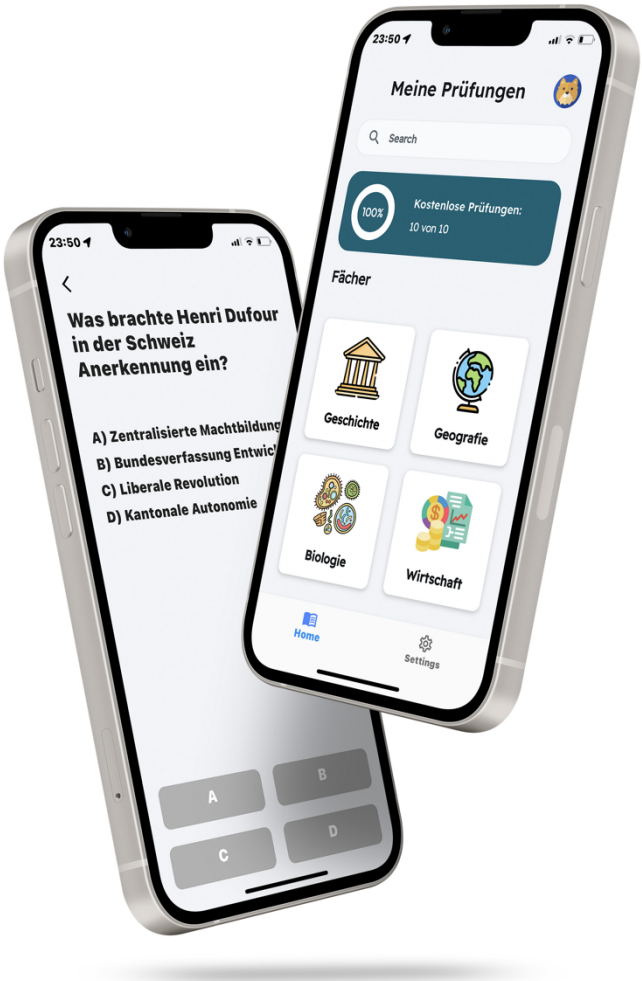
KANTONSSCHULE
SCHAFFHAUSEN

Entwicklung einer KI unterstützten Lern-App.



Jonas Riesen

Maturaarbeit

Informatik

Dr. Rainer Steiger

05.12.2023

Abstract

Die vorliegende Maturarbeit beschreibt den Prozess der Entwicklung einer Lern-App namens «Studyflow». Die App soll mit Hilfe einer Künstlichen Intelligenz die Lern-Dossiers und Lernziele ihrer Benutzerinnen und Benutzer analysieren und passgenaue Zusammenfassungen, Lernkarten und Testprüfungen erstellen.

Im praktischen Teil der Arbeit wurde die App von Grund auf entworfen, designt und programmiert. Im Anschluss wurde die App auf den App-Plattformen von Apple und Google veröffentlicht. In einem Praxistest mit einer Schulklasse wurde die Funktionalität und Benutzerfreundlichkeit getestet und weiter verfeinert.

Diese Arbeit bildet den theoretischen Teil der Arbeit. In ihr werden die zu Grunde liegenden Technologien erläutert und die einzelnen Arbeitsschritte beschrieben, die für die Umsetzung des App-Projekts nötig waren.

Download

Studyflow, die App, um die es in meiner Maturarbeit geht, wurde pünktlich fertiggestellt und kann hier heruntergeladen werden.

Google Play

1 Einleitung

1.1 Motivation und Ziel

Das Lernen in der Phase nach dem Präsenzunterricht hat sich seit Jahrzehnten nicht verändert. Man sitzt allein mit den Lernzielen, einem Stapel Lernunterlagen (dem Dossier), allenfalls mit einem Buch am Schreibtisch und schreibt eine Zusammenfassung um sich die wichtigsten Fakten, Zahlen und Zusammenhänge besser merken zu können. Im Hinterkopf steht stets die Frage, was an der bevorstehenden Prüfung gefragt werden könnte. Dann wird die Zusammenfassung so lange gelesen, bis man den Lernstoff verinnerlicht und die Lernziele erreicht hat.

Eventuell erstellt man noch Lernkarten oder man hat Glück und findet alte Prüfungsfragen. Dieser Prozess ist in Abbildung 1 dargestellt.

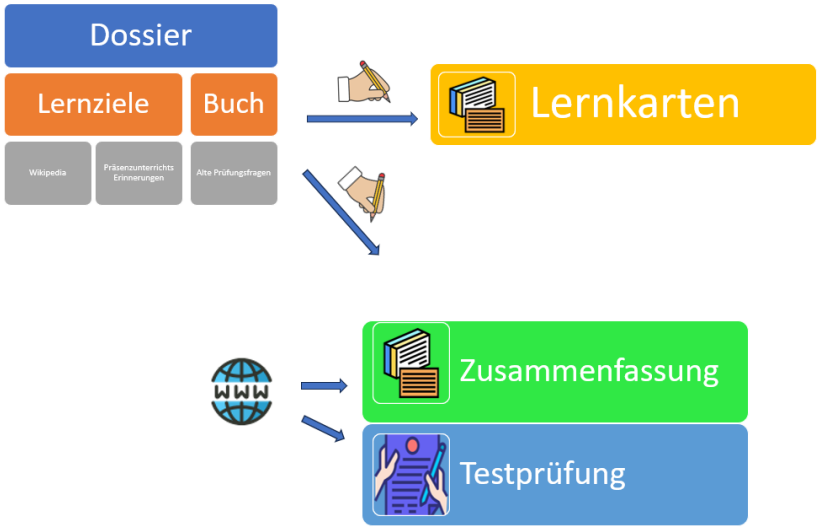


Abbildung 1 (Eigene Grafik)

Natürlich gibt es dazu heute auch viele Lern-Apps wie z.B. Quizlet, mit dem man Lernkarten digital erstellen kann. Auch kann es sein, dass man Zusammenfassungen im Internet findet, Beispiele wären die Simple Club Tutorials oder die Studyflix Zusammenfassungen.

Nur: Die Lern-Apps müssen alle aufwändig mit Inhalt gefüllt werden, was viel Zeit braucht. Bei den Zusammenfassungen ist das Problem, dass oft das Lernniveau falsch ist oder aber die Lernziele nicht oder nur teilweise abgebildet werden oder der eigenen Lehrperson andere Fragestellungen wichtig sind.

Ein weiterer grosser Nachteil herkömmlicher Lernsoftware ist, dass die Antworten auf Fragen klar definiert sein müssen. So liegt neben Multiple Choice und dem Einfügen klarer Lösungsworte nicht mehr drin. Mit der Nutzung eines Large Language Models (LLM) hingegen werden Textlösungen möglich.

Das Large Language Model (**LLM**) ist eine Art von Algorithmus der künstlichen Intelligenz (KI), der Deep-Learning-Techniken und massiv grosse Datensätze verwendet, um neue Inhalte zu verstehen, zusammenzufassen, zu generieren und vorherzusagen. Der Begriff generative KI ist eng mit LLMs verbunden, die in der Tat eine Art von generativer KI sind, die speziell für die Generierung textbasierter Inhalte entwickelt wurde. [1]

Da ich schon länger Software entwickle und in jüngster Zeit häufig mit KI resp. den der KI zugrundeliegenden LLM-Sprachmodellen auseinandersetzte, stellte ich mir die Frage, ob es nicht möglich wäre, wie in Abbildung 2 abgebildet, mit Hilfe einer LLM eine eigene App zu entwickeln. Diese App sollte Lerninhalte selbständig zusammenfassen und den Lernzielen der jeweiligen Stufe angepasste Lernkarten erstellen.

Als Basis werden das eigene Dossier und die eigenen Lernziele genutzt und anschliessend passgenaue Zusammenfassungen, Lernkarten und Testprüfungen erstellt.

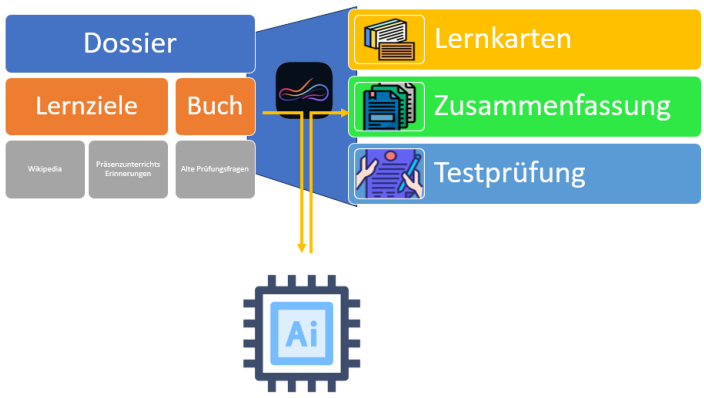


Abbildung 2 (Eigene Grafik)

[1] Vgl. https://www.computerweekly.com/de/definition/Large-Language-Model-LLM

2 Design der Benutzeroberfläche

Als **Benutzeroberfläche** bezeichnet man die Schnittstelle, mit der ein Mensch mit einem Computer grafisch interagieren kann.

Ein gutes Design der **Benutzeroberfläche** ist für die Akzeptanz der User absolut zentral.

Aber gutes Design bedeutet nicht nur, dass die App schön aussieht, es ist genauso wichtig, dass diese einfach und intuitiv zu bedienen ist. Dies bezeichnet man als Benutzerfreundlichkeit oder User Experience Design (UX-Design). Eine App, die nur gut aussieht, aber nicht gut zu bedienen ist, fällt bei den Benutzern schnell durch. Es ist daher wichtig, eine gute Balance zwischen der grafischen Gestaltung (UI-Design) und der Benutzerfreundlichkeit zu finden.

2.1 Zielgruppen Analyse

Es ist wichtig, das Design und die Bedienung einer App auf die Zielgruppe abzustimmen. So wird sichergestellt, dass die App den Nutzern wirklich gefällt und sie diese problemlos anwenden können. Meine App richtet sich an Schülerinnen und Schüler zwischen 15 und 20 Jahren. Für diese Altersgruppe ist ein Design wichtig, welches einheitlich und somit leicht zu bedienen ist. In dieser Altersgruppe ist ein entscheidender Punkt zudem, dass Inhalte schnell geladen werden. [2]

2.2 Design Prozess

Der Design Prozess verläuft über die folgenden drei Schritte, die nachfolgend genauer beschrieben werden:

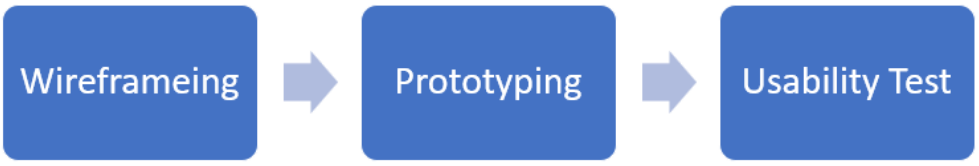


Abbildung 3 Übersicht Design Prozess (Eigene Grafik)

[2] Vgl. https://www.usability.ch/news/teenager-usability-website-design-fuer-jugendliche.html

2.2.1 Wireframe

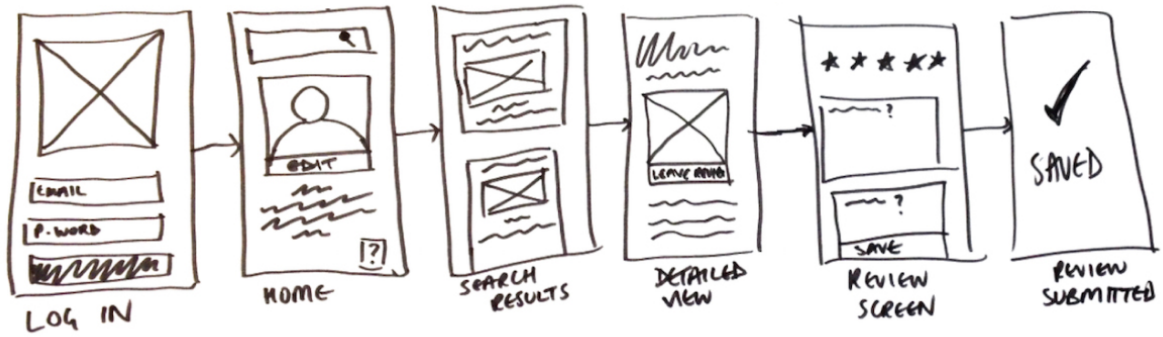


Abbildung 4 https://shorturl.at/cmKV4

Der erste Schritt beim Designen einer Benutzeroberfläche ist das sogenannte Wireframing. Beim Wireframe konzentriert man sich ausschliesslich auf die Struktur des Designs. Man stellt sich Fragen wie «welches Element sollte an welchem Ort sein?» oder «wie sollte die Nutzerführung aussehen?». Farben und Textinhalte sind zu diesem Zeitpunkt noch kein Thema. Meist wird der Wireframe von Hand skizziert. Eine gute Übersicht bietet die Abbildung 4.

