Zürcher Hochschule für Angewandte
Wissenschaften

Projektarbeit

Dialekte raten – Erweiterung einer Web-Applikation

Autoren:
Kai Mannhart
Nathalie Achtnich

Betreuer:
Mark Cieliebak
Manuela Hürlimann

Centre for Artificial Intelligence

5. Januar 2023

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Zürich, 29.12.2022	Kai Mannhart
Schlieren, 29.12.2022	Nathalie Achtnich

Zusammenfassung

Bis heute existieren zwar schon einige Online-Angebote rund um die schweizerdeutsche Sprache, jedoch noch kein Tool, das interessierten Personen – insbesondere Personen mit Muttersprache Schweizerdeutsch – das spielerische Erraten von Schweizer Dialekten ermöglicht. Diese Lücke soll die vorliegende Arbeit schliessen. Unter Verwendung der Audiodaten des STT4SG-350-Korpus wurde eine Web-Applikation in Typescript entwickelt, die eine Erweiterung der bestehenden Schweizer-Dialektsammlung-Applikation darstellt. Die Implementierung auf Basis eines vorhandenen Programmcodes erforderte im Vorfeld die sorgfältige Analyse desselben und nutzte viele der bestehenden Strukturen, zuzüglich des Einbaus der Features, die spezifisch für das vorliegende Projekt sind. Die entwickelte Software erlaubt das Abspielen von schweizerdeutschen Aufnahmen, nimmt einen Rateversuch über die Herkunft entgegen und gibt dem Ratenden anschliessend ein Feedback über die Güte der Vermutung sowie den korrekten Herkunftsort zurück. Dabei geschieht das Raten auf der Postleitzahl-Ebene und kann über eine interaktive Karte oder eine Textfeldeingabe erfolgen. Erste qualitative User-Tests haben ergeben, dass die entwickelte Applikation bei der Zielgruppe auf grosses Interesse stösst. Um die Attraktivität weiter zu steigern, wäre es denkbar, die Applikation in zukünftigen Arbeiten um Features zu ergänzen, die den Wettbewerbscharakter des Ratespiels sowie den Lerneffekt in Bezug auf die Schweizer Dialekte noch mehr fördern.

Abstract

So far, some online tools around the Swiss German language already exist, but still there is no tool that allows interested people – especially people with native language Swiss German – to playfully guess Swiss dialects. The present work aims at filling this gap. Using the audio data of the STT4SG-350 corpus, a Typescript web application was developed, which is an extension of the existing Schweizer Dialektsammlung application. Since the implementation is based on an existing program code, it was required to carefully analyze this code in advance. Many of the existing structures could be re-used while new features specific to the present project were also incorporated. The software at hand allows the playback of Swiss German recordings, then accepts a guess about the origin, and finally returns feedback to the guesser about the goodness of their guess as well as the correct place of origin. Guessing happens on the zip code level and can be done via an interactive map or a text field input. Initial qualitative user tests have shown that the present application is of great interest to the target group. In order to further increase its attractiveness, it is conceivable in future work to add features to the application that would promote the competitive nature of the guessing game as well as the learning effect with regard to Swiss dialects.

Vorwort

Schweizer Fachhochschulen sind geradezu prädestiniert dafür, dass sich dort Studenten aus unterschiedlichen Dialektregionen über den Weg laufen, da es in der Deutschschweiz nur sechs öffentliche Fachhochschulen gibt, jedoch 20 Kantone mit oftmals eigenen Dialektvarianten. So war es auch bei uns: Zürichdeutsch trifft auf Bündnerdeutsch, und Missverständnisse konnten nicht immer vermieden werden. Doch aus der Not galt es, eine Tugend zu machen, und so beschlossen wir, aus unserem „dialektalen Potential“ und dem geteilten Interesse für Sprache im Allgemeinen und Dialekte im Besonderen eine Projektarbeit zum Thema zu produzieren. Diese Arbeit und insbesondere ihr Produkt, die „Dialekte-Raten“-App, richten sich an alle, die unsere Faszination für die Schweizer Dialekte teilen.

Einen besonderen Dank möchten wir unseren Betreuern Prof. Dr. Mark Cieliebak und Manuela Hürlimann aussprechen, die uns in der Ausarbeitung des Themas der vorliegenden Arbeit tatkräftig unterstützt und uns im Verlauf des Semesters mit vielen Tipps und Ratschlägen versorgt haben. Zudem bedanken wir uns herzlich bei Julia Hartmann und Michel Plüss von der FHNW, die uns den Sourcecode der Dialektsammlung-Webapplikation und die SNF-Daten zur Verfügung gestellt und geduldig unsere Fragen dazu beantwortet haben. Ein weiterer Dank geht an unsere Korrekturleserin Mirjam Dietrich sowie an unsere Testuser, die sich inmitten der Weihnachtsfeierlichkeiten für unser Projekt Zeit genommen haben.

Triggerwarnung: In der gesamten vorliegenden Arbeit verwenden wir wann immer möglich das generische Maskulinum, das uns als leseflussfreundlichste Personenbezeichnung bei gleichzeitiger Einschliessung aller Menschen erscheint. Wir sind uns allerdings bewusst, dass die Meinungen hierzu in der aktuellen politischen (und anderswo geführten) Debatte weit auseinandergehen.

Inhaltsverzeichnis

1 Einleitung

Schweizerdeutsch ist die Landessprache der Schweiz mit den meisten Sprechern – und den meisten Dialekten. Etwas mehr als 60 Prozent der gut acht Millionen Einwohner geben Deutsch/Schweizerdeutsch als ihre Hauptsprache an [1]. Sie verteilen sich auf 20 der 26 Kantone, in denen Deutsch (u.a.) als Amtssprache festgelegt ist. Ein Grossteil dieser Kantone beansprucht einen charakteristischen Dialekt für sich, so zum Beispiel Zürichdeutsch, Berndeutsch, Bündnerdeutsch und weitere kantonale Dialekte, die von schweizerdeutschen Sprechern oft mit Stolz gesprochen (und voneinander abgegrenzt) werden.

In keinem anderen deutschsprachigen Land besitzen Dialekte ein solch hohes Prestige wie in der Schweiz [2]. Zwar wird in öffentlichen und schriftlichen Kontexten, wie z.B. in Nachrichten des Schweizerischen Rundfunks, Parlamentssitzungen des Nationalrates, Zeitungen und Gesetzestexten oder auch in der Schule, überwiegend Hochdeutsch eingesetzt. Jedoch wird in sämtlichen informellen Settings, einschliesslich Firmenmeetings, Bewerbungsgesprächen und vermehrt auch in der schriftlichen Kommunikation via E-Mail oder Messenger-Dienste auf dem Smartphone – insbesondere unter der jungen Generation –, zumeist Mundart verwendet, vorausgesetzt, sie ist für alle Beteiligten verständlich. Mit seinen Freunden ohne spezifischen Anlass Standarddeutsch zu sprechen, käme kaum einem Schweizer in den Sinn, auch dann nicht, wenn sie aus einem anderen Kanton stammen und die Verständigung aufgrund dialektaler Eigenheiten nicht völlig reibungslos klappt. Allerdings kann es beim Aufeinandertreffen von Sprechern aus verschiedenen Kantonen durchaus zu Sticheleien kommen, wenn einer der Beteiligten allzu dialektspezifische Charakteristika in seiner Sprechweise offenbart. Viele Dialekte haben neben ihren Eigenheiten auch ihren Ruf, der die Sprecher durch die ganze Schweiz verfolgen kann. Umfragen fördern regelmässig die schönsten und die unsympathischsten Dialekte der Schweizer Sprachlandschaft zutage, die allerdings gemäss einer Studie offenbar weniger mit den linguistischen Besonderheiten der einzelnen Dialekte zu tun haben, sondern mehr mit den Vorurteilen gegenüber den Kantonen und deren Bevölkerung [3].

Aufgrund des starken (auch emotionalen) Bezugs zu ihren Dialekten erstaunt es nicht, dass viele Schweizer den Dialekt ihres Deutschschweizer Gegenübers oft problemlos einem Kanton oder einer Region zuordnen können. Manche Schweizer behaupten sogar, gewisse Sprecher bis auf ihre Heimatsstadt oder ihr Heimattal lokalisieren zu können. Inwieweit dies der Fall ist, konnte bisher jedoch noch nicht in grösserem Umfang erprobt werden, da keine leicht zugänglichen Aufnahmen von schweizerdeutschen Sprechern samt Informationen zu ihrer Herkunft öffentlich verfügbar waren.

Hier soll die vorliegende Arbeit Abhilfe schaffen. Seit 2022 liegen neue Audiodaten vor, die erstmals systematisch in der ganzen Deutschschweiz aufgenommen wurden und auch über Angaben zum Wohnort des Sprechers verfügen. Die Idee besteht nun darin, diese Aufnahmen für eine Applikation zu verwenden, die dem User einen spielerischen Zugang zum Erra-

ten von Dialekten ermöglicht. Dies soll auf der Basis einer bestehenden Dialektsammlungs-Applikation geschehen, um gleichzeitig im Idealfall deren Bekanntheit zu steigern und die Gewinnung weiterer Daten anzustossen.

1.1 Ausgangslage

1.1.1 Die Schweizer Dialektsammlung

Die Schweizer Dialektsammlung [4] ist ein Forschungsprojekt der Swiss Association for Natural Language Processing (SwissNLP) mit dem Ziel, ein grosses Korpus an Schweizer Sprachdaten (mind. 2000 Stunden aus allen Schweizer Dialekten) anzulegen, mit dem in Zukunft ein auf Machine Learning basierendes Speech-to-Text-System trainiert werden kann. Bisher existieren für Schweizerdeutsch keine überzeugenden Systeme dieser Art, was vor allem mit der kargen Datenlage zu Schweizerdeutsch zusammenhängt: ML-Systeme sind zumeist auf grosse Datenmengen angewiesen, um einen nützlichen Grad an Präzision bei der Verschriftlichung (oder Übersetzung) von gesprochener Sprache zu erreichen. Die Hoffnung der Schweizer Dialektsammlung ist, genügend Daten zusammenzutragen, um die bisherige Leistung von ML-Systemen mit Schweizerdeutsch deutlich steigern zu können.

Die Sammlung geschah bis im Jahr 2021 über freiwillige Teilnehmer, die hochdeutsche Sätze in ihrem jeweiligen Dialekt nachsprachen, aufnahmen und hochluden. Weitere Freiwillige bewerteten die Aufnahmen danach, ob sie dem hochdeutschen Text entsprachen und tatsächlich Schweizerdeutsch waren, womit die Qualität der Aufnahmen sichergestellt werden konnte. Die Dialektsammlung wurde von Schweizer Prominenten und Schweizer Medien unterstützt und beworben; ausserdem lief bis im Sommer 2021 ein Wettbewerb inklusive Preisvergabe, um möglichst viele Menschen zum Mitmachen zu bewegen. In der ersten Phase konnten so 200 Audiostunden zusammengetragen werden.

Im Jahr 2022 konnte ausserdem eine von der Fachhochschule Nordwestschweiz (FHNW) in Zusammenarbeit mit der Schweizer Dialektsammlung durchgeführte Datensammlung von Schweizer Dialektaufnahmen abgeschlossen werden [5]. Die Datensammlung geschah systematisch in der ganzen Schweiz, indem von jeder Dialektregion dreissig Sprecher aus verschiedenen Regionen ausgewählt wurden, die insgesamt über 200'000 schweizerdeutsche Sätze aufnahmen. Diese Sammlung ist in ihrem Ausmass, in ihrer Systematik und sorgfältigen Auslese der Sprecher sowie der Erhebung der Herkunft auf Ortschaftsebene in der Schweiz bisher einzigartig und bietet nicht nur neue Möglichkeiten in Bezug auf ML-Systeme, sondern auch in Bezug auf das Testen und Erweitern von Schweizer Wissen über die landeigenen Dialekte.

