/projects/markdown/ (besucht am 02. 03. 2023).

[2] The HedgeDoc developers. *HedgeDoc - The best platform to write and share markdown*. URL: https://hedgedoc.org/ (besucht am 02. 03. 2023).

[3] GWDG. *HedgeDoc - Collaborative markdown notes*. URL: https://pad.gwdg.de/ (besucht am 02. 03. 2023).

[4] John MacFarlane. *Pandoc*. URL: https://pandoc.org/ (besucht am 02. 03. 2023).

[5] Hakim El Hattab et al. *The HTML presentation framework — reveal.js*. URL: https://revealjs.com/ (besucht am 02. 03. 2023).

[6] Damien Elmes et al. *Anki - powerful, intelligent flashcards*. URL: https://apps.ankiweb.net/ (besucht am 02. 03. 2023).

[7] Ankitects. *Anki for desktop computers*. URL: https://github.com/ankitects/anki (besucht am 02. 03. 2023).

[8] Ankitects Ply Ltd. *AnkiMobile Flashcards*. URL: https://apps.apple.com/de/app/ankimobile-flashcards/id373493387 (besucht am 02. 03. 2023).

[9] The Qt Company. *QTWebEngine*. URL: https://wiki.qt.io/QtWebEngine (besucht am 02. 03. 2023).

[10] Ankitects. *AnkiWeb*. URL: https://ankiweb.net/about (besucht am 02. 03. 2023).

[11] Ankitects. *Add-ons for Anki 2.1*. URL: https://ankiweb.net/shared/addons/ (besucht am 02. 03. 2023).

[12] D. E. Knuth. "Literate Programming". In: *The Computer Journal* 27.2 (Jan. 1984), S. 97–111. ISSN: 0010-4620. DOI: 10.1093/comjnl/27.2.97. eprint: https://academic.oup.com/comjnl/article-pdf/27/2/97/981657/270097.pdf. URL: https://doi.org/10.1093/comjnl/27.2.97.

[13] Bart Childs. *An Introduction to the WEB Style of Literate Programming*. URL: http://www.literateprogramming.com/IntroC.pdf (besucht am 02. 03. 2023).

[14] Project Jupyter. *Project Jupyter — Home*. URL: https://jupyter.org/ (besucht am 02. 03. 2023).

[15] GWDG. *Jupyter - GWDG - IT in der Wissenschaft*. URL: https://www.gwdg.de/application-services/jupyter (besucht am 02. 03. 2023).

[16] Google. *Welcome To Colaboratory*. URL: https://colab.research.google.com/ (besucht am 02. 03. 2023).

[17] Lars Quentin. *Ankiding*. URL: https://github.com/lquenti/ankiding (besucht am 02. 03. 2023).

[18] Lars Quentin. *ankiding-ci-gitlab*. URL: https://gitlab.gwdg.de/lars.quentin/ankiding-ci-gitlab (besucht am 02. 03. 2023).

[19] Lars Quentin. *ankiding-ci-github*. URL: https://github.com/lquenti/ankiding-ci-github (besucht am 02. 03. 2023).

[20] John MacFarlane et al. *CommonMark*. URL: https://commonmark.org/ (besucht am 02. 03. 2023).

[21] Github. *GitHub Flavored Markdown*. URL: https://github.github.com/gfm/ (besucht am 02. 03. 2023).

[22] GitLab. *GitLab Flavored Markdown (GLFM)*. URL: https://docs.gitlab.com/ee/user/markdown.html (besucht am 02. 03. 2023).

[23] HedgeDoc. *HedgeDoc Flavored Markdown*. URL: https://docs.hedgedoc.org/references/hfm/ (besucht am 02. 03. 2023).

[24] ImageMagick Studio LLC. *ImageMagick – Convert, Edit, or Compose Digital Images*. URL: https://imagemagick.org/ (besucht am 02. 03. 2023).

[25] Malabarba et al. *Compile a LaTeX document into a PNG image that's as short as possible.* URL: https://tex.stackexchange.com/questions/11866/compile-a-latex-document-into-a-png-image-thats-as-short-as-possible/ (besucht am 02. 03. 2023).

[26] Chromium Project. *PDFium.* URL: https://pdfium.googlesource.com/pdfium/+/master/README.md (besucht am 02. 03. 2023).

[27] Chromium Project. *Chromium.* URL: https://www.chromium.org/chromium-projects/ (besucht am 02. 03. 2023).

[28] Khan Academy. *KaTeX – The fastest math typesetting library for the web.* URL: https://katex.org/ (besucht am 02. 03. 2023).

[29] Deno Land Inc. *Deno — A modern runtime for JavaScript and TypeScript.* URL: https://deno.land/ (besucht am 02. 03. 2023).

[30] OpenJS Foundation. *Node.js.* URL: https://nodejs.org/en/ (besucht am 02. 03. 2023).

[31] Martin Gieseking. *dvisvgm.* URL: https://dvisvgm.de/ (besucht am 02. 03. 2023).

[32] Yevhenii Reizner. *RaztFalcon/resvg: An SVG rendering library.* URL: https://github.com/RazrFalcon/resvg (besucht am 02. 03. 2023).

[33] Asherah Connor. *kivikakk/comrak: CommonMark + GFM compatible Markdown parser and renderer.* URL: https://github.com/kivikakk/comrak (besucht am 02. 03. 2023).

[34] Lars Quentin. *lquenti/anki-media-no-autoplay.* URL: https://github.com/lquenti/anki-media-no-autoplay (besucht am 03. 03. 2023).

[35] Lauren Brose. *Video uploads now available across GitHub.* URL: https://github.blog/2021-05-13-video-uploads-available-github/ (besucht am 03. 03. 2023).