2.2.2 Das Wireframe

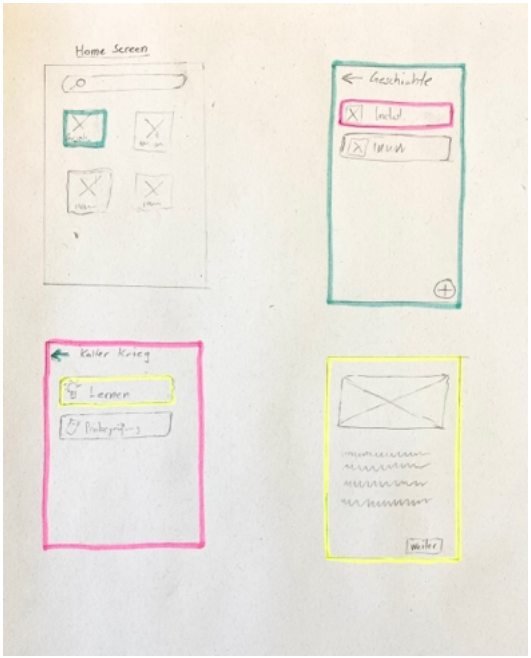


Abbildung 5 Erste Skizze für die App (Eigenes Bild)

Bei der Konzeption der Grundstruktur meiner App habe ich unterschiedliche Skizzen eingesetzt, die trotz der Digitalisierung mit Papier und Bleistift angefertigt wurden. Ein einfacher und logischer Aufbau wurde durch die Reduktion der Elementanzahl pro Bildschirm erreicht. Sobald bestimmte Elemente als intuitiv und logisch identifiziert wurden, flossen sie in neue Skizzen ein. Nicht passende Elemente unterlagen einem Optimierungs- und Weiterentwicklungsprozess von Entwurf zu Entwurf. Nach mehreren Iterationen entwickelte ich so ein passendes Design, das eine intuitive Benutzerführung ermöglicht.

2.2.3 Prototyping

Nach dem Wireframing folgt das Prototyping, bei dem die handgefertigten Skizzen digitalisiert werden. Hierfür werden häufig Tools wie Figma oder Adobe XD verwendet. Viele Prototyping-Tools bieten neben dem Design auch die Möglichkeit, die Benutzerführung interaktiv zu testen, indem eine umfangreiche App-Simulation durchgeführt wird.

Abbildung 6 Farbpalette https://shorturl.at/esFI4

Nachfolgend darauf wird Farbe ins Spiel gebracht, indem eine Farbpalette von maximal fünf Farben ausgewählt wird. Diese Farben werden dann konsequent einzelnen Elementen in der App zugewiesen. So habe ich beispielsweise die Farbe Orange konstant für den „Zurück"-Button verwendet. Alle Buttons, Schriften und andere Elemente erhalten entsprechend einheitliche Farben. Dies ermöglicht eine erneute Überprüfung des App-Aufbaus auf seine logische und intuitive Gestaltung hin, auch durch die Betrachtung der Farbgebung.

2.2.4 Das Prototyping

Zur Digitalisierung von Wireframes habe ich Figma[3] verwendet, da diese Software für Schüler und Schülerinnen kostenfrei ist und sich in der Praxis bereits bewährt hat. Die digitalen Skizzen entstehen durch den Einsatz grafischer Formen wie Rechtecke, Kreise und Linien. Dieser Prozess ist auf der nächsten Seite in Abbildung 7 sichtbar.

[3] Vgl. www.figma.com

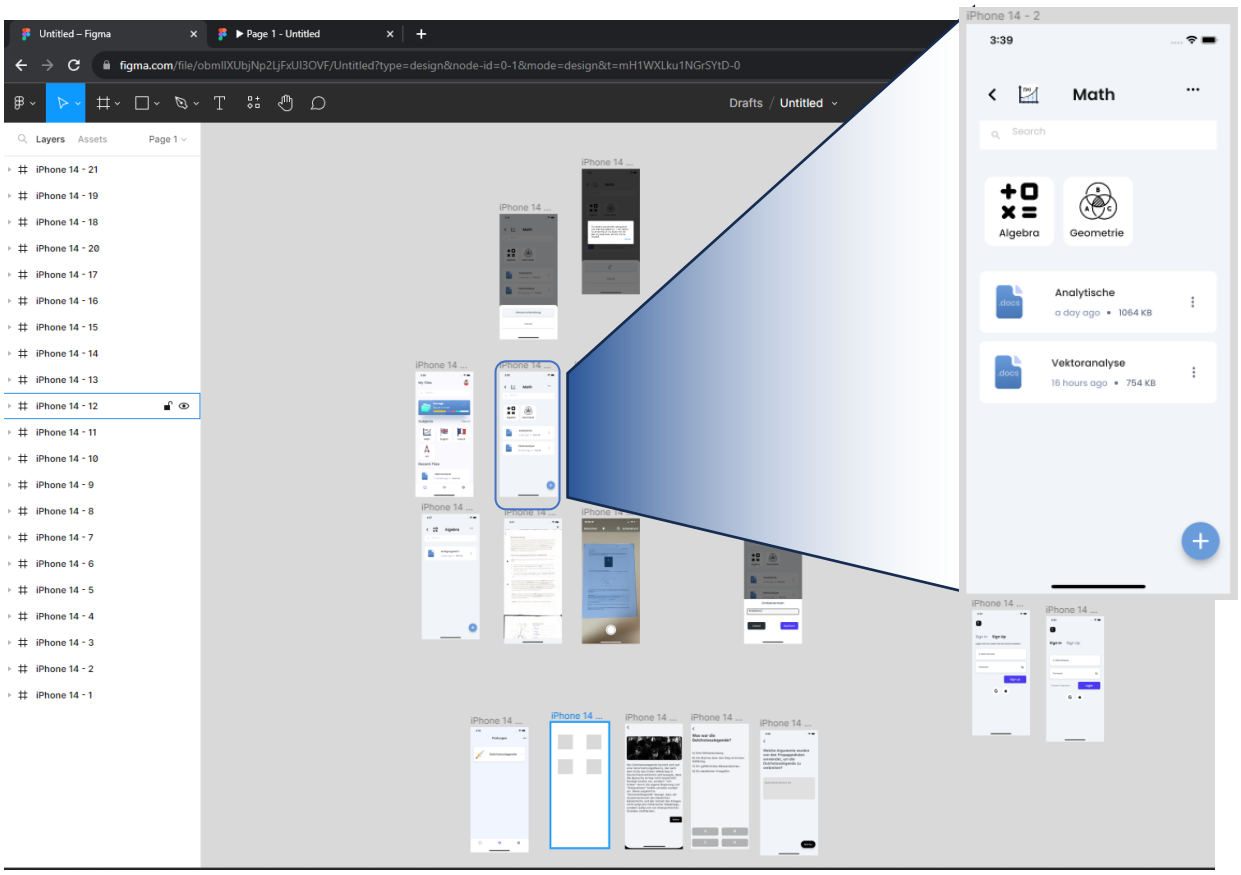


Abbildung 7 Figma (Eigenes Bild)

Jedes einzelne Layout resp. jede Screen-Ansicht wird nachgebaut, die grafischen Elemente platziert. Figma ermöglicht die Erstellung interaktiver Benutzeroberflächen aus einem Design. Dabei können Bedingungen hinzugefügt werden, wie zum Beispiel: „Beim Klicken auf dieses grafische Objekt sollte der folgende Bildschirm angezeigt werden". So entsteht ein ganzes Netz mit den Funktionen der App. Da meine App auch Fotos und Dokumente importieren kann, mussten sehr viele Abläufe und Prozesse definiert werden. Am Ende lag eine Simulation der App vor, die hier

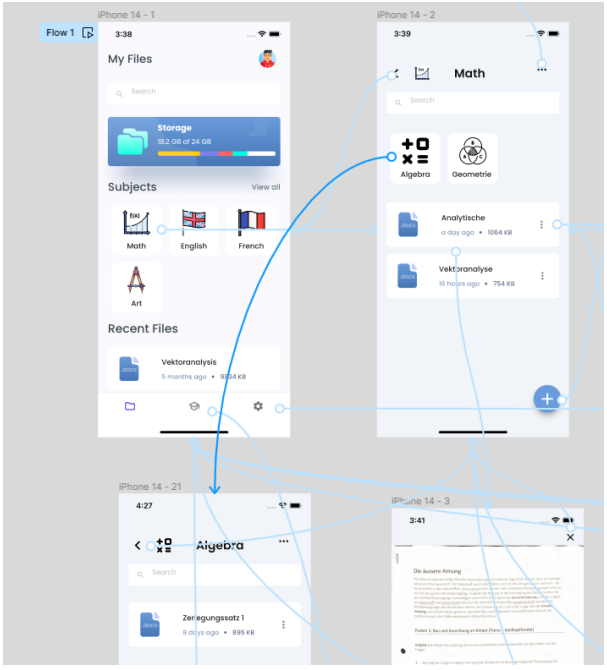


Abbildung 8 Figma Abläufe (Eigenes Bild)

https://shorturl.at/yG678 getestet werden kann.

Farbauswahl:

Zur Optimierung der Farbauswahl und zur Inspiration habe ich die Webseite Coolors[4] genutzt, um eine passende Farbpalette zu entwickeln. Diese Palette enthält idealerweise keine zu kräftigen Farben, um ein schlichtes und zeitloses Design zu unterstützen. Anschliessend wird das ausgewählte Farbschema auf das digitalisierte Wireframe übertragen. Meine Farbpalette ist in Abbildung 10 ersichtlich.

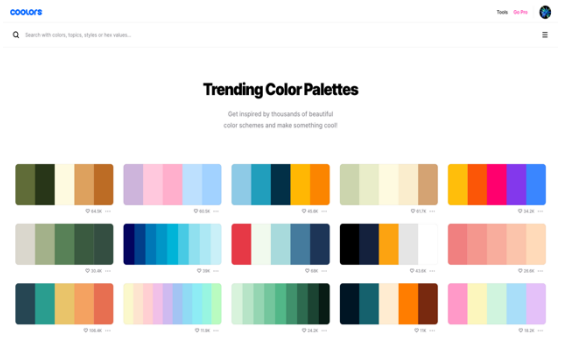


Abbildung 9 https://coolors.co/palettes/trending

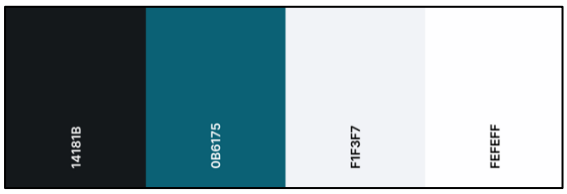


Abbildung 10 Meine Farbpalette

2.2.5 Usability-Testing

Beim Usability-Testing geht es darum, den beim Prototyping erstellten Prototyp zu testen und daraufhin das Design anzupassen, um es noch intuitiver zu gestalten. Denn bis anhin war das Design darauf ausgelegt, was der Programmierer selbst als intuitiv empfand.