1.1.2 Konkurrenzanalyse

Obwohl die Schweizer Dialektsammlung inzwischen über all die oben erwähnten Daten verfügt, wurde bisher noch kein Versuch unternommen, diese für eine Art Ratespiel über die

Schweizer Dialekte einzusetzen. Auch sonst existiert bis heute noch kein (online verfügbares) Instrument für ein spielerisches Raten von Schweizer Dialekten. Zwar haben schon verschiedene Zeitungen [6][7] sogenannte „Dialekt-Checks“ publiziert, bei denen Interessierte jeweils ausgewählte Dialekte anhand einzelner Ausdrücke oder Sätze zu klassifizieren versuchen konnten. Diese beschränken sich aber oft auf Spezialbegriffe eines Dialekts und/oder sind nur schriftlich vorhanden – die Klassifizierung geschieht also nicht aufgrund phonetischer Merkmale des Dialekts.

Des Weiteren gibt es verschiedene Web-Dienste, die den umgekehrten Service anbieten, indem sie die Herkunft des Nutzers anhand seines Dialekts zu ermitteln versuchen. Dazu gehört das „Chochichästli-Orakel“ [8], das anhand der Aussprache von zehn charakteristischen Wörtern die Herkunft des Nutzers errät. Dabei sind die möglichen Aussprachen der Wörter schriftlich festgehalten und werden vom Nutzer aus einer Dropdown-Liste ausgewählt. Fürs Smartphone gibt es die „Dialäkt-Äpp“ [9], die den User ebenfalls seine Aussprache spezifischer Wörter aus einer Liste auswählen lässt und ihn anschliessend anhand dieser klassifiziert. Der User kann sich ausserdem andere Dialekte aus verschiedenen Zeitepochen anhören – die Daten hierzu stammen aus dem Phonogrammarchiv der Universität Zürich [10], das die Aufgabe hat, alle Schweizer Dialekte (inklusive der anderen Landessprachen der Schweiz) zu dokumentieren und archivieren. Gleichzeitig sammelt die „Dialäkt-Äpp“ neue Audiodaten für die Universität Zürich und trägt damit selbst zur Archivierung der Schweizer(deutschen) Dialekte bei. Weder die Audiodaten des Phonogrammarchivs noch jene, die die „Dialäkt-Äpp“ gesammelt hat, wurden bisher für die Konzeption einer „Dialekte-Raten“-Applikation verwendet, und es bleibt auch ungeklärt, inwieweit sich diese Daten dafür eignen würden, insbesondere die historischen Daten des Phonogrammarchivs.

Zu guter Letzt seien hier noch die Dialektkarten [11] erwähnt, die eine Digitalisierung einer Auswahl von Arbeitskarten des *Sprachatlas der deutschen Schweiz* darstellen. Der Sprachatlas dokumentiert die Aussprache einzelner Wörter oder Laute in sämtlichen deutschsprachigen Regionen der Schweiz. Dank den Dialektkarten ist es möglich, diese Informationen kompakt auf einer Webseite abzurufen und weitere Auswertungen zu konsultieren. Jedoch bieten auch die Dialektkarten kein Audiomaterial und sind nicht dazu gedacht, Dialekte zu erraten.

Zusammengefasst existieren rund ums Thema Schweizer Dialekte zwar schon einige wenige Online-Angebote, aber bislang bleiben diese Nischenprodukte. Eine Applikation, mit der sich Dialekte raten lassen, gibt es bisher noch nicht, doch die dafür nötigen Daten sind inzwischen vorhanden. Aufgrund des engen Bezugs der Schweizer zu ihren Dialekten bietet sich der Bau einer solchen Applikation an, die es ihnen ermöglicht, ihr Wissen über die Dialekte spielerisch auf den Prüfstand zu stellen.

1.2 Zielsetzung / Aufgabenstellung

Das Ziel der vorliegenden Arbeit ist es, mittels einer Applikation Personen mit Schweizerdeutsch als Muttersprache (und allen weiteren Interessierten) eine Möglichkeit zur Verfügung zu stellen, ihr Wissen über die Deutschschweizer Dialekte auf eine spielerische Art zu erproben und zu erweitern. Die Aufgabe lautet hierbei, auf Basis der bestehenden Dialektsammlung-Webapplikation eine Webseite zu gestalten, auf der die Herkunft von schweizerdeutschen Sprechern erraten und ein Feedback zur Treffsicherheit eingeholt werden kann. Die Ankopplung an die bestehende Dialektsammlung geschieht auch aus der Hoffnung heraus, dass das neue Feature zur weiteren Bekanntheit der Dialektsammlung beitragen wird und dadurch neue Sprecher gewonnen und weitere Audiodaten zur schweizerdeutschen Sprache gesammelt werden können. Deshalb wird die Zielgruppe vordergründig auf Personen mit Schweizerdeutsch als Muttersprache eingegrenzt und die Applikation, wie im nächsten Kapitel beschrieben, bewusst so konzipiert, dass sie für Muttersprachler einen besonderen Reiz bietet.

1.3 Outline

Die vorliegende Arbeit beginnt in Kapitel 2 mit einem Überblick über die bestehende Dialektsammlung-Webapplikation und beschreibt anschliessend die Konzeption der vorliegenden Arbeit und deren Integration in die bestehende Webseite, wobei technische Gesichtspunkte miteinbezogen werden. Auch die Vorgehensweise und Arbeitsprozesse in der Entwicklung der Webapplikation werden erläutert. In Kapitel 4 werden die Resultate der realisierten Lösung anhand Usertests vorgestellt und in Kapitel 5 diskutiert, inwiefern das Ziel der Arbeit erreicht wurde. Ebenfalls erfolgt ein Ausblick auf mögliche Erweiterungen des neuen Features.

Bei der nachfolgenden Beschreibung und Diskussion des Projekts wird ein grundlegendes Verständnis für Software-Entwicklung und Webapplikationen vorausgesetzt.

2 Analyse der bestehenden Applikation

2.1 Überblick

Die Schweizer Dialektsammlung ist eine Webapplikation, die auf dem Mozilla-Common-Voice-Projekt basiert. Das Front-End der Applikation wurde mittels React/Typscript realisiert und läuft auf einem Express-Server. Im Backend ist eine Mysql-Datenbank initialisiert, in der die Userdaten und -einstellungen sowie die restlichen Daten abgelegt werden. Die Audiodaten sind in einem Amazon-S3-Bucket als Key/Value-Pair gespeichert. Währenddessen sind die dazugehörigen Metadaten sowie der Key ebenfalls in der Datenbank festgehalten.

Der Projektcode wird auf GitHub gehostet und verwendet yarn [12] als Paket-Verwaltung. Eine Übersicht über die wichtigsten yarn-Befehle zur Verwaltung der Module findet sich im Anhang unter 7.2.2.

Die Struktur des Projekts sieht folgendermassen aus:

- `common`: Ordner für Features, die sowohl vom Front-End als auch vom Back-End benutzt werden, z.B. Settings-Interface.
- `server`: Ordner für die Server-seitige Applogik.
- `web`: Ordner für die Webseiten-Files.

Daneben ist ein `docker`-Ordner für die Dockerfiles vorhanden und ein `docs`-Ordner mit diversen Dokumentationen zum Builden des Projekts und über das Common-Voide-Projekt.

Im Folgenden wird auf das Front-End, die Schnittstelle zwischen Web und Server sowie auf das Back-End ausführlicher eingegangen.

2.2 Front-End

Das Herzstück der bestehenden Dialektsammlung-Applikation ist die Seite, wo die Nutzer zur Sammlung beitragen können: Entweder nehmen sie selbst Sätze auf (siehe Abbildung 1), oder sie prüfen die Sätze, die andere Nutzer aufgenommen haben (siehe Abbildung 2). Auch das vorliegende Projekt sieht die Interaktion des Users mit der Applikation vor, sodass entschieden wurde, den neuen Teil an dieser Stelle anzugliedern und codemässig stark darauf aufzubauen. Deshalb erfolgt nun eine grobe Beschreibung des bestehenden User Interfaces sowie des dahinter liegenden Codes. In Kapitel 3.2.1 wird dann bei der Beschreibung der Integration der „Dialekte-Raten“-Seite in das bestehende Projekt der Bezug zum in diesem Kapitel beschriebenen Code-und UI-Design hergestellt.

2.2.1 User Interface

Die Seite besteht aus zwei Tabs (einen fürs Aufnehmen, einen fürs Prüfen), zwischen denen hin- und hergewechselt werden kann. Bei beiden Use Cases arbeitet sich der User jeweils

Abbildung 1: UI für das Aufnehmen von Sätzen.

durch fünf Sätze, bevor er seine Daten abschickt und damit an den Server liefert. Danach entscheidet er sich, ob er weiteraufnehmen/-prüfen (und damit die nächsten fünf Sätze angezeigt bekommen) oder sich auf der Webseite registrieren will. Das Aufnehmen/Prüfen lässt sich jederzeit mit dem „Zurück"-Button abbrechen. Für schnellere Abläufe gibt es Tastenkürzel, die es erlauben, per Tastendruck eine Aufnahme zu starten oder zu stoppen, eine Aufnahme als korrekt oder falsch zu klassifizieren und weitere Funktionen. Auch können via Melde-Button Probleme mit einem Satz bekanntgegeben werden, so zum Beispiel, wenn der Satz einen Rechtschreibe-Fehler oder unbekannte Wörter enthält.

2.2.2 Code

Der Hauptcode für das User Interface der Aufnehmen- und Prüfen-Seite befindet sich in der Projektstruktur unter `web/src/components/pages/contribution`, aufgegliedert in die Unterordner `speak` fürs Aufnehmen und `listen` fürs Prüfen. Gemäss dem verwendeten Tooling gibt es für jede Seite ein *tsx*-File für die React-Komponenten und ein *css*-Stylesheet (`speak.tsx`/`speak.css` resp. `listen.tsx`/`listen.css`). Der Sprechteil benötigt noch weitere Files für das Handling der Aufnahmen, die hier nicht berücksichtigt werden. Die *tsx*-Files beinhalten die Klassendefinitionen der `SpeakPage` und `ListenPage`, die Erweiterungen von React-Components darstellen und über Props und einen State verfügen, deren Interfaces im Vorfeld definiert werden. Für das Laden des State von den Props steht die Funktion `getDerivedStateFromProps` zur Verfügung. Um die Props zu laden, wird das Framework React Redux verwendet, das Daten aus einem internen Store einliest, was in der Funktion `mapStateToProps` passiert. Des Weiteren sind auch die Funktionen, die den Abschluss der drei Lebensphasen der React-Component markieren, vorhanden, wobei `componentDidMount` und `componentDidUpdate` vor allem für die Steuerung des Einblendens der jeweiligen Kurzanleitung genutzt werden, die dem Nutzer bei erstmaliger Ansicht

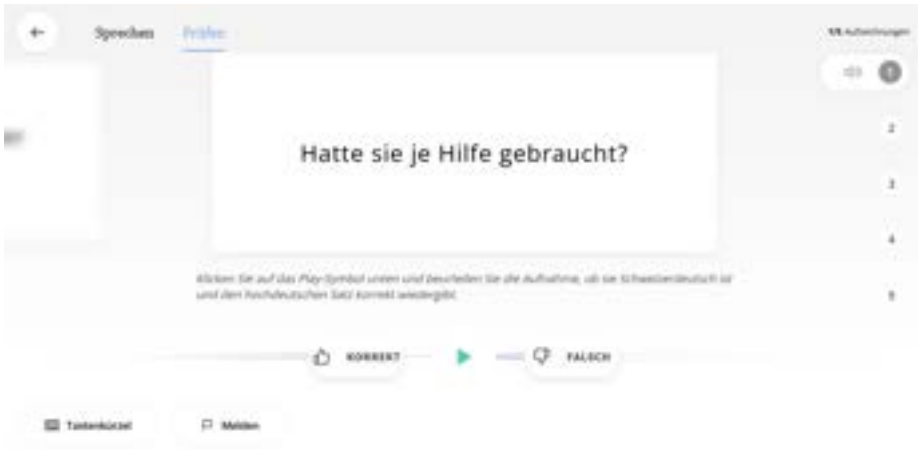


Abbildung 2: UI für das Prüfen von aufgenommenen Sätzen.

sowohl des Aufnehmen- als auch des Prüfen-Tabs angezeigt wird.

In der `render`-Methode wird dann das eigentliche UI aufgebaut. In beiden Pages ist hier die sogenannte `ContributionPage` eingebettet, die im File `contribution/contribution.tsx` definiert ist. Deren Props werden in `speak.tsx` resp. `listen.tsx` abgefüllt. Dort wird u.a. definiert, welche Buttons eingeblendet werden, welche Tastenkürzel benutzt werden können und welche Instruktion anzuzeigen ist.

Die restlichen Methoden dienen überwiegend dazu, die verschiedenen User-Aktionen zu verarbeiten, und tragen ihre Funktion meist schon gut verständlich im Namen (z.B. `play`, `vote`, `startRecording`, `handleSkip`). Ein Überblick über die Methoden der `ListenPage` (ausgenommen die React-Lifecycle-Methoden) sowie ausgewählte Abhängigkeiten findet sich exemplarisch in Abbildung 3.

Die Texte, die für die Instruktion, Buttons und weitere sprachliche Elemente verwendet werden, befinden sich in einem separaten Dokument `messages.ftl` unter `web/locales/de/`. Auf diese Texte wird über ihr Label in sogenannten `Localized`-Components zugegriffen, indem das Label dem `id`-Parameter übergeben wird. Diese `Localized`-Components stammen noch aus dem Common-Voice-Projekt und sind eigentlich dazu gedacht, Texte dynamisch in verschiedenen Sprachen anzeigen zu können.

2.3 Schnittstelle zwischen Front- und Back-End

Um die Clips aus dem Bucket zu laden, Userdaten in der Datenbank zu speichern oder sonstige Daten zu handlen, muss das Front-End auf Funktionen auf der Server-Seite zugreifen. Obwohl Web- und Serverseite beide in Typescript geschrieben sind und auch sonst codetechnisch kompatibel wären, verwendet die Web-Seite Requests, um auf den Server

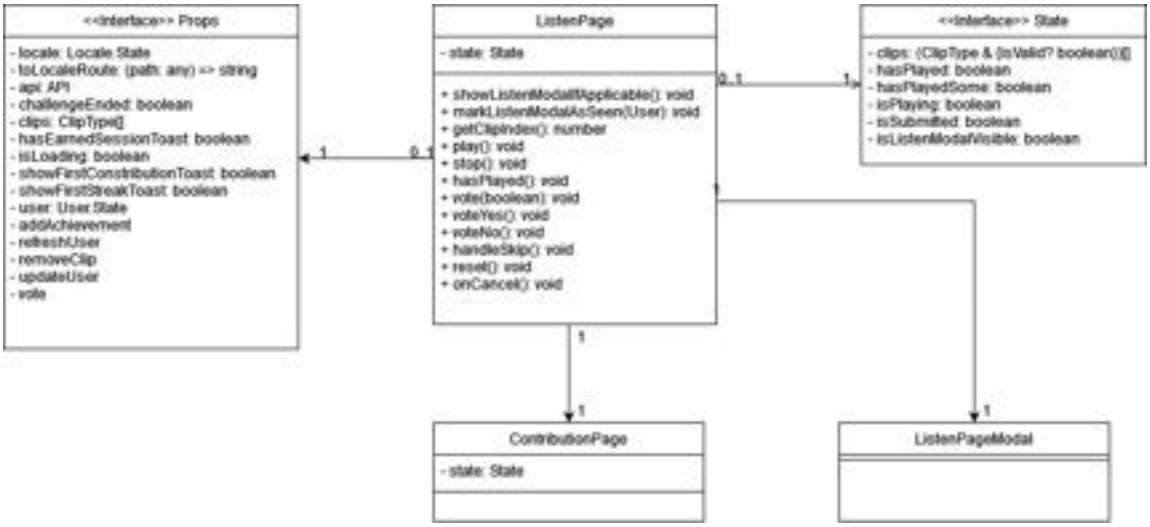


Abbildung 3: Klassendiagramm mit ausgewählten Feldern, Methoden und Abhängigkeiten für ListenPage.

zuzugreifen, damit die Kopplung zwischen Front- und Back-End möglichst gering bleibt. Die grobe Abhängigkeit der involvierten Komponenten ist in Abbildung 4 dargestellt.

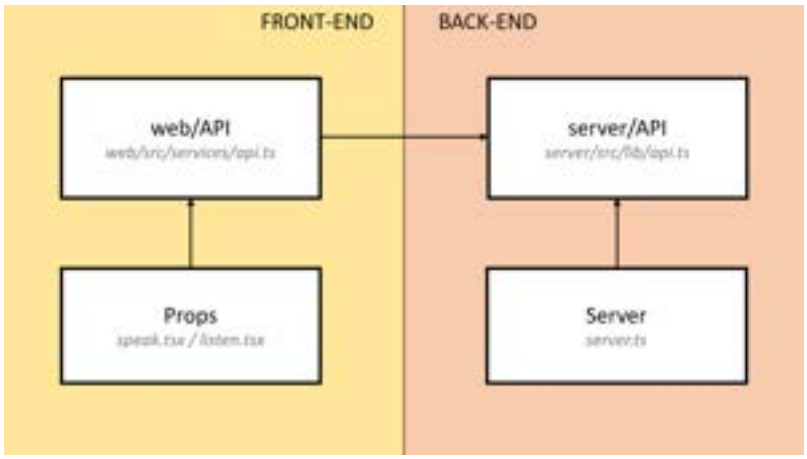


Abbildung 4: Kommunikation zwischen Front- und Back-End.

Wie in der Abbildung 4 ersichtlich und auch schon in Abbildung 3 angedeutet, kann über die Props von der Front-End-Page aus auf die Web-API zugegriffen werden. Die Klasse `API` auf der Web-Seite (in der Abbildung 4 links oben) befindet sich im File `api.ts` unter `web/src/services/` und hält die Funktionen, in die die Requests eingebettet sind.

Sie werden mittels einer hauseigenen `fetch`-Methode als GET- oder POST-Requests an eine als String übergebene Adresse geschickt. Diese Adresse leitet den Request an die `API`-Klasse auf der Server-Seite weiter (in der Abbildung 4 rechts oben), die im File `server/src/lib/api.ts` definiert ist. Über einen express-Router ist dort festgelegt, welche Pfade zu welcher Methode führen. Ein Matching sieht dann folgendermassen aus:

```
// web/src/services/api.ts

const API_PATH = location.origin + '/api/v1'

fetchRequestedLanguages(): Promise<string[]> {
    return this.fetch('${API_PATH}/requested_languages')
}
```

.....

```
// server/src/lib/api.ts: Route

router.get('/requested_languages', this.getRequestedLanguages)
```

```
// server/src/lib/api.ts: Methode

getRequestedLanguages = async (request: Request, response:
   Response) => {
    response.json(await this.model.db.getRequestedLanguages())
}
```

Damit auf der Server-Seite die richtige Funktion aufgerufen wird, ist es auch von Relevanz, an welcher Position die Route im Router definiert wird. Der Router sucht für jeden eintreffenden Request von der Web-Seite seine Routen von oben nach unten durch und schickt sie an den ersten Pfad, bei dem das Matching klappt. Pfade mit einem Asterisk fangen alle Requests auf, die bis dorthin gelangen, und stehen deshalb jeweils zuunterst in der Routenliste.

```
// server/src/lib/api.ts

getRouter(): Router {
    const router = PromiseRouter();
    ...
    router.use('*', (request: Request, response: Response) =>
        {
         response.sendStatus(404);
    });

    return router;
}
```

Nicht alle API-Requests vom Front-End werden über die `API`-Klasse des Servers vermittelt: Manche Requests werden mittels eines unterschiedlichen Pfads (`/clips` als Zusatz) an die Klasse `Clip` unter `server/src/lib/` weitergeleitet. Dies betrifft alle Requests, die auf irgendeine Weise mit Audioclips zu tun haben. Exemplarisch wird in Abbildung 5 ein Interaktionsdiagramm gezeigt, das den Weg eines Prüfresultats – also wenn ein Nutzer bestimmt hat, ob er eine Aufnahme für gültig hält oder nicht – von der ListenPage im Front-End bis zur Datenbank im Back-End abbildet. Dieser Weg führt im Server über die `Clip`-Klasse als Weitervermittler statt über die `API`-Klasse.[1]

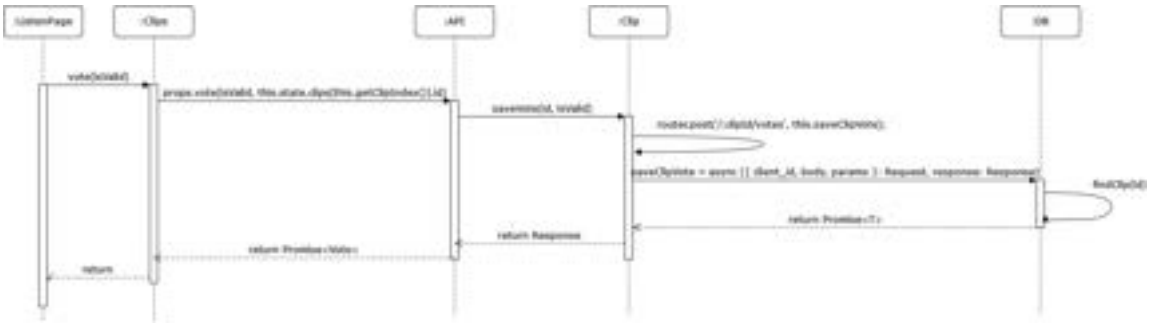


Abbildung 5: Interaktionsdiagramm Front-End – Back-End.

Man beachte, dass das Resultat der Datenbank-Query als Promise bis zur `Clips`-Klasse auf der Web-Seite geliefert wird und dann nur indirekt über ein `dispatch` und die Props zur `ListenPage` gelangt, sodass diese es nutzen kann.

[1] Genau genommen gelangen auch diese Requests zuerst an die `API`-Server-Klasse und werden von dieser dann mittels `getRouter`-Aufruf der `Clip`-Klasse an die entsprechende Route weitergeleitet, die in der `Clip`-Klasse festgelegt ist. Dies wurde der besseren Darstellbarkeit halber in der Abbildung 5 weggelassen.

Auch die Audioclips selbst werden über die Props in die `ListenPage` geladen. In der `getDerivedStateFromProps`-Methode findet die Übergabe der Clips von den Props an den State statt. Zuvor werden die Props in der `mapStateToProps`-Funktion vom internen Store, der in der Klasse `Clips` angelegt ist, geladen. Der API-Aufruf, der die Audioclips aus der Datenbank zieht, erfolgt durch die ebenfalls in der `Clips`-Klasse definierten Aktion `refillCache`.

2.4 Back-End

2.4.1 S3-Bucket

Zur lokalen Entwicklung wird ein S3-Proxy eingesetzt, der in einem Docker-Container läuft. Das Image dafür stammt von Andrew Gaul und wird von ihm unterhalten. Die Konfiguration des Image im Projekt ist relativ klein gehalten, als einziges wird die Autorisation ausgeschaltet, sodass für die lokale Entwicklung keine Credentials erstellt und angegeben werden müssen. Die Konfiguration hierfür befindet sich im File `/docker/docker-compose.yaml`.

Der S3-Bucket dient dazu, die tatsächlichen Audiodaten abzuspeichern, die dann für den Betrieb der Applikation über die Metadaten in der Datenbank (siehe nächstes Kapitel 2.4.2) geladen werden. Die Audiodaten selbst werden nicht mit Docker mitgeliefert, sondern müssen separat in den Bucket importiert werden.

Für die scharfgestellte Webseite steht für die Audiodaten statt des lokalen Buckets ein AWS-S3-Bucket zur Verfügung.

2.4.2 Mysql-Datenbank

Für die lokale Entwicklung wird auch für die Datenbank eine dockerisierte MySQL-Instanz verwendet, die auf dem offiziellen MySQL-Image auf Docker-Hub basiert. Die Datenbank wird beim Start des Projekts mit `yarn` (siehe Kapitel 7.2.2 im Anhang) automatisch über die Migrationsfiles unter `server/js/lib/model/db/migrations` populiert.