Abbildung 11 https://shorturl.at/erOV8

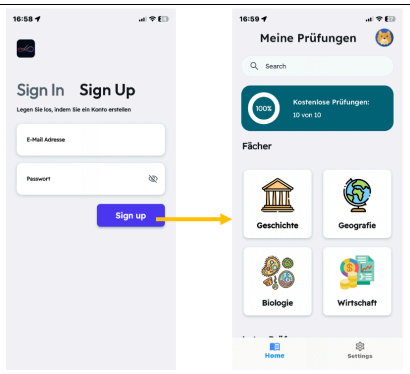
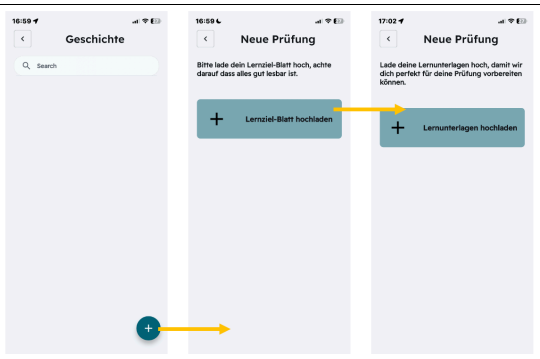
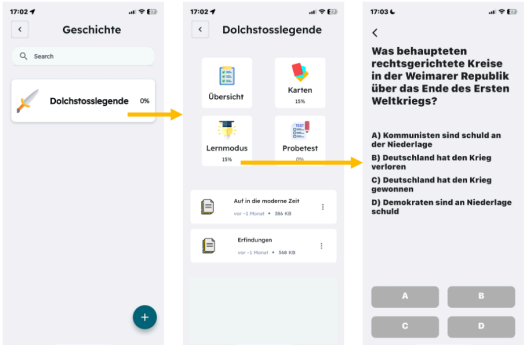
Neben der Suche nach Verbesserungsmöglichkeiten, bietet Usability Testing weitere Vorteile wie z.B. das bessere Kennenlernen des Verhaltens und der Präferenzen der Zielgruppe. Ein Usability Test ist relativ einfach aufgebaut: Es gibt einen Vermittler und einen Teilnehmer. Der Vermittler gibt dem Teilnehmer eine Aufgabe, welche der Teilnehmende ausführen soll, während er vom Vermittler beobachtet wird. Der Teilnehmer gibt fortlaufend Feedback. Der Informationsfluss ist in Abbildung 11 zu sehen.

[4] Vgl. www.coolors.co

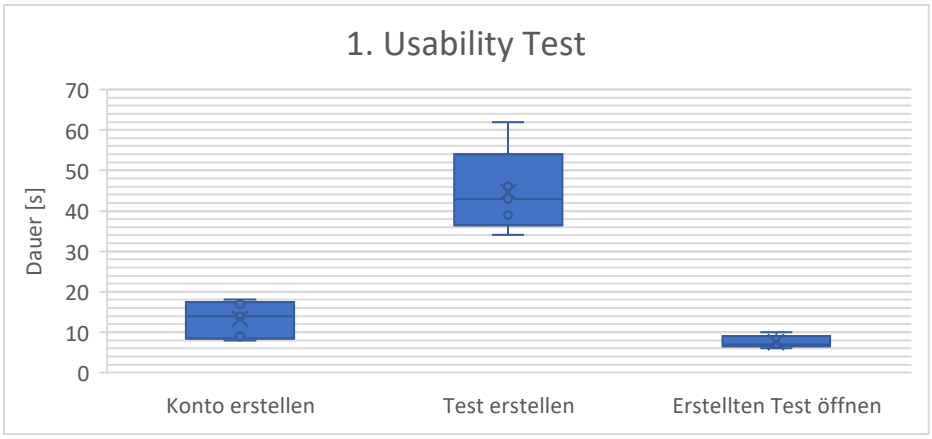
2.2.6 Mein Usability-Test

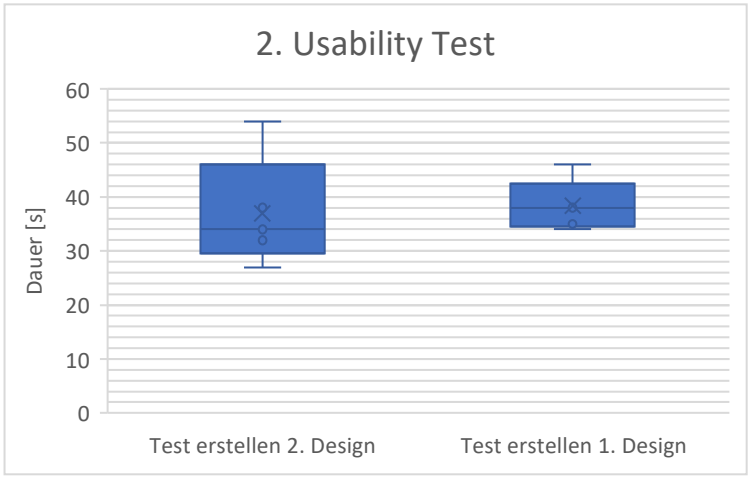
Für die Überprüfung des Designs habe ich einen Versuchsaufbau gewählt, bei dem eine Testperson ein Handy mit dem interaktiven Design vor sich liegen hat. Der Proband erhält eine Aufgabe, deren korrekte Ausführung zeitlich gemessen wird. Als Testpersonen wurden fünf Schüler und Schülerinnen im Alter von 15 bis 20 Jahren ausgewählt. Folgende drei Aufgaben wurden ihnen im Rahmen des Tests gestellt:

(In der linken Spalte wird jeweils die Aufgabe dargestellt, während in der rechten Spalte visualisiert ist, welche Schritte der Testnutzer auszuführen hatte.)

1. Neues Konto erstellen	
2. Neuen Test erstellen, inklusive hochladen von Lernzielen sowie Lernblättern	
3. Erstellten Test öffnen	

Die Auswertung des Usability-Tests ergab folgende Ergebnisse: Im Durchschnitt benötigten die Tester nur 13,2 Sekunden, um ein Konto zu eröffnen, was als zufriedenstellendes Resultat gewertet wurde. Bei der Aufgabe „Test erstellen" lag der Mittelwert bei 44,8 Sekunden, mit einer Spannweite von 28 Sekunden. Eine derart hohe Spannweite deutet darauf hin, dass die Nutzerführung nicht für alle Tester optimal war. Die Aufgabe „Erstellten Test öffnen" wurde im Schnitt in 7,8 Sekunden erledigt, mit einer geringen Spannweite von 4 Sekunden. Um den Prozess der Testerstellung intuitiver zu gestalten, wurde basierend auf dem Feedback aller Testnutzer die Navigation und die Erklärungstexte zur Hochladung der Lernunterlagen überarbeitet. Nach einer erneuten Phase des Wireframings und der Digitalisierung des Designs mit Figma wurden fünf neue Tester herangezogen, um die Änderungen zu bewerten. Die grafische Auswertung zeigte eine Verringerung des Mittelwerts auf 38,5 Sekunden und eine Reduzierung des Spannbereichs auf 12 Sekunden. Mit diesen Ergebnissen empfand ich die Nutzerführung als deutlich verbessert und erachtete es nicht mehr für notwendig das Design weiter anzupassen.





3 App Development

Ein Ziel bei der Programmierung einer App ist es, dem Design Leben und somit Funktionalität einzuhauchen. Bei einem Projekt dieser Grösse, ist es wichtig nicht einfach «darauf los zu programmieren», sondern eine genaue Struktur und Vereinheitlichung aufzustellen. Dieses Vorgehen spart vor allem beim Finden eines Bugs und beim Refractoring sehr viel Zeit.

Beim Refractoring geht es darum, den Code zu verbessern und zu optimieren, ohne dabei das Verhalten der Software zu verändern. [5]

Führt man sich vor Augen, dass allein für den «Smartphone-Teil», dem sog. Frontend, meiner App «Studyflow» über 14'000 Zeilen Programmcode nötig sind, wird klar, dass eine klare Struktur zentral ist. Zum Vergleich: Der Programmcode füllt 250 A4-Blätter, oder ein halbes Paket Kopierpapier.

Zur Veranschaulichung sieht man hier den Code, der allein hinter dieser einzelnen Kachel, welche in Abbild 12 zu sehen ist, steckt.

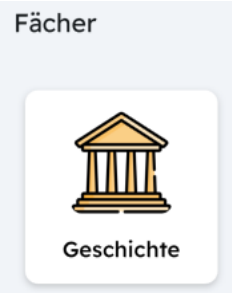


Abbildung 12 (Eigenes Bild)

```
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(0.0, 20.0, 0.0, 20.0),
  child: Container(
    width: double.infinity,
    height: 360.0,
    decoration: BoxDecoration(),
    child: Column(
      mainAxisSize: MainAxisSize.max,
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        Row(
          mainAxisSize: MainAxisSize.max,
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            InkWell(
              splashColor: Colors.transparent,
```

[5] Vgl. https://shorturl.at/gmuIZ

```
            focusColor: Colors.transparent,
            hoverColor: Colors.transparent,
            highlightColor: Colors.transparent,
            onTap: () async {
              logFirebaseEvent(
                  'HOME_PAGE_Container_68790f3u_ON_TAP');
              logFirebaseEvent('Container_navigate_to');

              context.pushNamed(
                'subject_page',
                queryParameters: {
                  'title': serializeParam(
                    'Geschichte',
                    ParamType.String,
                  ),
                }.withoutNulls,
              );
            },
            child: Container(
              width: 150.0,
              height: 150.0,
              decoration: BoxDecoration(
                color: FlutterFlowTheme.of(context)
                    .secondaryBackground,
                boxShadow: [
                  BoxShadow(
                    blurRadius: 4.0,
                    color: Color(0x33000000),
                    offset: Offset(0.0, 2.0),
                  )
                ],
                borderRadius: BorderRadius.circular(10.0),
              ),
              child: Column(
                mainAxisSize: MainAxisSize.max,
                mainAxisAlignment:
                    MainAxisAlignment.spaceEvenly,
                children: [
                  ClipRRect(
                    borderRadius: BorderRadius.circular(8.0),
                    child: Image.asset(
                      'assets/images/2318366.png',
                      width:
                          MediaQuery.sizeOf(context).width *
                              0.2,
                      fit: BoxFit.cover,
                    ),
                  ),
                  Text(
                    'Geschichte',
                    style: FlutterFlowTheme.of(context)
                        .bodyMedium
                        .override(
                          fontFamily: '<Readex Pro',
                          color: Colors.black,
                          fontSize: 17.0,
                          fontWeight: FontWeight.w500,
                        ),
                  ),
                ],
              ),
            ),
          ),
        ),
```

Der Text der Box (Geschichte) findet sich erst ganz am Schluss des Programmcodes. Der Rest des Codes beschreibt nur das Aussehen dieser Kachel.

3.1 Projekt Aufbau, Übersicht und Begrifflichkeiten

In der Softwareentwicklung wird zwischen Frontend- und Backend-Entwicklung unterschieden. Die Frontend-Entwicklung befasst sich mit der Umsetzung des Designs und dem Erstellen einer Benutzeroberfläche, die alle grafischen Elemente und deren Funktionen beinhaltet. Ihre Aufgabe ist es, dem Benutzer die Interaktion mit der visuellen Seite der Applikation zu ermöglichen. Im Gegensatz dazu ist die Backend-Entwicklung für die Schaffung der Funktionalität verantwortlich, die von der Frontend-Anwendung genutzt wird, um die App vollständig bedienbar zu machen. Das Backend umfasst Server, Datenbanken und Anwendungslogik. Meist läuft das Frontend auf einem Notebook oder einem Smartphone, das Backend auf einem Server in einem Rechenzentrum. Damit das Frontend mit dem Backend kommunizieren kann, wird eine Schnittstelle, die **API**, entwickelt. In Abbildung 13 ist visuell die Trennung zwischen Frontend und Backend ersichtlich.

Eine **API** (Application Programming Interface) ist eine Programmierschnittstelle, die die Kommunikation zwischen verschiedenen Softwareanwendungen ermöglich. [6]

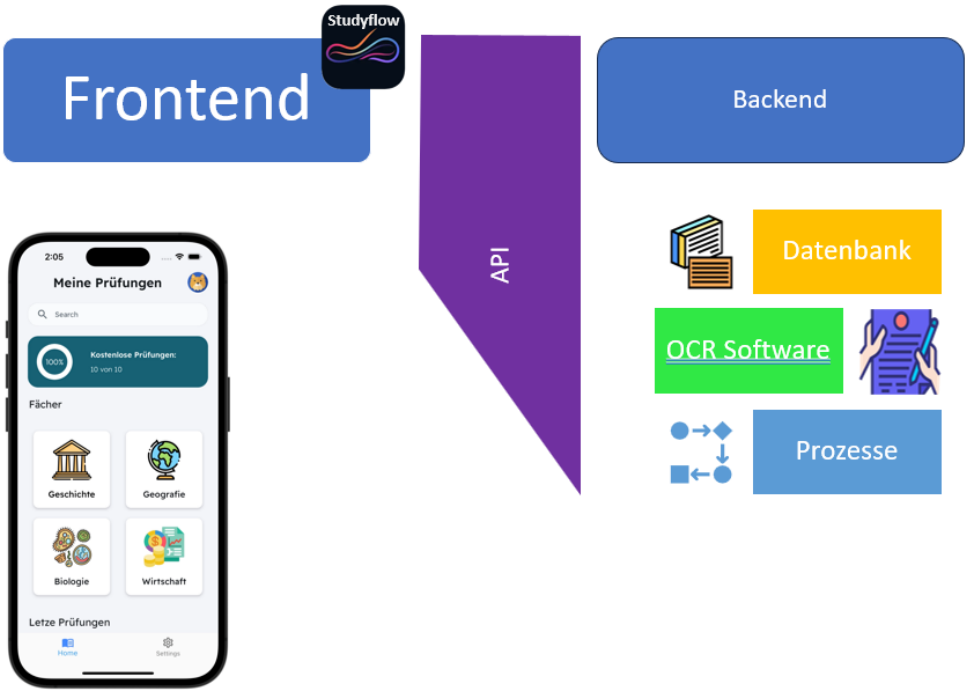


Abbildung 13 (Eigene Grafik)

[6] Vgl. https://www.cloudflare.com/de-de/learning/security/api/what-is-an-api/#:~:text=An%20application%20programming%20interface%20,transmit%20data%20to%20another%20program

3.2 API-First

Wenn man zeitgleich am Frontend und Backend entwickelt, mündet dies meist in einer inkonsistenten Kommunikation oder einer Inkompatibilität zwischen den beiden Systemen. Speziell bei grösseren Projekten kann dies rasch zu immensen Problemen führen, da es viel schwieriger wird die Software überhaut zu entwickeln, zu warten und neue Features einzubauen.