Die Datenbank umfasst ingesamt 36 Tabellen. Für das vorliegende Projekt besonders relevant ist hierbei die `clips`-Tabelle. Sie speichert die Metadaten der über die Schweizer Dialektsammlung gesammelten Audioaufnahmen samt deren Path, mit welchem die effektiven Voice Samples aus dem S3-Bucket geholt werden können. Diese werden von der oben beschriebenen `ListenPage` benötigt, die ja die Qualität dieser Aufnahmen verifiziert. In der Abbildung 6 ist die Struktur der `clips`-Tabelle inklusive Attribute dargestellt. In der Tabelle befinden sich Angaben zur Clip-Identifikation, der Speicherort, die verschriftliche hochdeutsche Übersetzung des aufgenommenen Satzes und Angaben dazu, ob das Voice Sample bereits geprüft wurde und wenn ja, ob es als gültig oder ungültig befunden wurde. Wenn man herausfinden möchte, welcher Dialekt der Aufnahme zuzuordnen ist, muss man dies über die ID des Users (`client_id`), der den Satz aufgenommen hat, und einen Join

mit der Tabelle `user_clients` tun, worin die Postleitzahl des Sprechers abgelegt ist.

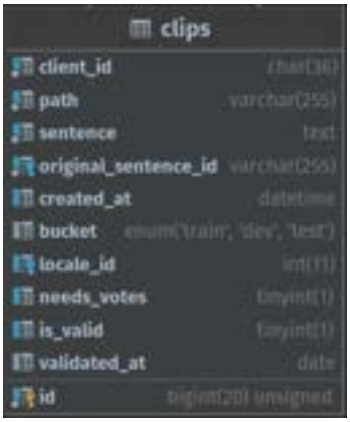


Abbildung 6: Struktur der Tabelle `clips`.

3 Erweiterung der Applikation

In diesem Kapitel geht es zuerst darum, wie die Projektidee zur Erweiterung der Dialektsammlung-Applikation um eine Sektion „Dialekte raten“ im Detail aussah, bevor in einem weiteren Abschnitt die konkrete Umsetzung beschrieben wird. In den letzten zwei Unterkapiteln dieses Teils werden die verwendeten Audiodaten sowie der Arbeitsprozess thematisiert.

3.1 Konzeption

3.1.1 Projektidee

Wie in Kapitel 1.1 erläutert, besteht unter den Schweizern eine starke Verbundenheit mit ihrer Muttersprache und dem jeweiligen Dialekt, jedoch bisher keine Möglichkeit, diese Dialekte in grösserem Stil auf auditive und spielerische Weise besser klassifizieren zu lernen. Die vorliegende Arbeit zielt deshalb darauf ab, die Daten aus der SNF-Sammlung, die in Kapitel 3.3 näher beschrieben werden, für ein Dialekt-Rate-Spiel einzusetzen. Die User sollen sich einzelne Audioclips mit Aufnahmen von Schweizer Sprechern anhören können und anschliessend eine Vermutung abgeben, woher der Sprecher ihrer Meinung nach stammt. Um den Schwierigkeitsgrad zu erhöhen und das Spiel für Schweizer Muttersprachler etwas spannender zu gestalten, wird dabei nicht auf Kantonsebene, sondern auf Ortsebene (Postleitzahl) geraten. Danach erhalten sie ein Feedback, wie nahe ihr Rateversuch der Realität kam, d.h. wie weit der vermutete Herkunftsort vom tatsächlichen entfernt liegt. Das Spiel sollte dabei für den User ansprechend und intuitiv gestaltet sein, sodass das Raten der Dialekte Spass macht. Diese Projektidee ist als User Story inkl. UI-Skizze in der grauen Box auf der nächsten Seite zusammengefasst.

Damit ein Dialekt-Rate-Spiel einem echten Spiel gleichkommt, wäre es nötig, weitere Features einzubauen, bspw. eine Auswertung über alle bisherigen Rateversuche oder eine Rangliste, auf der man sich mit anderen Teilnehmern vergleichen kann. Aufgrund der limitierten Zeit fokussiert das vorliegende Projekt auf das Kernstück, das Raten der Dialekte selbst. Mögliche weitere Features werden in Kapitel 5 thematisiert und im Anhang befinden sich einige bereits ausgearbeitete User Storys dazu (siehe Kapitel 7.4).

3.1.2 Überlegungen zum User Interface

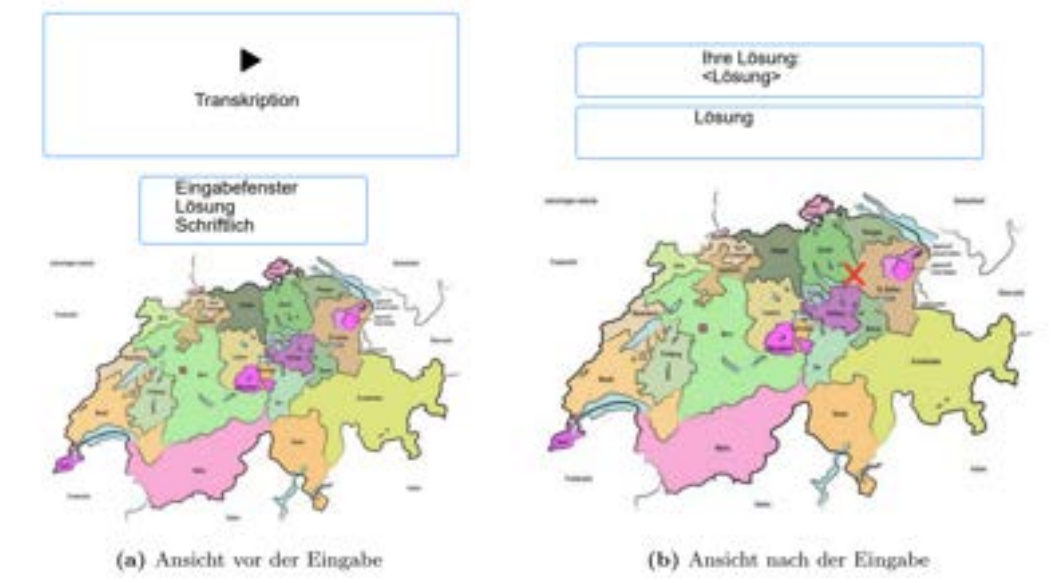
Das Dialektsample sollte über einen Play-Button abgespielt werden können. Für die Eingabe des Herkunftsortes bietet sich eine Karte an, die technisch etwas schwieriger zu gestalten wäre, jedoch dem User die Lokalisierung der vermuteten Herkunftsgegend massiv erleichtern würde. Als Alternative könnte ein Dropdown-Menü eingesetzt werden, aus dem der User eine Postleitzahl auswählen könnte.

User Story

Als Nutzer mit Muttersprache Schweizerdeutsch möchte ich lernen, die Herkunft eines schweizerdeutschen Sprechers so genau wie möglich zu bestimmen, damit ich Schweizer im Alltag nach ihrer Herkunftsgegend lokalisieren kann.

Tasks

- Ich kann ein Dialekt-Sample abspielen lassen.
- Ich kann den vermuteten Herkunftsort angeben.
- Nach der Eingabe wird mir der korrekte Ort angezeigt.
- Es wird mir auch die Distanz des korrekten zum vermuteten Ort angezeigt.



(a) Ansicht vor der Eingabe (b) Ansicht nach der Eingabe

Das Anzeigen des korrekten Herkunftsortes sollte idealerweise direkt in der Karte geschehen. Falls die Karte nicht zur Verfügung steht, könnte der korrekte Herkunftsort dem gerateten in schriftlicher Form gegenübergestellt werden, inklusive Angabe des Kantons zur besseren Lokalisierung. In beiden Fällen sollte die Distanz zwischen korrektem und geratenem Ort angezeigt werden, damit der User seine Performance einschätzen und bestenfalls bei weiteren Versuchen steigern kann.

3.2 Implementation

Die Implementierung gliederte sich in zwei Teile: Zum Einen musste entschieden werden, wo und wie die Dialekte-Raten-Erweiterung ins bestehende Projekt eingefügt würde, zum Anderen galt es die neuen Features zu erstellen. Nachfolgend werden diese beiden Schritte geschildert.

3.2.1 Integration ins bestehende Projekt

Wie in Kapitel 2.2 bereits impliziert, wurde die neue Seite aufgrund der ähnlichen Interaktionsweise als zusätzlicher Tab im Bereich der Nutzer-Kontributionen (Aufnehmen/Prüfen) zum Projekt realisiert. An der Stelle der schriftlichen Sätze wird die Karte eingeblendet, der Play-Button wird vom Prüfen-Tab übernommen. Darunter werden das Textfeld für die schriftliche Eingabe sowie der Absende-Button placiert. Alle anderen Buttons wurden entfernt, da die Tastenkürzel nicht benötigt und die Voice-Samples nicht in Fünfer-Gruppen geladen werden, sondern eine Aufnahme nach der anderen angezeigt, geraten und verarbeitet wird.

Der Hauptcode der neuen Seite wurde in den Files `guess.tsx` und `guess.css` unter `web/src/components/pages/contribution/guess` abgelegt. Um einen neuen Tab zu erzeugen, war es nötig, eine neue URL zu registrieren, indem eine zusätzliche Route im File `web/src/components/layout/content.tsx` eingefügt und in `urls.ts` sowie `contribution.tsx` darauf verlinkt wurde. Diese lautet analog zu den bereits bestehenden `../speak` und `../listen` auf `../guess`.

Das Codegerüst für die `GuessPage` wurde von der `ListenPage` übernommen, um das Design und den React-Lifecycle der anderen beiden Seiten zu bewahren. Das entsprechende Klassendiagramm mit ausgewählten Abhängigkeiten, Feldern und Methoden ist in Abbildung 7 dargestellt. Darauf aufbauend wurden die entsprechenden Features für die `GuessPage` implementiert.

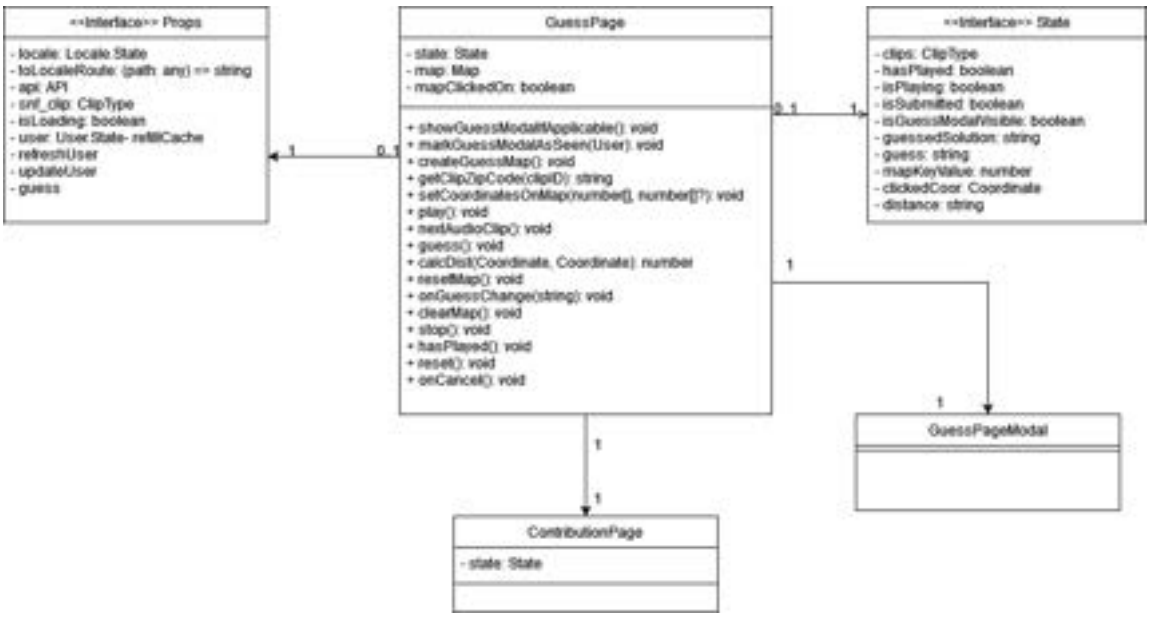


Abbildung 7: Klassendiagramm der GuessPage mit ausgewählten Abhängigkeiten.

3.2.2 Implementation der Features

Nachfolgend werden die neu hinzugefügten Features und deren Umsetzung genauer erläutert.

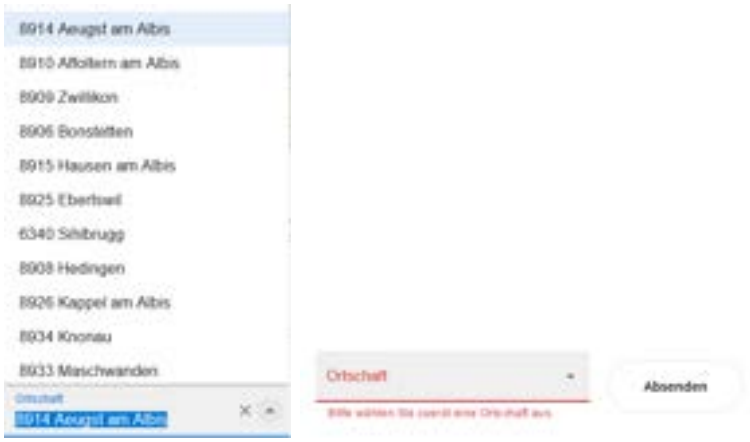
Kurzanleitung Die Kurzanleitung existierte schon auf der Aufnehmen- und der Prüfen-Seite und wurde auch für die Rate-Seite übernommen. Sie wird jeweils angezeigt, wenn ein User zum ersten Mal auf den entsprechenden Tab gelangt, und verschwindet für den Rest seines Aufenthalts auf der Webseite, sobald er sie geschlossen hat (auch nach einem Aktualisieren oder zwischenzeitlichen Verlassen des Tabs). Dafür muss die Information, ob der User die Anleitung bereits gesehen hat, abgespeichert werden. Die Kurzanleitung selbst ist ausserhalb des `guess.tsx`-Files abgelegt, nämlich unter `web/src/components/guess-page-modal/GuessPageModal.tsx`. Der Text besteht aus drei Komponenten: zwei `Localized`-Feldern (`title` und `body`) und einem direkt in der export-Funktion definierten Abschnitt. In der `GuessPage` ist die Kurzanleitung als `GuessPageModal` im `return`-Teil der `render`-Methode eingebaut.

Die Markierung der Kurzanleitung als gesehen geschieht in der `GuessPage`-Funktion `markGuessModalAsSeen`. Bei nicht angemeldeten Usern wird die Information im lokalen Speicher festgehalten, bei angemeldeten Usern ein Vermerk in der Datenbank angebracht, damit die Anleitung auch beim nächsten Aufruf der Webseite nach dem Einloggen nicht mehr gezeigt wird. Ob die Kurzanleitung gezeigt werden soll, wird in der Funktion `showGuessModalIfApplicable` überprüft, die zu Beginn des React-Lifecycles und dann bei jedem Update aufgerufen wird. Wenn die Kurzanleitung angezeigt werden soll, wird das Attribut `isGuessModalVisible` des State auf `true` gesetzt und beim nächsten `render` entsprechend verwertet.

Bei nicht angemeldeten Usern wird die Info, ob die Kurzanleitung bereits gesehen wurde, aus den Cookies gezogen. Bei angemeldeten Usern stammt sie aus den Usersettings, die aus der Datenbank geladen werden. Die Definition der Usersettings befindet sich im File `common/settings.ts`.

Eingabefeld Das schriftliche Eingabefeld für eine Vermutung für die Herkunft eines Sprechers wurde als Autocomplete-Komponente von Material-UI realisiert. Dieses erlaubt, im `options`-Attribut eine Liste mit vordefinierten Vorschlägen mitzugeben, aus denen der Nutzer dann aus einem Dropdown-Menü einen Vorschlag auswählen kann (siehe Abbildung 8a. Die vordefinierten Ortschaften werden in der Funktion `getMunicipalities` geladen, die auf ein JSON-File unter `web/src/components/pages/ profile/info` zugreift. Darin waren für die Erleichterung des Registrierungsprozesses des Users bereits alle Ortschaften der Schweiz abgespeichert und mussten nur noch entsprechend aufbereitet werden. Das Eingabefeld muss allerdings nicht nur direkte Usereingaben verarbeiten, sondern auch Orte, die der User auf der Karte anklickt, abfüllen. Dies passiert durch das Setzen des Attributs `value` mit `guess`, das den geratenen Ort des Nutzers enthält. Wenn der User sei-

ne Vermutung via Eingabefeld korrigiert, muss auch dies wieder korrekt abgefüllt werden. Dafür existiert die Methode `onGuessChange`, die aufgerufen wird, sobald sich der Eintrag im Textfeld ändert. Wenn der User einen Rateversuch absenden will, ohne dass er zuvor einen Ort ausgewählt hat, wird er über einen Textfeld-Error darauf aufmerksam gemacht (siehe Abbildung 8b).



(a) Eingabefeld mit Dropdown-Menü.

(b) Eingabefeld, wenn leerer Versuch abgeschickt wird.

Abbildung 8: Das schriftliche Eingabefeld.

Korrekte Lösung Um die Distanz zwischen der vermuteten und der tatsächlichen Herkunft des Sprechers der Aufnhame zu berechnen, muss als Erstes die korrekte Postleitzahl aus der Datenbank geladen werden. Der Zugriff auf die Datenbank geschieht in der `getClipZipCode`-Methode, verläuft über die beiden `API`-Klassen auf der Web- und der Server-Seite und gelangt dann an die `DB`-Klasse im Back-End. Sobald die Postleitzahl bekannt ist, werden über die Funktion `getCoordinatesFromZipcode` die geographischen Koordinaten der Ortschaft extrahiert: Auf der Seite nominatim.openstreetmap.org lassen sich via API-Call und unter Angabe der Postleitzahl deren Koordinaten abfragen. Wenn dann die Koordinaten der vermuteten Lösung ebenfalls bereitstehen, wird mittels der `calcDist`-Methode die Distanz zwischen der geratenen und der tatsächlichen Herkunft mathematisch berechnet.

Sobald der Rateversuch des Users abgesendet wurde, verschwindet die Instruktion und an ihre Stelle tritt die Auswertung des Versuchs. Diese umfasst die Angabe der Distanz zwischen vermutetem und tatsächlichem Herkunftsort sowie eine grobe Einordnung, wie gut der User abgeschnitten hat (siehe Abbildung 9). Gleichzeitig erscheint auf der Karte die Luftdistanz zwischen korrektem und geratenem Ort (siehe nächsten Abschnitt **Karte** resp. Abbildung 12).

Abbildung 9: Feedback auf Rateversuch.

Karte Für die Anzeige der Schweizer Landkarte wurde das kostenlose Open-Source-Tool OpenLayers[2] und die Basiskarte von OpenStreetMap verwendet. OpenLayers ermöglicht es, Features in Form von Layern über die Karte zu legen, beispielsweise Positionsmarker oder Informationen in Form von Popups. Die Instanziierung der Karte findet in der Methode `createGuessMap` im Konstruktor der `GuessPage` statt, die Karte selbst wird als Klassenfeld gespeichert. Die Verknüpfung mit dem HTML-div-Element in der `render`-Methode, in welchem die Karte schlussendlich angezeigt wird, geschieht indessen erst in der `componentDidMount`-Methode, wenn die Mounting-Phase des React-Lifecycles beinahe abgeschlossen ist. Würde die Verknüpfung mit dem div-Element früher passieren, würde die Karte nicht angezeigt werden, da dann das HTML-Element noch nicht zur Verfügung steht (die `render`-Methode wird erst gegen Schluss der Mounting-Phase aufgerufen). Wenn umgekehrt die Instanziierung der Karte später geschähe, könnte die Seite nicht geladen werden, weil die Karte zu Beginn der Mounting-Phase der Seite schon bereitstehen muss.

Neben dem grundlegenden Layer, das die Karte selbst beinhaltet, werden der Karte bei der Erstellung zwei weitere Layer hinzugefügt. Das eine davon ist für die Anzeige der Kantonsgrenzen zuständig (siehe Abbildung 10) und lädt die dafür benötigten Daten aus dem File `swiss_cantons.geojson`, das direkt im `web`-Ordner abgespeichert ist. Die Daten, welche die Koordinaten der Kantonsgrenzen in Form von Polygonen enthalten, stammen vom Bundesamt für Landestopografie Swisstopo [13] und wurden mithilfe von dessen REFRAME-Dienst [14] in geographische Koordinaten (Längen-/Breitengrade, mit denen auch die OpenLayers-Karte arbeitet) umgewandelt.

Das andere Layer ist dafür verantwortlich, die diversen Positionsmarker anzuzeigen und wieder zu löschen. Ein Marker wird mittels `ClickEvent` bei Klick auf die Karte hinzugefügt (siehe Abbildung 11), sofern der Klick innerhalb der Schweizer Grenzen geschah und sich der Zyklus des Ratespiels nicht in der Phase des Lösung-Anzeigens befindet (in dieser Phase sind alle Interaktionen mit der Karte ausser Scrollen/Zoomen gesperrt). Wenn der User auf einen anderen Ort klickt, wird der erste Marker entfernt und ein neuer eingefügt. Ebenso wird ein Marker auf der Karte positioniert, wenn der User eine Ortschaft aus dem Textfeld

[2]Das v.a. für mobile Geräte noch besser geeignete Tool Leaflet war leider aufgrund von veralteten Projekt-Dependencies nicht verwendbar.

Abbildung 10: Landkarte der Schweiz inklusive Kantonsgrenzen.

auswählt. Dafür wird wiederum eine Postleitzahl in Koordinaten umgerechnet.

Abbildung 11: Marker auf Zürich.

Sobald der Nutzer einen Rateversuch abgibt und die Lösung angezeigt wird, wird die Karte wieder zentriert und auf minimalen Zoom eingestellt, damit der Nutzer auf einen Blick sowohl seine Rateposition als auch die korrekte Herkunft inklusive Abstand sieht (siehe Abbildung 12). Die Karte wird dann, wie bereits erwähnt, für alle Maus-Aktionen ausser Scrollen gesperrt, sodass er keine weiteren Marker setzen kann. Wenn er aber möchte, kann er nochmals hineinzoomen, um z.B. die tatsächliche Herkunftsregion genauer zu inspizieren. Wenn er dann durch Klick auf den „Weiterraten"-Button in die nächste Rate-Phase eintritt, wird die Karte mittels Minimalzoom und Rezentrierung erneut ausgerichtet.

Audio-Clips Während das Laden der Audio-Clips bei der `ListenPage` über den Redux-Store und die Props erfolgt (siehe Kapitel 2.3), ruft die `GuessPage` direkt die API-Methode auf und lädt den Clip aus der Datenbank resp. dem Bucket. Da das Dialekte-Raten jeweils nur einen Clip aufs Mal benötigt, wäre es zu aufwendig gewesen (effizienzmässig wie code-technisch), den einen Clip ebenfalls über den Cache zu laden. Entweder wären bei jeder neuen Raterunde alle Clips (auch jene der `ListenPage`, die für das Dialekte-Raten nicht benötigt werden) neu geladen worden, oder es hätten sehr viele Änderungen am bestehen-

Abbildung 12: Luftdistanz auf der Karte zwischen vermutetem und tatsächlichem Herkunftsort.

den Code vorgenommen werden müssen.

Das Interaktionsdiagramm fürs Laden des Clips wird in der Abbildung 13 dargestellt, wiederum ohne den Umweg über die API-Server-Klasse. Der Aufruf geschieht im Abschluss der Mounting-Phase und wird in der `Clip`-Klasse auf der Server-Seite mit dem Parameter Client_Id versehen, der benutzt werden kann, um jene Clips auszuschliessen, die der eingeloggte User bereits geraten hat (diese Funktion ist aktuell aber noch nicht implementiert). Da die Audiodaten aus einem anderen Sammlungsset und voraussichtlich auch aus einem anderen Bucket stammen als die Prüf-Daten und sich auch sonst einige Details u.a. in den Metadaten der Clips unterscheiden, wurde ein komplett neuer „Code-Pfad“ (d.h. eigene Methoden) für sie erstellt.

Abbildung 13: Interaktionsdiagramm für das Laden eines Clips in die `GuessPage`.