Um dies zu verhindern, verwendet man das Prinzip des API-First, dies bedeutet, dass man zuerst entscheidet, wie die Endpunkte, Datenstrukturen und Verhaltensweisen aussehen sollen. Dadurch ist bereits vorgegeben, wie die Kommunikation zwischen dem Backend und dem Frontend aussehen wird. Es definiert klare Verträge und Erwartungen. Und genau, weil es klare Verträge und Erwartungen gibt, ermöglicht es einem gleichzeitig am Frontend wie auch am Backend zu arbeiten.

Zuerst versucht man alle möglichen Anwendungsfälle zu identifizieren und potenzielle Endpoints ausfindig zu machen. Ein Endpoint ist ein digitaler Übergabeort, welcher durch die API zugänglich ist und sowohl Antworten entgegennehmen wie auch senden kann. Die identifizierten Endpoints werden in einem API-Vertrag festgehalten. Dies wird schriftlich in einem Dokument, dem sog. Vertrag, festgehalten. Üblicherweise definiert man, auf welche Weise der Endpunkt erreichbar ist. Abbildung 14 zeigt ein Ausschnitt des API-Vertrages für «Studyflow».

```
API-Vertrag für die Erstellung einer neuen Prüfung

Endpunkt: Erstellung einer neuen Prüfung
- URL: `POST /new_exam`
- Methode: POST
- Beschreibung: Dieser Endpunkt ermöglicht die Erstellung einer neuen Prüfung mit spezifische
Lernzielen und Lernmaterialien.

Anfragebody:
{
  "user_id": 238482018385,
  "exam_id": "87he3",
  "learning_goals": "",
  "learning_material": ""
}

Erfolgreiche Antwort (200 OK):
{
  "status": "In Bearbeitung"
}

Fehlerantworten:
- 400 Bad Request: Wenn der Anfragebody fehlerhaft ist und nicht verarbeitet werden kann.
- 500 Internal Server Error: Wenn ein unerwarteter Fehler auf dem Server auftritt und die
Anfrage nicht bearbeitet werden kann.
```

Abbildung 14 (Eigenes Bild)

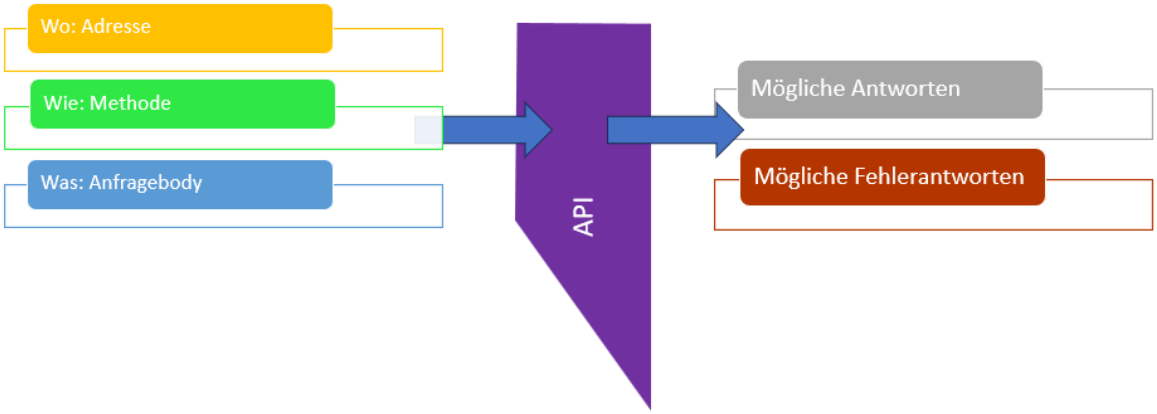


Abbildung 15 (Eigene Grafik)

In meinem Beispiel:

Adresse: /new_exam

Methode: Post (was immer das ist)

Anfragebody: User, Examen ID, Lernmaterial

Erfolgreiche Antworten: 200 = alles gut

Fehlerantworten: 400 = Anfragebody weist Fehler auf / 500 = interner Serverfehler.

Abbildung 13 zeigt visuell welche Informationen für ein erfolgreicher API-Call nötig sind und wie die API darauf reagieren kann.

3.3 Frontend

Im Folgenden wird die Entwicklung des Frontends von «Studyflow» detailliert beschrieben, einschliesslich der verschiedenen notwendigen Schritte.

3.3.1 Auswahl Programmiersprache

Beim Entwerfen einer mobilen Applikation steht man als Entwickler zu Beginn oft vor der Entscheidung, für welches Betriebssystem diese entwickelt wird. In der Schweiz, wie Abbildung 16 zeigt, führt iOS, dicht gefolgt von Android. Um ein breites Publikum zu erreichen, war es für mich sinnvoll, Applikationen für beide dominierenden Smartphone-Betriebssysteme zu entwerfen.

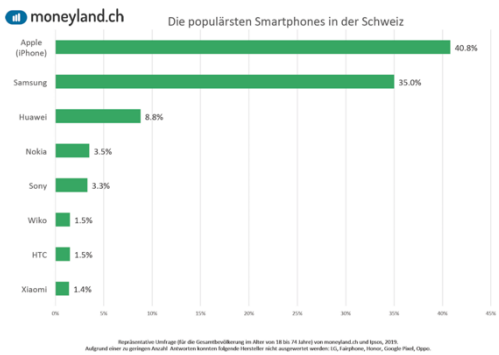


Abbildung 16 https://shorturl.at/jlAKL

Hierfür gibt es grundsätzlich zwei Ansätze: Das Erstellen von zwei separaten Apps für jedes Betriebssystem (sog. native Apps), oder die Verwendung einer Cross-Plattform-Programmiersprache wie «React Native» oder «Flutter», um mit einem einzigen Codebase für mehrere Betriebssysteme entwickeln zu können[7]. Um Ressourcen und Zeit effizient einzusetzen, entschied ich mich für den Cross-Plattform-Ansatz. Es gibt jedoch mehrere solcher Cross-Plattform-Programmiersprachen auf dem Markt. Nach gründlichem Abwägen fiel die Wahl auf «Flutter», da es im Vergleich zu «React Native» mehrere Vorteile wie eine umfassendere Dokumentation[8] und eine bessere Performance aufweist.

3.3.2 Frontend Entwicklung

Bei der Frontend Entwicklung wird nun der zuvor entworfene Prototyp in die Realität umgesetzt. Bei den Prototypen konnte ich das Design schnell durch graphische Elemente nachbauen und hatte schon eine teils funktionsfähige Probeversion der App.

Das Erstellen der endgültigen App basierend auf dem Prototyp darf zeitlich nicht unterschätzt werden. In der Softwareentwicklung können nicht einfach grafische Elemente durch Mausbewegungen verschoben werden, jede Position muss mühsam in Relation zu etwas anderem definiert werden. Dadurch wird sichergestellt, dass sich das Design adaptiv an verschiedene Displaygrössen anpasst. So hat allein das Frontend von «Studyflow» über 14'0000 Zeilen Code. Um den Aufbau ein wenig besser zu veranschaulichen, ein Beispiel Screen, welcher auf Abbildung 17 zu sehen ist.

Der Screen «Meine Prüfungen» mit den Fächern generiert 5 API-Calls und über 700 Zeilen Code.

Abbildung 17 (Eigenes Bild)

[7] https://shorturl.at/eiqzE

[8] https://docs.flutter.dev/

3.4 Backend

Das Backend ist in meinem Projekt verantwortlich für die Verarbeitung und Speicherung der durch «Studyflow» bearbeiteten und hochgeladenen Lernunterlagen.
Darunter fällt eine OCR, die Textbereinigung, die Segmentierung des Textes und schliesslich den eigentlichen Output: die Erstellung der Lerninhalte. Einen Überblick über den Funktionsumfang bietet Abbildung 18.

OCR steht für "Optical Character Recognition" und ist eine Technologie, die es ermöglicht, gedruckten oder handgeschriebenen Text aus Bildern oder gescannten Dokumenten in bearbeitbaren, digitalen Text umzuwandeln. [9]

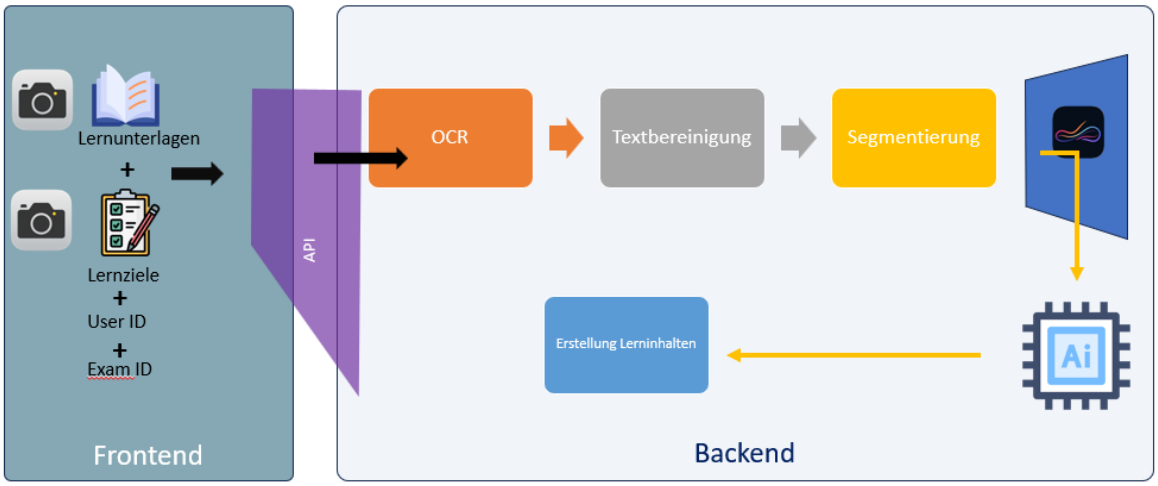


Abbildung 18 (Eigene Grafik)

Das Backend von «Studyflow» bekommt von der API bei der Erstellung eines neuen Tests die folgenden vier Parameter: Die User ID, die Exam ID in Textform, das Learning Material

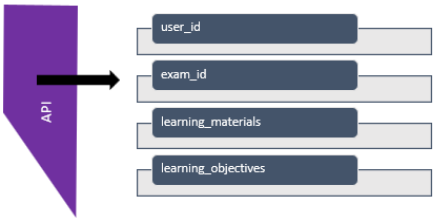


Abbildung 19 (Eigene Grafik)

[9] https://www.ibm.com/blog/optical-character-recognition/#

(Lernunterlagen) und die Learning Objects (Lernziele) als mehrere Fotographien, die User mit dem Smartphone erstellen.

Die Smartphone Fotos, welche der Nutzer aufgenommen hat, werden noch im Frontend in das sog. Base 64 Format kodiert.