Speichern des Rateversuchs Um die Rateversuche von Usern für allfällige spätere Zwecke und Auswertungen festzuhalten, wurde eine zusätzliche Tabelle `user-guesses` mit den Attributen `id` (für den jeweiligen Rateversuch), `client_id` (für den ratenden User), `clip_id` (für den geratenen Audioclip) und `zipcode` (für die geratene Ortschaft) angelegt. Die entsprechenden Files sind im `migrations`-Ordner abgespeichert und werden bei Start des Projekts automatisch lokal ausgeführt. Sobald der User einen Rateversuch abschickt, wird über die Props eine `guess`-Aktion ausgeführt, die im File `web/src/stores/clips.ts` definiert ist. Diese leitet die Anfrage über die API-Schnittstelle an den Server weiter, der

dann das Insert-Statement in die `user-guesses`-Tabelle vornimmt.

3.3 Verwendete Daten

Wie in Kapitel 1.1 bereits erwähnt, standen für die vorliegende Erweiterung der Projektsammlung neue Audiodaten zur Verfügung, die für das Dialekte-Raten eingesetzt wurden. In diesem Kapitel werden jene Daten genauer beschrieben: zum Einen, wie die Daten gesammelt, zum anderen, wie sie ins Projekt importiert wurden.

3.3.1 SNF-Datensatz

Die neuen Daten stammen aus dem sogenannten STT4SG-350-Korpus von Michel Plüss et al. [5], für den zwei Sammlungen durchgeführt wurden: einmal fürs Testset (STT4SG-350 Test) und einmal fürs Trainingsset (STT4SG-350 Train). Für die Dialektsammlung wurde nur das Trainingsset verwendet, das allein über 200'000 Aufnahmen umfasst. Das Ziel für dieses Set war, für jede Dialektregion ca. 32'500 Sätze einsprechen zu lassen, mit ca. 30 Speakern pro Region, wobei sich die Sätze von Region zu Region unterscheiden. Die verwendeten Sätze stammen zu 94% aus Schweizer Zeitungen, zu 6% aus Protokollen von zwei Schweizer Parlamenten. Von den Sprechern wurden neben der Postleitzahl ihres Wohnorts auch das Alter (in Zehn-Jahres-Schritten, also z.B. „twenties“, „sixties“ etc.) und das Geschlecht erhoben. Dabei sind Personen aus allen Deutschschweizer Kantonen vertreten ausser einem (Appenzell Ausserrhoden). Die Verteilung auf die Kantone sowie auf die Dialektregionen ist in Abbildung 14 ersichtlich, wobei sich die Zahlen auf die absolute Anzahl Samples beziehen.

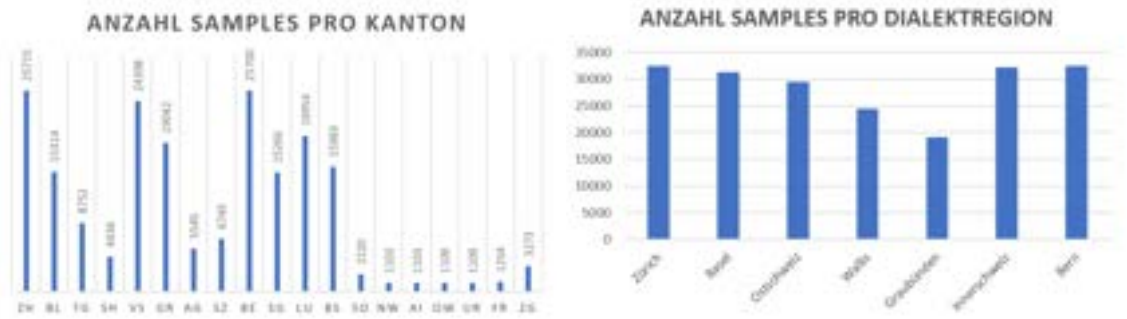


Abbildung 14: Herkunft der Samples nach Kanton resp. Dialektregion.

Obwohl die Anzahl Samples pro Kanton (links in Abbildung 14) ein eher heterogenes Bild ergibt, gleicht sich dieses wieder aus, wenn man die Dialektregionen miteinbezieht (rechts in Abbildung 14): Bis auf das Wallis und das Graubünden sind die Dialektregionen etwa gleichmässig vertreten. Für die Dialekte-Raten-Applikation sind ausgewogene Daten

keine zwingende Voraussetzung, aber dennoch ist eine gewisse Diversifizierung von Vorteil, um die Spannung für den Nutzer grösser zu halten. Wäre eine Region übermässig stark repräsentiert, könnten mehr Zufallstreffer gelandet werden, und der Lerneffekt würde vermindert oder bliebe mit der Zeit ganz aus.

Obwohl nicht davon auszugehen ist, dass eine Mehrheit der Nutzer tatsächlich auf Postleitzahl-Ebene raten wird, d.h. granulare Dialektunterschiede zwischen benachbarten Dörfern ausmachen kann – viele werden sich eher an kantonalen Grenzen orientieren –, würde die vorliegende Applikation dies theoretisch erlauben. Deshalb ist auch die Anzahl Herkunftsgemeinden von einem gewissen Interesse: Das Trainingsset umfasst 162 verschiedene Postleitzahlen.

Die Menge an Herkunftsgemeinden wird selbstverständlich bedingt durch die Anzahl unterschiedlicher Sprecher, die für die Sammlung aufgenommen wurden. Beim Trainingsset handelt es sich um 215 Sprecher, was – wie vom Sammelprojekt angestrebt – im Schnitt etwa 30 Sprecher pro Region bedeutet. Die genaue Verteilung der Sprecher auf die Dialektregionen ist in Graphik 15 abgebildet. Bei dieser Anzahl Sprechern kann davon ausgegangen werden, dass die Nutzer nicht in absehbarer Zeit anhand der Stimmen raten lernen (d.h. die Stimme eines Sprechers als die der „Frau aus Thun“ erkennen).

Abbildung 15: Anzahl Sprecher pro Dialektregion.

3.3.2 Import der Daten

Für den Import der Daten wurde ein Python-Skript erstellt, das sowohl die Audioclips in den S3-Bucket lädt als auch die Metadaten der Clips in die Datenbank schreibt. Das Skript befindet sich unter `/docker/SNF_data_migration_trainset.py`. Zuerst legt es, falls noch nicht vorhanden, die Tabelle `snf_clips` in der Datenbank an und befüllt diese dann mit den Einträgen für die Audiofiles, die im zu den Audiodaten gehörigen `tsv`-File enthalten sind. Die Tabelle verfügt, im Gegensatz zur `clips`-Tabelle des ursprünglichen Projekts (siehe 2.4.2), über das Attribut `zipcode`, sodass der Herkunftsort der Aufnahme leicht ermittelt werden kann. Eine Übersicht über die `snf_clips`-Tabelle und ihre Attribute findet sich in

Abbildung 16.

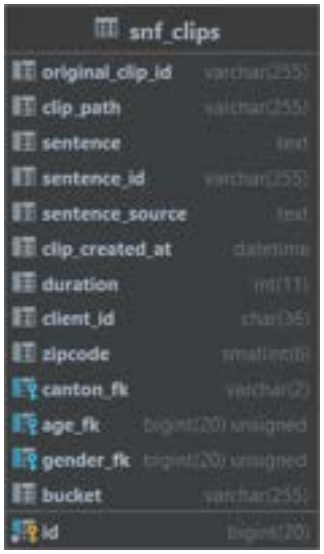


Abbildung 16: Tabelle `snf_clips` mit Attirbuten.

Die Audioaufnahmen selbst werden mit einem Key, der gleich formatiert ist wie bei den über die Dialektsammlung erhobenen Audiodaten, in den Bucket geladen.

Der End Point, der Bucket sowie die Credentials wurden folgendermassen konfiguriert:

```
s3 = boto3.resource('s3',
                    aws_access_key_id='local-identity',
                    aws_secret_access_key='local-credential',
                    endpoint_url='http://localhost:8080')
s3_bucket_name='common-voice-corpus'
my_bucket=s3.Bucket(s3_bucket_name)
```

3.4 Arbeitsweise

3.4.1 Planung mit Agile und Jira

Für ein Studienprojekt wie das vorliegende bot sich der iterative Agile-Ansatz an, da er einigen Planungsfreiraum bietet und die effiziente Behandlung unerwarteter Probleme ermöglicht, die bei der Erweiterung eines bereits bestehenden Projekts zu erwarten waren. Als Planungstool wurde Jira und daran gekoppelt das Dokumentationstool Confluence eingesetzt.

Als die umzusetzende User Story feststand, wurde diese User Story als Epic festgelegt, von

dem ausgehend die einzelnen Tasks als Stories definiert und diese schliesslich in weitere Subtasks aufgebrochen wurden. In der tabellarischen Darstellung auf der nächsten Seite ist beispielshaft abgebildet, wie diese Aufteilung vonstatten ging.

User Story Als Nutzer möchte ich wissen, wie gut ich die Herkunft eines schweizerdeutschen Sprechers errate, damit ich mich im Raten verbessern kann.	$\Longrightarrow$	**Epic** Als Nutzer möchte ich wissen, wie gut ich die Herkunft eines schweizerdeutschen Sprechers errate, damit ich mich im Raten verbessern kann.
Tasks • Ich kann ein Dialekt-Sample abspielen lasssen. • Nach Eingabe des Herkunftsortes wird mir der korrekte Ort angezeigt. • Es wird mir auch die Distanz des korrekten zum vermuteten Ort angezeigt.	$\Longrightarrow$	**User Stories** • SP-5 Dialekt-Sample abspielbar machen. • SP-6 Lösung anzeigen. • SP-8 Distanz zw. korrektem und vermutetem Ort angeben.
User Story • SP-6 Distanz zw. korrektem und vermutetem Ort angeben.	$\Longrightarrow$	**Subtasks** • SP-28 Herkunftsort auf Karte anzeigen. • SP-27 Distanz in km in UI anzeigen. • SP-34 Auswahl aus Autocomplete auf Karte anzeigen.

Die Tasks wurden auf ingesamt vier Sprints verteilt, die je zwei Wochen dauerten. Durchschnittlich betrug der Arbeitsfortschritt pro Sprint 15 Story Points.

3.4.2 Versionskontrolle mit Git

Für die Versionierung hat sich Git resp. GitHub angeboten, da die bestehende Applikation bereits in einem GitHub-Repository abgelegt ist. Für die Dialekte-Raten-Erweiterung wurde ein Fork vom ursprünglichen Projekt angelegt, in dem dann die neuen Features implementiert wurden. So sollte die potentielle Integration in die Originalfassung möglichst einfach vonstatten gehen können.

Für jeden Task wurde ein eigener Feature-Branch angelegt, der nach Abschluss des Tasks wieder auf den Dev-Branch germerged wurde.

3.4.3 Eingesetzte Tools

In Bezug auf die einzusetzende Software konnte auf die bereits für die ursprüngliche Applikation verwendeten Tools zurückgegriffen werden. Für die lokale Entwicklung waren schon ein gemockter S3-Bucket und eine Mysql-Datenbank vorhanden, die beide in einem Docker-Container deployed sind (siehe Kapitel 2.4). Eine Anleitung, wie diese in Betrieb zu nehmen sind, findet sich im `docs`-Ordner des vorliegenden Projekts im File `Kickstart.md`.

Als Entwicklungsumgebung wurden Visual Studio Code und WebStorm eingesetzt. Für das Debugging auf der Datenbank wurde Datagrip verwendet.

4 Resultate

Dieses Kapitel stellt die Ergebnisse der vorliegenden Arbeit unter Bezugnahme auf das Projektziel vor. Für eine Einschätzung der Tauglichkeit der entwickelten Applikation wurden qualitative User-Tests durchgeführt, deren Erkenntnisse nachfolgend präsentiert werden. Ausserdem werden verschiedene Aspekte des Entwicklungsprozesses der Software reflektiert.

4.1 Design der Webseite

Das finale Design der Dialekte-Raten-Webseite ist in der Abbildung 17 ersichtlich. Im Gegensatz zu den Sprechen- und Prüfen-Tabs ist sie mit einem Titel versehen, damit auch Nutzer mit einem Direktlink (die also nicht über die offizielle Schweizer-Dialektsammlung-Webseite dahingelangen) wissen, was sie erwartet.

Abbildung 17: User Interface Dialekte raten.

4.2 User-Tests

Insgesamt haben sich sechs User mit Muttersprache Schweizerdeutsch zur Verfügung gestellt, die die Webseite in der Anwesenheit eines der Urheber der Arbeit getestet und

anschliessend einen Fragebogen ausgefüllt haben. Der Fragekatalog findet sich im Anhang im Kapitel 7.3, die Ergebnisse der Umfrage sowie der Beobachtungen der Autoren werden nachfolgend zusammenfassend wiedergegeben.

4.2.1 Usability

Aufgabenangemessenheit Hierin waren sich alle Test-User einig: Die Handhabung der Webseite ist einfach und die Schritte bis zur Lösung minimal. Auch die Benutzereingaben müssten nur einmal erfolgen – ausser für eine Person, die berichtete, dass sie manchmal mehrmals auf die Karte klicken musste, bis das Textfeld mit der gewählten Ortschaft abgefüllt wurde. Die Beobachtungen ergaben, dass dies mindestens bei zwei weiteren Personen der Fall war, auch wenn sie es in der nachträglichen Befragung nicht vermerkten.

Selbstbeschreibungsfähigkeit Die meisten User hielten die Webseite für selbsterklärend und verstanden, was sie zu tun hatten bzw. welche Möglichkeiten zur Eingabe des vermuteten Herkunftsortes zur Verfügung standen (obwohl die überwiegende Mehrheit die Karte nutzte und nur einer auch das Eingabefeld). Zwei bemerkten allerdings die mangelhafte Beschreibung der Zoom-Funktion. (Ein anderer brachte den Einbau einer Zoom-Funktion gar als Verbesserungsvorschlag.) Tatsächlich nutzte von den sechs Usern nur einer die Zoom-Funktion der Karte. Ein weiterer versuchte es, doch scheiterte an der Handhabung – was jedoch auch mit dem verwendeten Notebook, das nicht ihm gehörte, zu tun haben könnte.

Steuerbarkeit Hier ergab sich ein durchzogenes Bild: Auf die Frage, ob Aktionen rückgängig gemacht oder korrigiert werden können resp. ob jederzeit abgebrochen werden könne, antworteten zwei User mit Nein, zwei hielten die Frage für nicht zutreffend (weil sie es nicht ausprobiert hätten) und zwei antworteten mit Ja. Es ist nicht bekannt, ob die User, die negativ oder neutral geantwortet haben, tatsächlich nie versucht haben, eine Eingabe zu korrigieren/zu löschen, oder ob sie auf die Korrektur der bereits abgesendeten Vermutung anspielten.

Erwartungskonformität Die Funktionsweise der Webseite scheint den Erwartungen der Test-User zu entsprechen: Keiner hatte bezüglich ihrer Reaktionen etwas zu bemängeln.

Funktionale Fehler Die Test-User berichteten einstimmig, dass ihnen keine funktionalen Fehler aufgefallen seien. Tatsächlich gab es während des Ratens keine Zwischenfälle, auch bei Verhaltensweisen der User, mit denen die Autoren nicht gerechnet hatten (z.B. überstürztes Klicken auf die Buttons, mit denen man in die nächste Phase des Rate-Zyklus gelangt).

Design Die Ansicht der Webseite gefiel den Test-Usern grundsätzlich. Drei User schlugen vor, die Karte etwas grösser zu machen, womit sie auf deren Leserlichkeit anspielten

(wobei diese Befunde wiederum nicht eindeutig getrennt werden können von den Display-Einstellungen des verwendeten Notebooks). Ausserdem hatten drei User etwas Mühe mit dem Verständnis der Tonaufnahmen, von denen einige zu leise, zu kurz oder zu undeutlich waren. Einer schlug vor, die hochdeutschen Verschriftlichungen der Aufnahmen mit einzublenden, um Verständnisschwierigkeiten vorzubeugen.

Reaktionszeiten Die Reaktionszeiten der Webseite waren für alle Test-User tolerabel, für einen zumindest „mehr oder weniger“.

4.2.2 User-Fazit

Die vielleicht wichtigste Frage des Katalogs lautete, ob das Dialekte-Raten Spass gemacht habe. Hier bejahten die Test-User durchgehend, vier antworteten mit Verstärkern wie „sehr“, „super“ oder „cool“. Eine Person meinte, das Ratespiel berge Suchtpotential. Ein anderer User hatte das Spiel in einer Gruppe von etwa 6 Personen getestet (er bediente die Webseite, die anderen hörten sich die Aufnahmen mit an und gaben Vermutungen ab), wo auch das Potential der Applikation zum Wettbewerb ersichtlich wurde, indem jeder den anderen bzgl. Treffsicherheit zu überbieten versuchte.

Im Schnitt rieten die Test-User schätzungsweise etwa 50 Kilometer daneben. Am meisten Mühe bereiteten hierbei das Auseinanderhalten von Nachbarkantonen (z.B. Bern und Luzern oder Zürich und Schwyz), die genaue Lokalisierung in grossen Dialektregionen/-Kantonen wie Bern und Graubünden und Sprecher, die vermutlich einen anderen (oder nicht sehr stark ausgeprägten) Dialekt sprachen als den angegebenen. Dennoch gelang es einem User einmal (mit Unterstützung der Gruppe, die mit ihm spielte), den Herkunftsort auf Postleitzahlebene genau zu treffen – Distanz 0 Kilometer. Auch andere erzielten im Schnitt jedes zweite oder dritte Mal Ergebnisse von unter 10 Kilometern.

4.3 Reflexion des Software-Entwicklungs-Prozesses

Das Aufbauen auf einer bestehenden Applikation brachte sowohl Vorteile als auch Nachteile mit sich. Auf der einen Seite war es nützlich, im bereits vorhandenen Code eine funktionierende Basis für die grundlegenden Abläufe sowie eine Vorlage für ähnliche Features zu haben. Zudem konnten einige Komponenten der Aufnehmen- und Prüfen-Tabs auch in die `GuessPage` eingearbeitet werden. Auf der anderen Seite stellte die Einarbeitung in den bestehenden Code eine Herausforderung dar, die viel Zeit kostete. Gewisse Teile der Dialekte-Raten-Applikation wurden aufgrund der beschränkten Durchführungszeit implementiert, als noch nicht alle Funktionsweisen der Basis-Applikation verstanden waren, wie z.B. der Datenimport, wodurch sich gewisse Abweichungen von den Originalstrukturen ergaben (siehe Kapitel 3.3.2). Auch erwies sich die Verwendung mancher Komponenten im Nachhinein als suboptimal. Die `ContributionPage` zum Beispiel, die einen elementaren Baustein in allen Tabs mit User-Beiträgen darstellt, verlangt viele Parameter, die in der

Raten-Seite nicht mehr vorkommen. Diese mussten alle als optional markiert werden. Andere Versuche, den neuen Teil möglichst analog zum vorhandenen zu konzipieren, stifteten einige Verwirrung, beispielsweise das Laden der Clips: Zuerst wurde versucht, die fürs Raten benötigte Aufnahme analog zur `ListenPage` über den Cache zu laden (siehe Kapitel 2.3). Erst nach einer Weile wurde klar, dass sich damit der Ladeprozess unnötig verkomplizierte und es sinnvoller war, den einzelnen Clip losgelöst vom Cache aus der Datenbank zu holen (siehe Kapitel 3.2.2 Abschnitt **Audio-Clips**).

Die in diesem Kapitel vorgestellten Resultate der vorliegenden Arbeit werden im nächsten Kapitel in Bezug auf ihre Relevanz besprochen und in ein Fazit überführt. Aus den User-Feedbacks ergeben sich einige denkbare Fortführungen sowie Optimierungen der vorliegenden Arbeit.

5 Diskussion und Ausblick

Das Ziel der vorliegenden Arbeit war es, eine Applikation zu erstellen, die es Schweizer Muttersprachlern ermöglicht, spielerisch ihr Wissen über die Schweizer Dialektlandschaft zu ermitteln und zu erweitern. Dies wurde in Form einer Dialekte-Raten-Webseite umgesetzt und getestet. Es wurden sechs qualitative User-Tests durchgeführt, was in keiner Weise als repräsentativ für die gesamte Deutschschweizer Bevölkerung gelten kann, jedoch trotzdem erste wertvolle Hinweise darauf liefert, wie die Applikation von der Zielgruppe rezipiert wird.

Da sich die Test-User überwiegend positiv zur Dialekte-Raten-Applikation geäussert haben, ist davon auszugehen, dass diese tatsächlich ihrem intendierten Zweck dienen könnte, wenn sie publiziert und angemessen beworben würde. Zwar war in den kurzen Sequenzen, in welchen die Applikation ausprobiert wurde, kein nennenswerter Lerneffekt zu beobachten – dafür müsste sie deutlich länger im Einsatz sein. Aber das Interesse, das eigene Wissen über die Dialekte zu konsolidieren und auch gegenüber anderen hervorzuheben, war insbesondere bei der Gruppe deutlich zu spüren. Und auch die Einzeltester wurden angespornt vom Ehrgeiz, in der nächsten Runde ein noch besseres Ergebnis zu erzielen als in der letzten.

Aus diesen Befunden ergeben sich mehrere mögliche Weiterführungen des vorliegenden Projekts – einige davon finden sich als bereits formulierte User Storys im Anhang. Um den Spielcharakter der Applikation weiter auszubauen, könnten die Rateversuche der User ausgewertet und in Form einer Statistik für den User einsichtbar gemacht werden (siehe Anhang Kapitel 7.4.4). Des Weiteren könnte das Messen mit anderen Usern ermöglicht werden, indem auch die Ergebnisse anderer User bei einem bestimmten Sample angezeigt (siehe Anhang Kapitel 7.4.5) und in einer Rangliste aller User zusammengefasst werden (siehe Anhang Kapitel 7.4.6). Eine Alternative wäre, die Applikation für mehrere Personen gleichzeitig spielbar zu machen, indem jeder Mitspieler auf seinem Endgerät einen Tipp abgeben kann und ihm dann die Vermutungen seiner Mitspieler zusammen mit der Lösung angezeigt werden, inklusive Auswertung am Ende. Um den Lerneffekt zu fördern, wäre es wiederum denkbar, zusätzlich zur geratenen Aufnahme weitere Aufnahmen aus der vermuteten Region einzublenden, um den Unterschied hörbar zu machen (siehe Anhang Kapitel 7.4.7). Falls sich das Raten auf Ortschaftsebene mit der Zeit (für einige User) als zu anspruchsvoll herausstellen sollte, wäre es möglich, das Raten auch auf Kantonsebene anzubieten (siehe Anhang Kapitel 7.4.3). Wenn man den Kontext und das Ziel der Dialektsammlung mit berücksichtigt, könnte man Nutzern, die eine eigene Aufnahme hochladen, auch Statistiken dazu liefern, wie sie von anderen Nutzern geographisch eingeschätzt werden.

Allerdings sollten auch die Verbesserungsvorschläge für die bestehende Applikation, die von den Test-Usern geäussert wurden, nicht ausser Acht gelassen werden. Um das Nutzererlebnis noch angenehmer zu gestalten, wäre es sinnvoll, die Zoomfunktion besser ersichtlich

zu machen, beispielsweise durch grössere Zoom-Buttons. Um die Steuerbarkeit durch den User zu erhöhen, könnte eine Clear-Funktion eingebaut werden, mit der sich alle Marker von der Karte löschen lassen. Auch die teilweise schwerfällige Zuordnung einer Postleitzahl zum angeklickten Punkt auf der Karte müsste verbessert werden. Da dies aktuell über eine externe API geschieht, die offensichtlich nicht allen Koordinaten eine Postleitzahl zuordnen kann, könnten Alternativen geprüft werden wie z.B. der Bezug von Ortsgemeinde-Geodaten über Swisstopo, die dann allenfalls lokal abgelegt werden müssten.