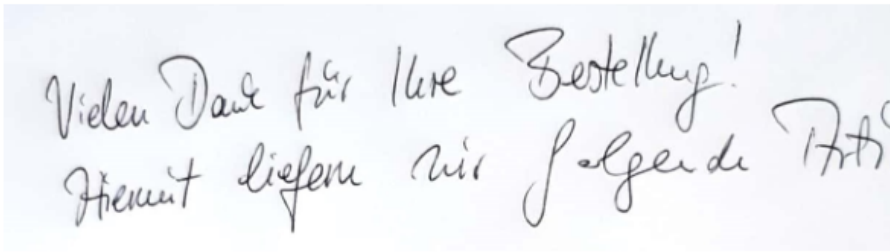
Base 64 übersetzt Bilder in einen Text mit normalen Buchstaben (ASCII Code), damit diese über das Internet zum Backendserver versendet werden können. Beispiel: Das Symbol für Geschichte sieht in sehr tiefer Auflösung im Base 64 Format wie folgt aus:

	-Ars4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsEAAA7BAbiRa+0AAAELSURBVChTZ-ZBPLwNRFMV/86Y1aKI1EiyqEqTBB7ARqS2JBTtbX6pLS0tCk-NhYICpNZqSE+ATt2LXEhJl5z5s/oeUk972TnHNPbo6hNPiDUIIw0unHP/O906RxWscsTLK6ucfyYjVT+sydTpuT-gzpR+4qNWpVAWZzfvTI2u8bWzi62bSOSFY33XpeW28AfXeDQyXPmRnxSwGle4/t+4hk44+Jon27vDcMQCCGQUUilMsPK+nai/yTHyBkheT1WTqZchIxYZqZqPfvxPI/by2NkGKB0clyEU-hHPDy7lpRrjpeJv8qNzQyCHsIplJqbnKU3NMaz5x5fk5amVeAZu1vWi9GNmE-VHct+Zp3fAN97pnsmK7sBcAAAAASUVORK5CYII=

Bevor das Backend mit der Verarbeitung der Daten beginnt, wird zunächst sichergestellt, dass die Werte nicht korrupt sind, also unverfälscht und unbeschädigt. Zudem wird geprüft, ob sie grundlegende Kriterien erfüllen, wie beispielsweise die Existenz eines Nutzers mit der entsprechenden User ID. Falls ein Kriterium nicht erfüllt ist, wird dies über die API dem Frontend mitgeteilt, allenfalls wird die Übertragung wiederholt oder dem Benutzer eine Fehlermeldung angezeigt.

3.4.1 OCR

Hat alles seine Richtigkeit, werden die Fotos der Lernziele und der Lernunterlagen mithilfe eines OCR-Services zu Text umgewandelt. Bei der Entwicklung einer App kann nicht alles von Grund auf neu programmiert werden. Man kauft sich gewisse Dienste ein oder recherchiert, ob es kostenlose Software gibt, die für einzelne Arbeitsschritte bereits existiert. Bei der Auswahl einer OCR-Texterkennungslösung stehen mehrere Anbieter auf dem Markt zur Verfügung. Vision AI von Google stellt eine bevorzugte Wahl dar, da es eine schnelle Performance bei einem tieferen Preis bietet. Hier zeigte sich auch, dass nebst der eigentlichen Programmierung der App auch die Hintergrundkosten ein zentrales Thema bei der App-Programmierung sind. So generiert VISION AI Kosten von 1.50$ pro 1000 Dokumente[10]. Ich habe auch kostenlose Alternativen (Open Source), wie Tesseract OCR[11], getestet. Bei diesen entstehen keine direkt laufenden Kosten, und Tesseract könnte auf dem Backendserver von «Studyflow» installiert werden. Nur braucht Tesseract viel Serverressourcen, die dann beim Hoster der Backendserver zu Mehrkosten führen. Zudem sind die Resultate erheblich schlechter, wie in Abbildung 20 zu sehen ist. Tesseract schneidet deutlich schlechter bei Handschriften ab und erfordert ein Vielfaches an Verarbeitungszeit, was ebenfalls zu höheren Kosten führt. Deshalb verwendet Studyflow Vision AI hauptsächlich, da die Ergebnisse erheblich besser sind.



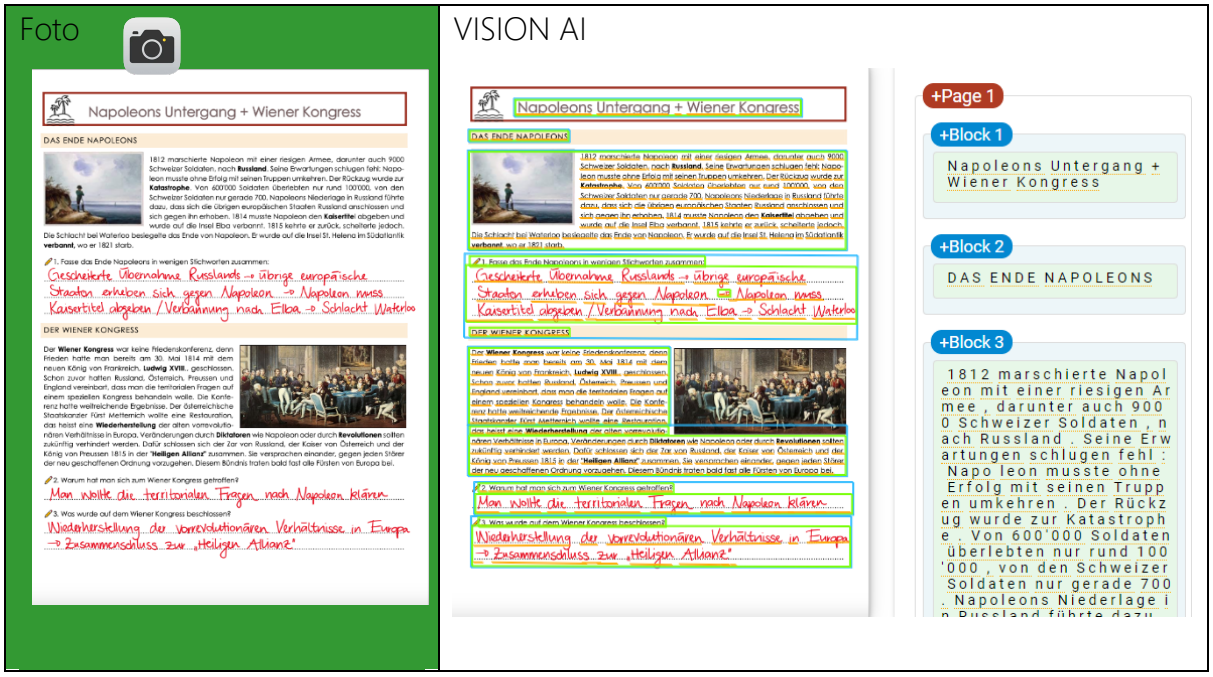
Tesseract:
Yoln Dos fr [hve Barklles
Shawst Lisle ai [e
LWACL

Google Vision AP:
Vielen Dank für Ihre Bestellung!
Fement liefern
Putke
nir de

Abbildung 20 Vergleich Tesseract und Google Vision API https://shorturl.at/twzV6

[10] https://cloud.google.com/vision/pricing?hl=de
[11] https://github.com/tesseract-ocr/tesseract

Die Bilddaten von «Studyflow» werden über eine weitere API auf die Server von VISON AI geladen und in einen Text übersetzt.



Selbst Handschriften erkennt Vision AI gut. Zudem – für ein Lern-App nicht unerheblich – prüft die Software den Text auf folgende Kategorien:

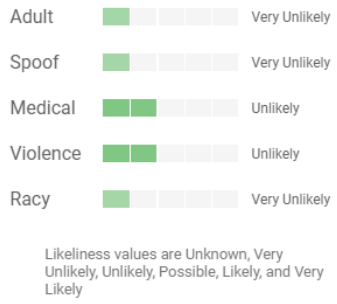


Abbildung 21 Screenshot Vision AI

Wenn Vision AI einen Text als problematisch einstuft, wird er von den nachfolgenden Verarbeitungsschritten ausgenommen, um eventuellen Missbrauch meiner App zu verhindern.

3.4.2 Textbereinigung

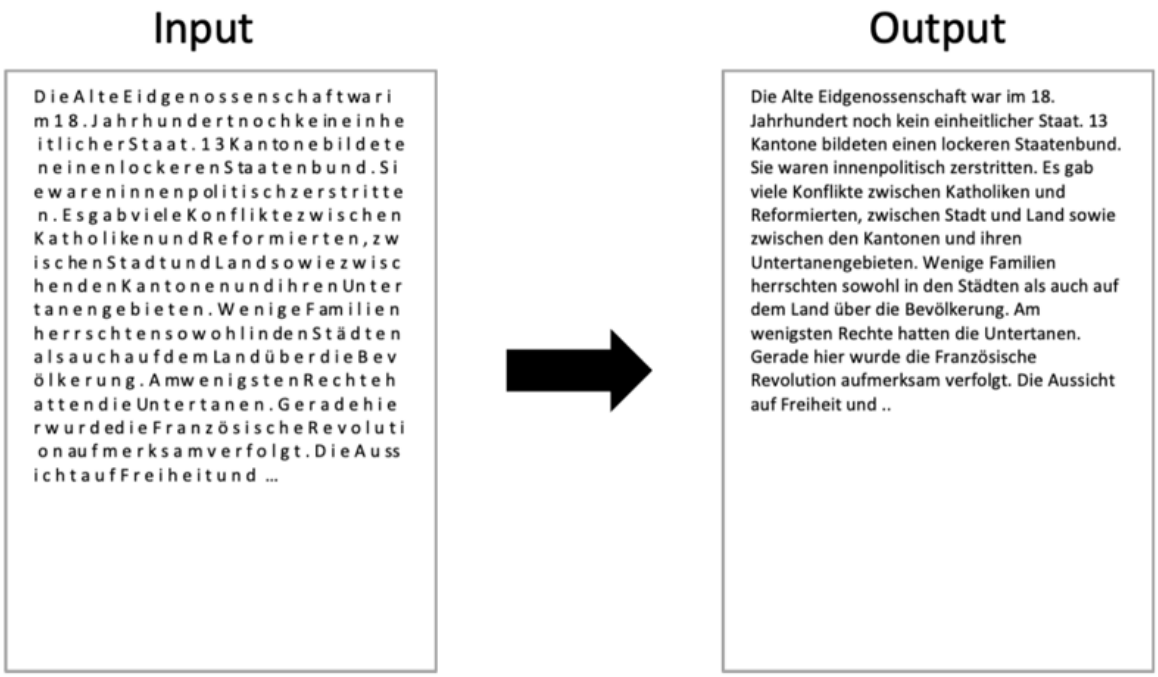


Abbildung 22 Textbereinigung (Eigene Grafik)

Wenn man sich den Output der OCR ein wenig genauer ansieht (Abbildungen 22) bemerkt man, dass die Qualität des Textes nicht wirklich gut ist, denn der Text beinhaltet viele orthographische Fehler. Für die Weiterverarbeitung des Textes ist es essenziell, diesen zunächst zu säubern. Diese Reinigungsaufgabe wird durch den Einsatz eines Large Language Models (LLM) bewältigt. Angesichts der Tatsache, dass der Reinigungsprozess nicht besonders komplex ist, ist die Verwendung eines kosteneffizienten Modells ratsam. Die Wahl fiel auf das Modell „text-davinci-003", das OpenAI bereits im Jahr 2020 herausgebracht hat. [12] Die Anwendung von „text-davinci-003" hat, wie sich herausstellte, eine signifikante Qualitätsverbesserung des bereinigten Textes bewirkt, wie in Abbildung 20 dargestellt.

3.4.3 Segmentierung

Für die Generierung von Zusammenfassungen, Lernkarten oder Aufgaben braucht es eine sehr leistungsstarke LLM, welche weiter noch beschrieben wird. Allerdings kann der

[12] https://platform.openai.com/docs/models/

bereinigte Text nicht «einfach so» an eine solche LLM weitergegeben werden, weil diese nicht viel Input Text bearbeiten können. So beträgt der maximale Input beim LLM «GPT-3.5-turbo» etwa 14 Seiten Text.

Die meisten LLM arbeiten nicht mit einer Anzahl Buchstaben oder Wörtern, sondern mit sogenanntem Token. Ein Token bezeichnet eine Textsequenz. Diese kann aus ganzen Wörtern, mehreren Wörtern oder aus Wortteilen bestehen.

Hier ein Beispiel:

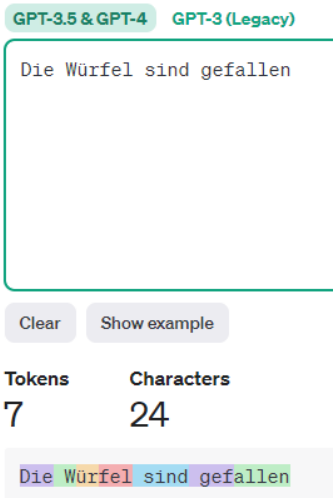


Abbildung 23 https://platform.openai.com/tokenizer

Farbig abgebildet sieht man die einzelnen Token.

Übrigens werden die Kosten einer LLM-Benutzung auf Token-Basis abgerechnet.

Abbildung 24

Da für eine Prüfung an der Kantonsschule ohne weiteres über 20 Seiten Lernunterlagen anfallen (mehr als 10'000 Tokens, zugelassen sind max. 8'000 Tokens), muss dieser Text vor der Übergabe an die LLM, die dann Lerninhalte erstellt, segmentiert werden.

Um dies zu ermöglichen, habe ich einen Segmentierungsalgorithmus entwickelt, der den gesamten Text zunächst in Segmente mit jeweils 2.000 Tokens unterteilt.

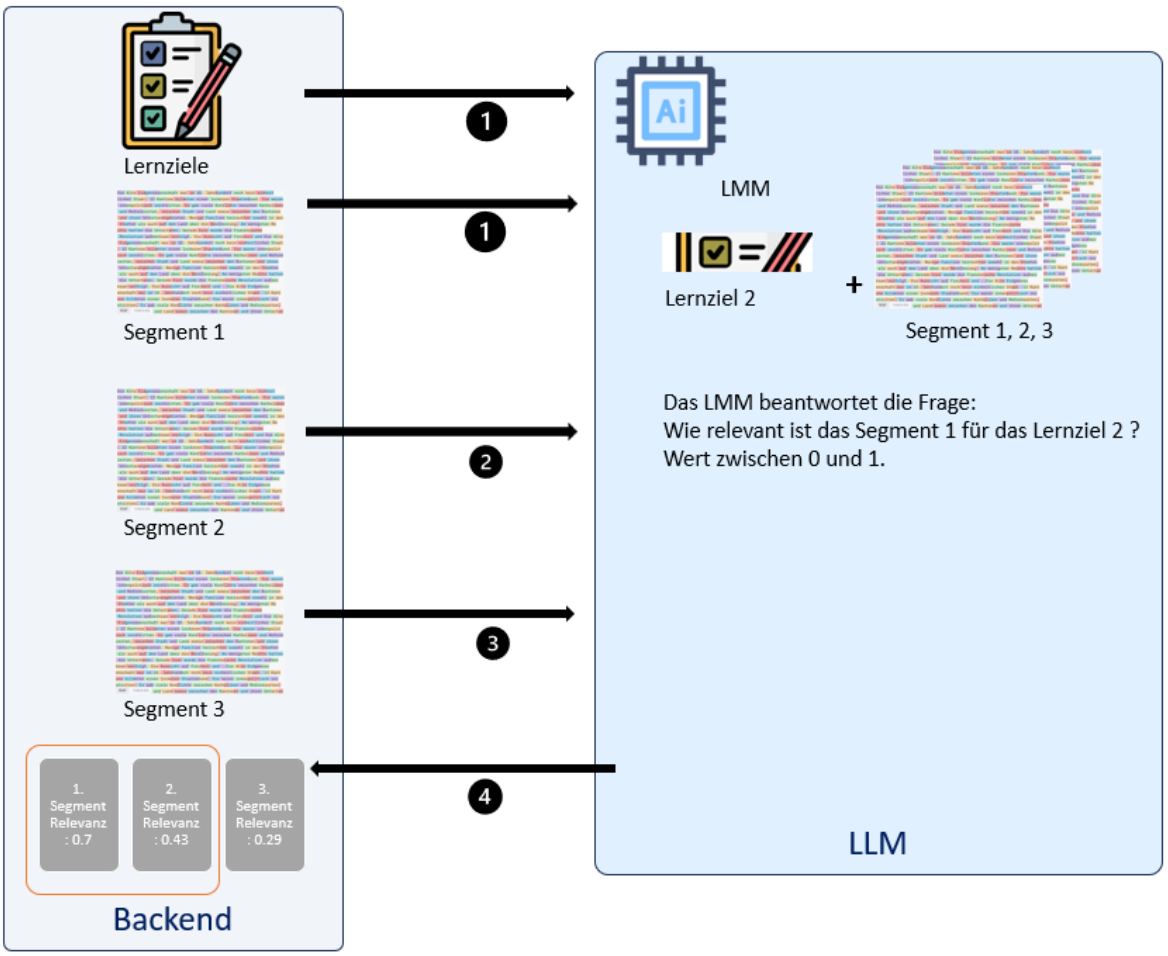


Abbildung 25 Segmentierungsalgorithmus (Eigene Grafik)

Diese Segmente werden nun zusammen mit einem Lernziel in die LLM (Schritte 1-3) geladen. Das LLM bewertet eigenständig die Relevanz jedes Segments im Hinblick auf das jeweilige Lernziel und trifft diese Entscheidung nach eigenem Ermessen. Anschliessend gibt es einen Wert zwischen 0 und 1 an das Backend zurück. Dieser Prozess ist in Abbildung 25 graphisch zu sehen. Das Backend nimmt die drei relevantesten Sequenzen und lädt diese gemeinsam in die LLM. Damit erreicht «Studyflow», dass die LLM nur mit den für das Lernziel relevanten Sequenzen arbeitet und die 8000er Token -Grenze nicht überschritten wird. Jetzt ist alles bereit für die Erstellung der Lerninhalte.

3.4.4 Erstellung von Lerninhalten

Nach der Segmentierung kommt das Herzstück des Backends zum Zug: die Erstellung von Lerninhalten, welche dann im Frontend dargestellt werden können. Das Backend kann drei Arten von Lernformate erstellen, wie in Abbildung 26 zu sehen ist. Um die User bestmöglich auf eine Prüfung vorzubereiten, wird ihnen die Theorie in kurzer Form zusammengefasst. Für das Training werden Lernkarten mit Multiple-Choice Aufgaben generiert. Zudem – und das ist wohl der grösste Vorteil von «Studyflow» – erhalten die User mögliche, an die Lernziele und Stufe adaptierte Prüfungsaufgaben direkt auf ihr Smartphone, die hervorragend auf eine reale Prüfung vorbereiten.

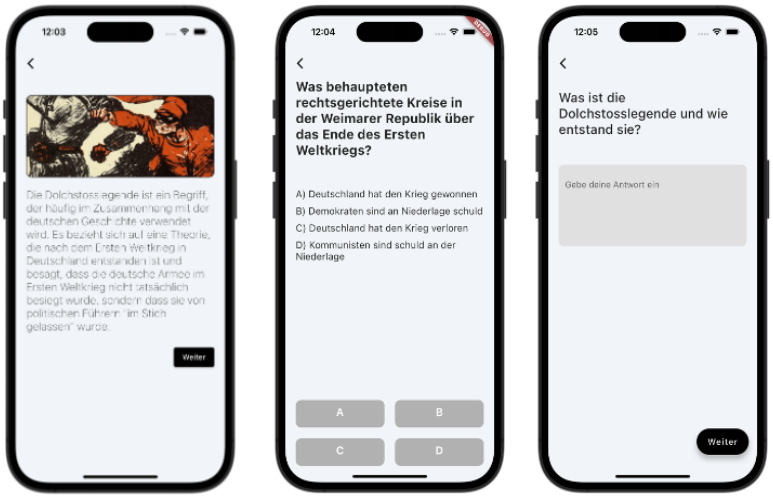


Abbildung 26: Lernformate: Zusammenfassung, Multiplechoice und Prüfgunsfragen

Diese drei Arten von Lerninhalten werden alle mithilfe eines LLM generiert. Dies ist jedoch nur möglich, da wir im vorherigen Schritt den Text segmentiert haben

Um mit einem LLM zu interagieren, gibt man ihm Anweisungen was es zu tun hat, sogenannte Prompts. Da die Qualität der Prompts massgeblich ist für die Qualität des Outputs, ist es wichtig, Prompts zu haben, welche möglichst gute Resultate liefern. Es gibt also gute und schlechte Prompts. Da dies nicht ganz einfach ist, gibt es Unternehmen welche Prompt Engineers einstellen. [13]

[13] https://prompt-engineering-jobs.com

Für das Backend von «Studyflow» sind gute Promtps ebenfalls sehr wichtig. Gibt man eine Frage online direkt in einem LLM ein, erhält man im Browser ein brauchbares Ergebnis, wie das Beispiel in Abbildung 27 zeigt.

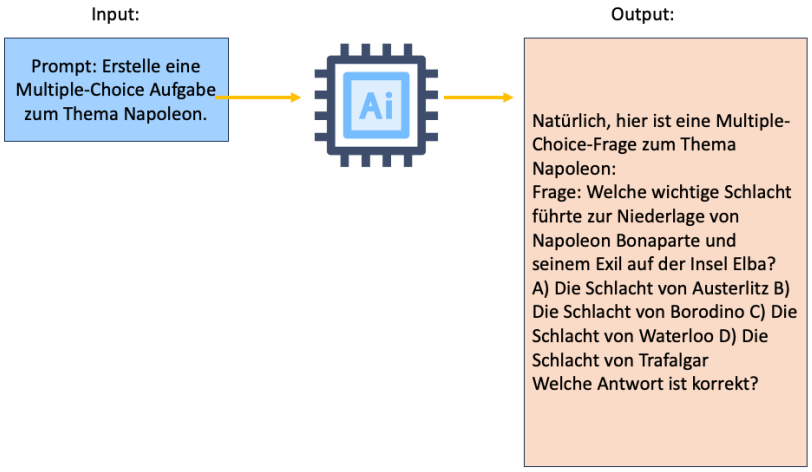


Abbildung 27 (Eigene Grafik)

Der Output enthält eine Frage und vier Auswahlmöglichkeiten, jedoch auch einen Einstiegssatz und eine Arbeitsanweisung am Schluss in der Form einer Frage. Für einen User, welcher sich über «chat.openai.com» Multiple-Choice Fragen generieren lassen will, ist dieses Ergebnis nutzbar, da er im Stande ist, aus dem Output die Frage und Auswahlmöglichkeit zu «erkennen».

Für eine Software, in diesem Fall meinem Backend, ist dieses Ergebnis jedoch nicht nutzbar, da das Backend nicht im Stande ist, die nötigen Informationen zu extrahieren. Es «merkt» nicht, welche Textteile zur Einleitung gehören, welches die Multiple Choice Fragen sind und welche Arbeitsanleitungen sind.

Wenn man den Prompt nun weiter spezifiziert, erreicht man immer brauchbarere Ergebnisse, wie in Abbildung 28 zu sehen ist.

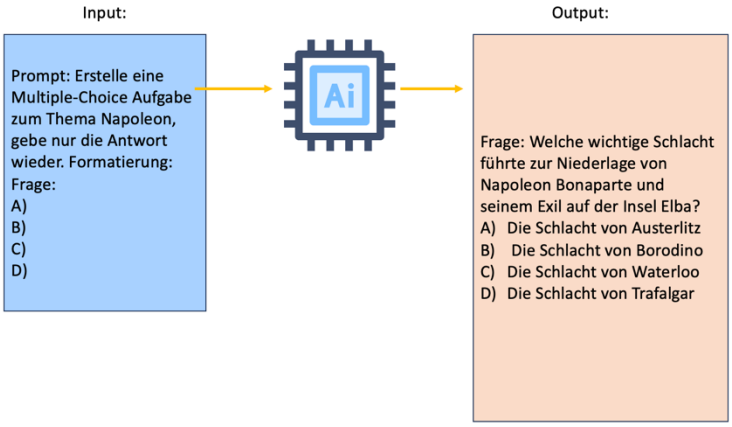


Abbildung 28 (Eigene Grafik)

Allerdings ist der Output für eine Lern-App wie «Studyflow» noch nicht nutzbar, da das Niveau der Multiple-Choice Frage in den meisten Fällen nicht an den User angepasst ist. So kann es vorkommen, dass eine Frage generiert wird, welche für den Nutzer gar nicht beantwortbar ist, oder dass sie zu lange ist. Zudem kann das Backend nicht erkennen, welches die richtige Antwort ist.

Ein Grossteil der Programmierarbeit von «Studyflow» entstand exakt an dieser Stelle: Der Generierung eines vielversprechenden Prompts.
Um gute Ergebnisse zu erzielen, damit die LLM (gpt-3.5-turbo) konsistent dem Niveau angepasste Fragen generiert, welche vom Backend verstanden werden und schliesslich auf dem Frontend entsprechend angezeigt werden können, brauchte es viele Iterationen. Bei der Entwicklung wurden endlose Testabfragen an das LLM gestellt, die wiederum ausgewertet werden mussten.
Man erhält durch diese Tests einen immer besseren, über 300 Zeichen grossen Prompt. Es zeigte sich, dass dieser sogar universell für jede möglichen Eingaben von «Studyflow» genutzt werden konnte und zum wohl zentralsten Teil der App wurde. Der Prompt ist immer

spezifisch auf ein bestimmtes LLM zugeschnitten, daher bleibt der Prompt so lange gültig, wie dieses LLM existiert.

Inkludiert man in diesen Prompt zudem die passenden Lernunterlagen und das Lernziel werden die Ergebnisse nochmals erheblich verbessert. Dazu nutzt «Studyflow» die Sequenzierung: Es werden also nur die Lernunterlagenteile im Prompt integriert, die für die Frage bzw. das Lernziel auch relevant sind, was oben beschrieben wurde.

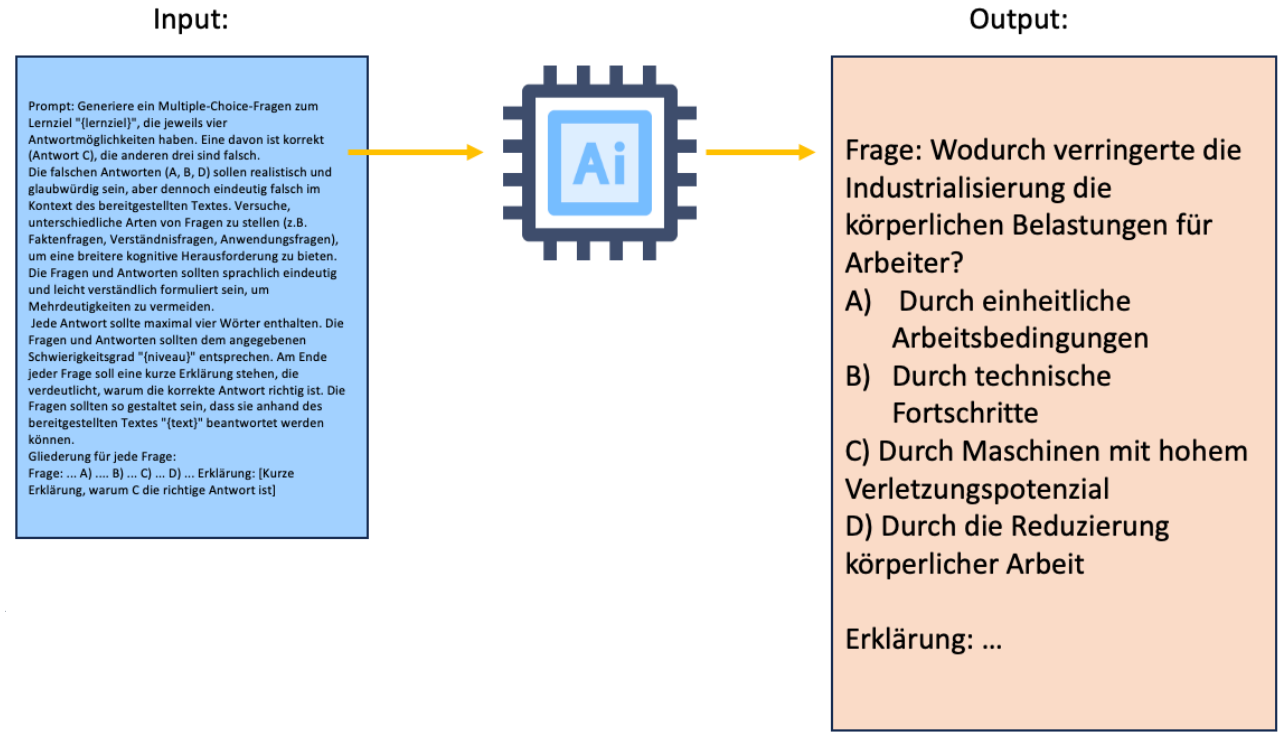


Abbildung 29 (Eigene Grafik)

3.4.5 Auswahl Datenbank

Eine Datenbank ist ein Ablageort für Daten. Dabei werden diese Daten systematisch nach einem Schema abgelegt, damit sie wiedergefunden werden können.

In der heutigen Zeit benötigen die meisten Anwendungen eine Datenbank, um Daten zu speichern und zu verarbeiten. So braucht auch «Studyflow» eine Datenbankanbindung, um überhaupt zu funktionieren. Denn alle hochgeladenen Lernblätter von den Usern müssen irgendwo gespeichert werden, sowie alle generierten Inhalte zur Prüfungsvorbereitung oder auch ganz banal die Login-Daten wie E-Mail-Adressen oder Passwörter.

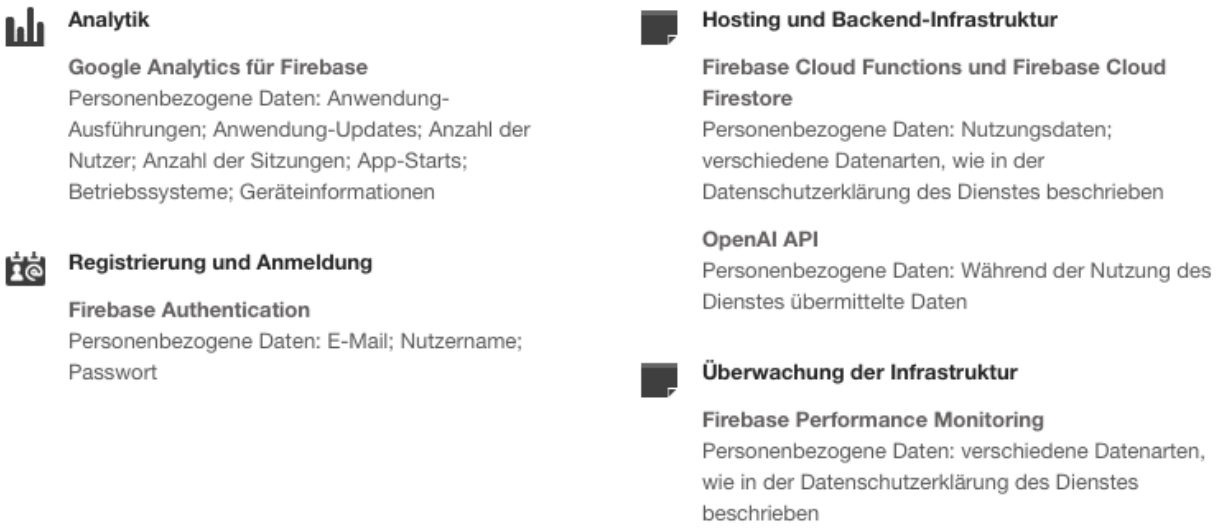
Angesichts der Vielzahl verfügbarer Datenbanken stellt sich für Entwickler oft die Frage: «Wie wählt man die richtige Datenbank für sein System aus?»
Ein zentrales Entscheidungskriterium ist die Wahl zwischen Selbsthosting oder einem gehosteten Service. Das Selbsthosten einer Datenbank bietet erhebliche Vorteile. Dazu gehören eine vollständige Kontrolle über die Daten, klare Preisstrukturen und die Möglichkeit, spezifische Sicherheits- und Leistungsoptimierungen vorzunehmen[14]. Gleichzeitig sind damit Herausforderungen verbunden, wie der erhebliche Zeitaufwand für Verwaltung und Wartung, sowie potenzielle Sicherheitsbedenken. Auch die Skalierung kann komplizierter sein.
Bei gehosteten Diensten profitieren Entwickler von der Expertise des Anbieters in Bezug auf Wartung, Sicherheit und Skalierbarkeit, was oft zu erhöhter Effizienz und geringerem Verwaltungsaufwand führt. Wegen dieser Überlegungen und aus Gründen der Zeitersparnis entschied ich mich, einen gehosteten Dienst zu verwenden. Aber auch hier gibt es eine riesige Auswahl an Anbietern, darunter Amazon DocumentDB[15], Firestore Database[16] und Azure Cosmos DB[17]. Die Wahl fiel auf die Firestore Database von Google, da ich bereits Erfahrung damit hatte und das Angebot ein attraktives kostenloses Kontingent beinhaltete.

[14] https://shorturl.at/npuJK
[15] https://aws.amazon.com/de/documentdb/
[16] https://firebase.google.com/docs/firestore
[17] https://azure.microsoft.com/de-de/products/cosmos-db

3.5 Veröffentlichung

Sobald der Programmierungsprozess abgeschlossen ist, ist der nächste Schritt die Veröffentlichung der App. Bis die App jedoch im Apple Store oder Google Play Store erscheint, müssen viele Dinge getan werden. Unter anderem muss ein App Logo designt werden. Da ich nicht besonders begabt im Designen bin, nutze ich «Midjourney[18]» zur Erstellung von meinem Logo. «Midjourney» ist eine KI, welche auf Basis von Text Bilder generieren kann. Dabei ist auch wieder die Qualität des Prompts wichtig, um gute Ergebnisse zu erreichen. Mein Ergebnis ist nebenan zu sehen.

Zudem muss für die Einreichung auch eine Datenschutzerklärung vorhanden sein. In dieser Datenschutzerklärung muss aufgezeigt werden, wie Daten gesammelt und benutzt werden und ob sie auch an dritte weitergegeben werden. Meine App sammelt personenbezogene Daten, die in der Abbildung 30 alle aufgeführt sind.



Analytik

Google Analytics für Firebase
Personenbezogene Daten: Anwendung-Ausführungen; Anwendung-Updates; Anzahl der Nutzer; Anzahl der Sitzungen; App-Starts; Betriebssysteme; Geräteinformationen

Registrierung und Anmeldung

Firebase Authentication
Personenbezogene Daten: E-Mail; Nutzername; Passwort

Hosting und Backend-Infrastruktur

Firebase Cloud Functions und Firebase Cloud Firestore
Personenbezogene Daten: Nutzungsdaten; verschiedene Datenarten, wie in der Datenschutzerklärung des Dienstes beschrieben

OpenAI API
Personenbezogene Daten: Während der Nutzung des Dienstes übermittelte Daten

Überwachung der Infrastruktur

Firebase Performance Monitoring
Personenbezogene Daten: verschiedene Datenarten, wie in der Datenschutzerklärung des Dienstes beschrieben

Abbildung 30 (Eigenes Bild)

[18] https://www.midjourney.com/

Die ganze Datenschutzerklärung von «Studyflow» kann unter diesem Link eingesehen werden: https://shorturl.at/qILQ4

Um eine App in den Apple App Store hochzuladen, ist eine Lizenz notwendig, welche jährlich 109.- kostet. Google ist etwas günstiger: Um seine App im Google Play Store zu veröffentlichen, benötigt man eine Lizenz, die einmalig nur 25 CHF kostet[19]. Allerdings werden Apps bei Apple nicht nur viel schneller veröffentlicht (12h anstelle von 8d bei Google), sondern auch von Mitarbeitenden von Apple ausgiebig getestet, damit die Qualität im Store hoch bleibt und keine Sicherheitsprobleme mit Apps entstehen[20]. Meine erste Einreichung von «Studyflow» wurde abgelehnt, weil ich versehentlich angab, dass es sich um eine iPad-App handelt. Der Tester merkte, dass das Design nicht iPad-konform war und lehnte mein App ab. Die zweite Einreichung war dann erfolgreich.

[19] https://shorturl.at/fqrF4
[20] https://developer.apple.com/app-store/review/guidelines/

4 Test Schule Neuhausen

Nach der Veröffentlichung hatte ich die Chance, meine App mit einer Sekundarschule zu testen. Es handelte sich um eine 2. Sekundarklasse mit 23 Schülerinnen und Schülern.
Es war mir wichtig, dass die Lernenden keinerlei Bedienungsanleitungen bekamen, da das Ziel war, dass die App intuitiv bedienbar ist (siehe user experience).
Es wurden lediglich Installationsangaben gemacht, die in Abbildung 31 dargestellt sind. Dies war notwendig, da die Installation unter Android etwas komplizierter war, weil die App noch nicht im Google Play Store geprüft und zugelassen war.

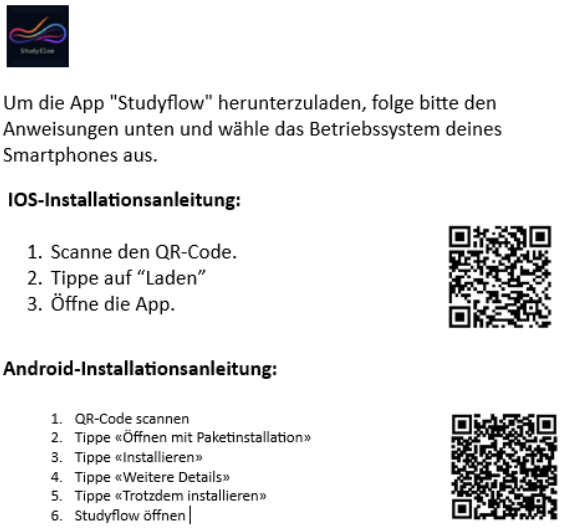


Abbildung 31 Installationsanleitung (Eigenes Bild)

Nachdem die App installiert war, nutzte die Klasse Studyflow, um sich auf eine Geschichtsprüfung über das Thema Napoleon vorzubereiten.
Die Lernenden fotografierten ihre Lernunterlagen mit handschriftlichen Notizen (Input in Abbildung 32 zu sehen).
Nach wenigen Minuten erhielten sie von Studyflow die fertigen Lernformate:
Lernkarten, Zusammenfassung und Testfragen sowie eine Testprüfung (Output).

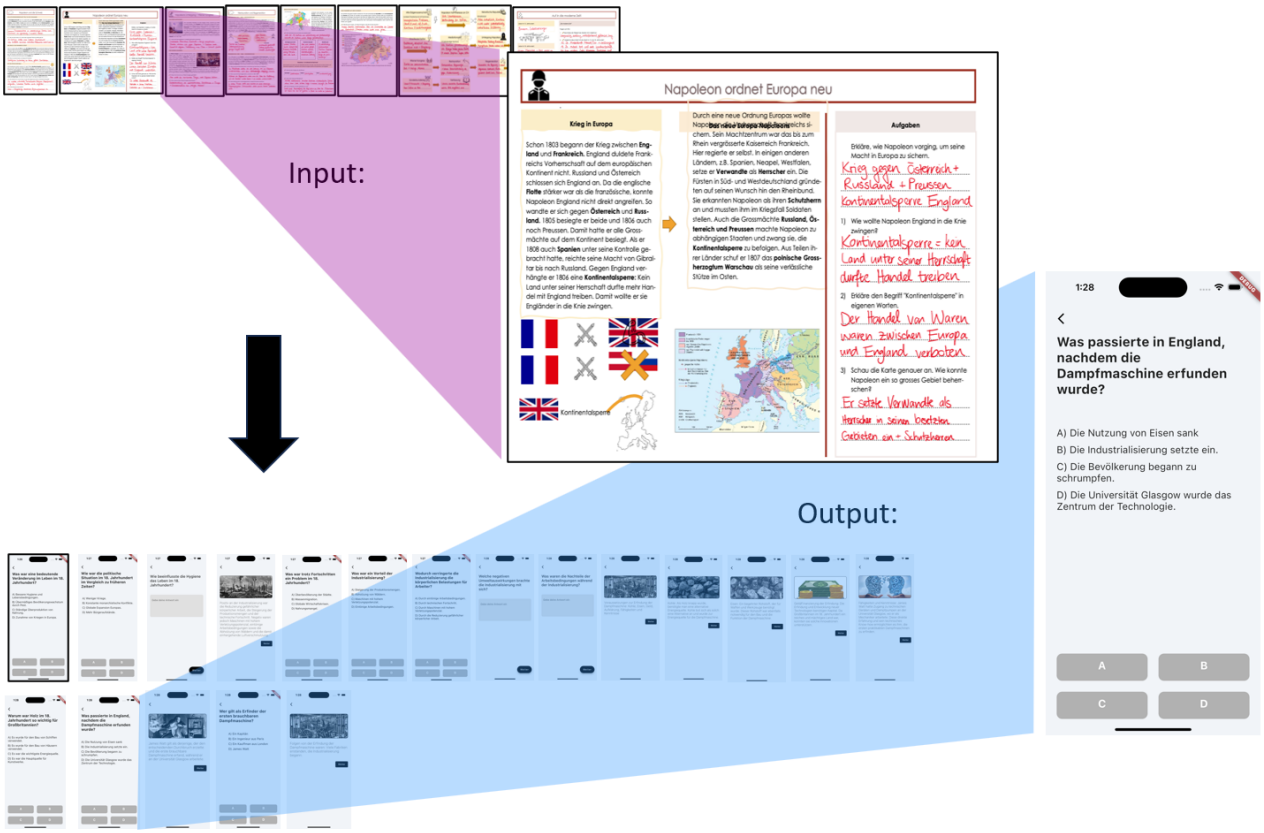


Abbildung 32 (Eigene Grafik)

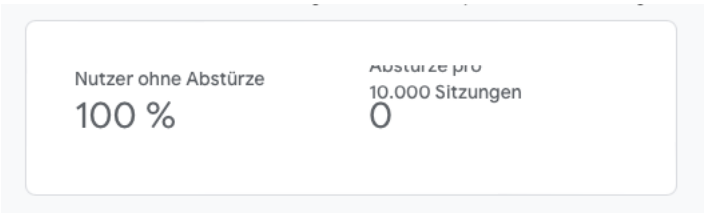


Abbildung 33 (Eigenes Bild)

Diese Lektion war äusserst aufschlussreich für mich, da ich viele Daten sammeln konnte. Sehr erfreulich war, dass die Lernenden keine Hilfestellungen brauchten. Sie konnten die Lernmedien mit «Studyflow» ohne weiteres erstellen.

Zur Softwarestabilität: Wie man in Abbildung 33 sehen kann, hatte ich seit der Veröffentlichung, noch keinen einzigen App Absturz, was für die Qualität der Software spricht. Auch lässt sich nun genau analysieren, wie intuitiv die gesamte App ist, da genau ausgewertet

werden kann, wie die Nutzer durch die App navigiert sind, wie in Abbildung 34 zu sehen ist.

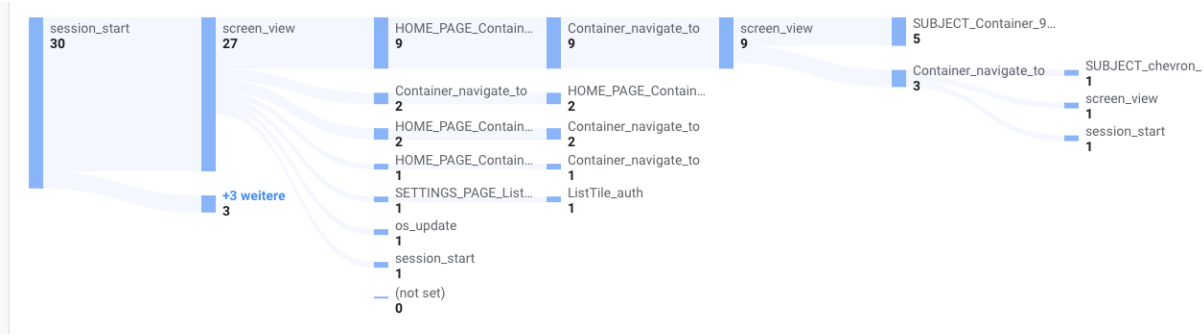


Abbildung 34 User Navigation (Eigene Grafik)

Ebenfalls ein wichtiger Aspekt der Auswertung war, wie der Server, auf dem das Backend läuft, mit 23 gleichzeitigen Usern funktioniert. Zu meinem Erstaunen stellte dies kein Problem dar. In Abbildung 35 sieht man die Ram Auslastung, während die Klasse die App nutze. Auch die Zugriffzeit, welche in Abbildung 36 zu sehen ist, liegt in einem akzeptablen Rahmen.

Das Feedback der Klasse ist mehrheitlich positiv ausgefallen, es wurde darauf hingewiesen, dass eine Übersicht aller Zusammenfassungen hilfreich und eine Fortschrittsanzeige im Prüfungsvorbereitungsmodus sinnvoll wäre. Zudem ist erwähnenswert, dass die Lehrperson von der App begeistert ist und plant, sie zukünftig zur Vorbereitung auf Prüfungen im Unterricht einzusetzen.

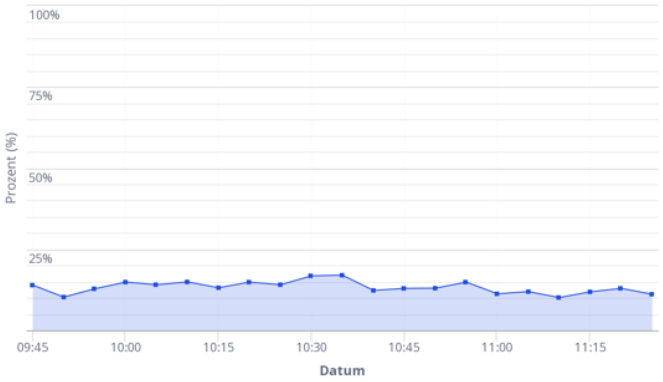


Abbildung 35 Ram Auslastung (Eigene Grafik)

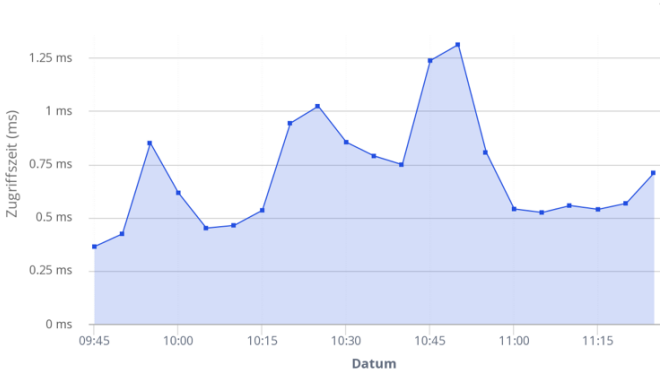


Abbildung 36 Server Zugriffszeit (Eigenes Bild)

5 Fazit

Der grosse Erfolg dieser Arbeit liegt darin, dass es mir gelungen ist, eine App zu entwerfen und zu veröffentlichen. Diese App überführt individuelle Lernmaterialien eigenständig in angepasste Zusammenfassungen und Lernkarten, wobei sie sowohl die Lernstufen als auch die spezifischen Lernziele berücksichtigt. Besonders freut mich, dass die App den Praxistest an der Schule bestanden hat und ich dort viel über die Nutzung und das Navigationsverhalten von Lernenden lernen konnte.

Eine App-Idee in die Realität umzusetzen, braucht seine Zeit. Allein die Entwicklung des Designs dauert schon gut über 20 Stunden. Doch die meiste Zeit nimmt die reine Entwicklung der App in Beschlag; sie dauerte weit mehr als 200 Stunden. Ein wesentlicher Grund für diese Dauer ist der Mangel an fundiertem theoretischem Wissen im Bereich des Prompt Engineerings. Um zu verstehen, wie man effektive Prompts gestaltet, war es notwendig, das Verfahren von der Basis auf durch eigene Versuche und Tests zu erlernen. Ganz anders sieht die Verfügbarkeit von Informationen zum reinen Programmieren aus. Im Internet gibt es zahlreiche Tutorials, Anleitungen und Hilfestellungen zu jedem erdenklichen Problem, das man beim Programmieren haben kann. Ich habe bereits mehrere kleinere App-Projekte umgesetzt, jedoch noch nie ein so grosses wie Studyflow. Deshalb konnte ich sehr viel darüber lernen, wie man eine Software-Anwendung dieser Grössenordnung plant, umsetzt und veröffentlicht.

Ich habe vor, die App auch nach dem Abschluss dieser Arbeit weiterzubetreiben. Die nächsten Schritte sind, weitere Daten über das Nutzungsverhalten zu sammeln, um dann anhand der gewonnenen Erkenntnisse die App weiterzuentwickeln. Doch um an Daten zu gelangen, braucht meine App noch mehr Nutzer. Damit das Akquirieren von Usern am einfachsten gelingt, habe ich vor, Werbung auf TikTok oder Instagram zu schalten, da dort genau meine Zielgruppe anzutreffen ist.

6 Mein Dank

Ich möchte meinen Dank an meinen Betreuer, Herrn Steiger, richten. Seine Unterstützung und Begleitung während des gesamten Prozesses waren unerlässlich und seine hohe Verfügbarkeit für Fragen war eine grosse Hilfe. Speziell geschätzt habe ich die schnellen und präzisen Rückmeldungen. Ein weiterer Dank geht an die zweite Sekundarklasse 2b in Neuhausen und deren Lehrpersonen, die meine App ausführlich getestet haben.