Falls die Erweiterung der Applikation in die bestehende integriert werden sollte, müsste beachtet werden, dass aktuell der Import der SNF-Daten über ein externes Python-Script erfolgt, das manuell gestartet werden muss. Für die Harmonisierung mit den bisherigen Datenbank-Migrationen wäre es sinnvoll, dieses Skript resp. zumindest die Teile, die die Datenbank-Tabellen erstellen, in ein `db-migrate`-File zu verschieben, das dann automatisch bei Projektstart eingelesen wird.

Im Nachhinein wäre es vermutlich besser gewesen, von Grund auf eine eigenständige Applikation zu implementieren, zumindest was den programmiertechnischen Aufwand und Anspruch betrifft. Die bestehende Applikation ist relativ schwerfällig und trägt einiges an Altlasten mit sich, was die programmatische Freiheit einschränkte und den Einsatz gewisser bevorzugter Paradigmen erschwerte. Trotzdem ist es nützlich, dass nun der Bezug zum Mutterprojekt bestehen bleibt, da es auch eine der ursprünglichen Absichten war, noch mehr Sprecher mit Muttersprache Schweizerdeutsch für die Dialektsammlung zu gewinnen. Gemäss den erzielten Resultaten birgt die vorliegende Applikation tatsächlich das Potential, die Zielgruppe auf spielerische Weise mit der Dialektsammlung in Berührung zu bringen – und vielleicht auch für deren grösseren Zweck zu begeistern.

6 Verzeichnisse

6.1 Literaturverzeichnis

[1] Bundesamt für Statistik. *Sprachen*. URL: https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/sprachen-religionen/sprachen.html. (abgerufen am: 09.12.2022).

[2] Christoph Neuenschwander. *Dialekte haben in der Schweiz hohes Prestige*. URL: https://www.solothurnerzeitung.ch/solothurn/kanton-solothurn/dialekte-haben-in-der-schweiz-hohes-prestige-ld.1675456. (abgerufen am: 09.12.2022).

[3] Thomas Kobel. *Der gruusigste Dialekt der Schweiz: Besser als sein Ruf!* URL: https://www.srf.ch/sendungen/einstein/der-gruusigste-dialekt-der-schweiz-besser-als-sein-ruf. (abgerufen am: 09.12.2022).

[4] swissnlp.org. *Die Schweizer Dialektsammlung*. URL: https://dialektsammlung.ch/de/about. (abgerufen am: 11.12.2022).

[5] Michel Plüss Jan Deriu Christian Scheller Yanick Schraner Claudio Paonessa Larissa Schmidt Julia Hartmann Tanja Samardzic Manfred Vogel und Mark Cieliebak. *Stt4sg-350: A speech corpus for all swiss german dialect regions.* In preparation, 2023.

[6] Linus Schöpfer. *Können Sie Schweizerdeutsch?* URL: https://www.tagesanzeiger.ch/koennen-sie-schweizerdeutsch-197383391667. (abgerufen am: 11.12.2022).

[7] Anna Rothenfluh. *Puttitsch ... whaaat? Verstehst du diese schweizerdeutschen Wörter?* URL: https://www.watson.ch/quiz/wissen/772410423-schweizerdeutsch-quiz-verstehst-du-diese-10-dialektausdruecke. (abgerufen am: 11.12.2022).

[8] Dominik Heeb. *Das Chochichästli-Orakel*. URL: http://home.datacomm.ch/heeb/dialect/. (abgerufen am: 12.12.2022).

[9] Adrian Leemann. *Dialäkt Äpp*. URL: https://apps.apple.com/us/app/dial%C3%A4kt-%C3%A4pp/id606559705. (abgerufen am: 12.12.2022).

[10] Universität Zürich. *Phonogrammarchiv der Universität Zürich*. URL: https://www.phonogrammarchiv.uzh.ch/de.html. (abgerufen am: 12.12.2022).

[11] Yves Scherrer. *Dialektkarten*. URL: http://dialektkarten.ch/. (abgerufen am: 12.12.2022).

[12] Yarn. *Safe, stable, reproducible projects*. URL: https://yarnpkg.com/. (abgerufen am: 13.12.2022).

[13] Bundesamt für Landestopografie swisstopo. *swissBOUNDARIES3D*. URL: https://www.swisstopo.admin.ch/de/geodata/landscape/boundaries3d.html. (abgerufen am: 19.12.2022).

[14] Bundesamt für Landestopografie swisstopo. *REFRAME*. URL: https://www.swisstopo.admin.ch/de/karten-daten-online/calculation-services/reframe.html. (abgerufen am: 19.12.2022).

6.2 Abbildungsverzeichnis

7 Anhang

7.1 Projektstruktur

Hier findet sich eine Auflistung aller Ordner und Files, die für das vorliegende Projekt erstellt (in rot) oder verändert wurden.

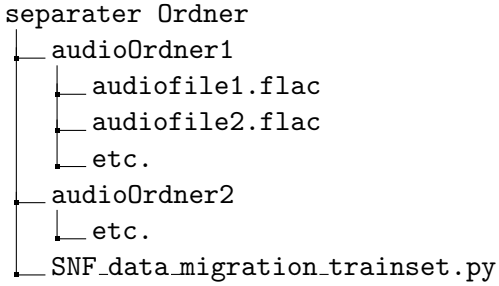
```
web
├── locales
│   └── de
│       └── messages.ftl
├── src
│   ├── components
│   │   ├── guess-page-modal
│   │   │   └── GuessPageModal.tsx
│   │   ├── layout
│   │   │   └── content.tsx
│   │   ├── pages
│   │   │   └── contribution
│   │   │       ├── guess
│   │   │       │   ├── guess.tsx
│   │   │       │   └── guess.css
│   │   │       ├── contribution.tsx
│   │   │       └── contribution.css
│   │   ├── primary-buttons
│   │   │   └── primary-buttons.tsx
│   │   └── app.tsx
│   ├── stores
│   │   └── clips.ts
│   ├── services
│   │   └── api.ts
│   └── urls.ts
├── package.json
└── swiss_cantons.geojson
server
└── src
    └── lib
        └── model
            └── db
                └── migrations
                    ├── 20221201114520-add-guess-flag-for-modal-to-user-clients-table.ts
                    └── 20221201161001-create-table-guesses.ts
```

7.2 Arbeiten mit dem Projekt

7.2.1 Projekt lokal in Betrieb nehmen

Hierfür wurde im Projekt eine separate Anleitung angelegt, die unter `docs/Kickstart.md` abgespeichert ist. Damit die Projekterweiterung ebenfalls funktioniert, müssen vor dem Start des Projekts mit `yarn start` zuerst die Audioaufnahmen in den S3-Bucket geladen werden. Dafür ist idealerweise ein separater Ordner (ausserhalb des Projekts) zu erstellen, in dem die Audiodateien entzippt werden. Die Ordnerstruktur (siehe unten) sowie die Dokument-Namen, die als Bucket-Key fungieren, müssen dabei beibehalten werden. Danach kopiert man das Python-File `SNF_data_migration_trainset.py` unter `docker` in den obersten Ordner und führt es bei laufendem Docker-Container (Datenbank *und* S3 Proxy) aus.

Ordnerstruktur:



7.2.2 Paketverwaltung mit yarn

Um die Abhängigkeiten der Node-Module, die in den *package.json*-Dokumenten festgelegt sind, zu installieren, kann folgender Befehl verwendet werden:

```
$ yarn
```

Um das Projekt zu starten:

```
$ yarn start
```

Neue Node-Module können hinzugefügt werden, indem erst das entsprechende Modul im *package.json*-File definiert wird (Caveat: Es gibt mehrere *package.json*-Files, je eins im *server*- und im *web*-Ordner sowie eins für das gesamte Projekt. Frontend- resp. Backend-Module müssen an den entsprechenden Orten installiert werden). Dies kann z.B. mittels folgendem Befehl geschehen:

```
$ cd web
$ yarn add <package-name>
```

Damit wird auch sichergestellt, dass die aktuellste Version des Moduls importiert wird. Anschliessend wieder yarn laufen lassen, um die Module zu installieren:

```
$ yarn
```

7.2.3 Anpassungen an der Datenbank

Wenn Anpassungen an der Datenbank vorgenommen werden müssen, beispielsweise eine neue Tabelle zu erstellen oder ein bestehender Key abzuändern ist, sollte dies mittels db-migrate geschehen.

```
$ db-migrate create <name-des-files>
```

Dann wird unter `server/src/lib/model/db/migrations` ein entsprechendes File erstellt, das zusätzlich zur Gewährleistung der Einzigartigkeit den Zeitstempel im Namen trägt. Dieses File kann dann unter der Methode `up` mit dem gewünschten SQL-Statement ergänzt und als ts-File abgelegt werden. Wenn das Projekt das nächste Mal gestartet wird, wird automatisch nach noch nicht durchgeführten Migrations gesucht und das neue File ausgeführt. Nach der Ausführung wird in der DB-Tabelle `migrations` ein entsprechender Eintrag vermerkt.

Wenn eine Anpassung fälschlicherweise geschehen ist, erstellt man am besten ein neues File mittels `db-migrate`, in welchem man die Änderung rückgängig macht (bspw. `drop table XY ...`). Beide Files sollten dann nicht auf GitHub geladen werden, damit sie bei den anderen Entwicklern nach dem nächsten Pull und Start nicht automatisch ausgeführt werden.

Wenn das SQL-Statement einen Fehler geworfen hat und kein Befehl ausgeführt wurde, kann man das File löschen und den entsprechenden Eintrag aus der *migrations*-Tabelle entfernen. Dabei muss aber berücksichtigt werden, dass unter `server/js/lib/model/db/migrations` jeweils eine js-Kopie der Migrations-Files abgelegt wird und diese sowie das automatisch erstellte `map`-File ebenfalls gelöscht werden müssen.

7.3 Fragen zur Usability

Folgende Fragen haben die Testuser beantwortet. Die Fragen wurden anhand den Usability-Kriterien der Norm ISO 9241-110 ausgearbeitet und bewusst so formuliert, dass auch Ja/Nein-Antworten möglich sind. Gleichzeitig wurde genügend Platz für allfällige Ausführungen gelassen (die meisten User füllten den Test von Hand aus).

- Ist die Webseite selbsterklärend? Verstehe ich jederzeit, was ich tun muss/kann?
- Ist die Handhabung einfach? Anzahl Schritte bis zur Lösung minimal?
- Reagiert die Webseite jeweils, wie ich es erwarte?
- Muss ich gewisse Informationen mehrmals angeben, bevor die Webseite reagiert?
- Sind die Abläufe konsistent? (= logisch und immer gleich)
- Kann ich Aktionen rückgängig machen/korrigieren? Kann ich jederzeit abbrechen, wenn ich möchte?
- Sind die Reaktionszeiten der Webseite tolerabel?
- Gefällt mir das Design der Webseite? Verbesserungsvorschläge?
- Ist mir irgendein Fehler aufgefallen? Hat etwas nicht (auf Anhieb) funktioniert?
- Hat mir das Dialekte-Raten Spass gemacht?
- Sonstiges:

7.4 Projekterweiterung

7.4.1 Zweck

Nachfolgend werden die User Storys beschrieben, die beim Brainstorming der Projektarbeit zum Dialekte-Raten zusammengekommen sind und die die Grundlage für weitere Projekte bilden könnten.

7.4.2 Personae

Da es sich beim Dialekte-Raten um einen sprachlichen Vorgang handelt, erschien es uns sinnvoll, die Nutzer nach sprachlichen Kriterien zu unterscheiden. Je nachdem, wie gut ein Nutzer mit der schweizerdeutschen Sprache vertraut ist, sollte er durch andere Aufgabenstellungen und Anreize angesprochen werden.

Personen mit Muttersprache Schweizerdeutsch Bei dieser Nutzergruppe kann davon ausgegangen werden, dass sie mit Schweizerdeutsch als Sprache sowie mit den verschiedenen schweizerdeutschen Dialekten hinreichend vertraut ist. Sofern die Person in der Schweiz aufgewachsen ist, ist sie vielleicht auch in der Lage, die feinen Unterschiede zumindest eines Kantondialekts auszumachen. Da die Schweizer Dialektsammlung immer noch an neuen Daten von Muttersprachlern interessiert ist, ist diese Gruppe besonders interessant für das gesamte Projekt.

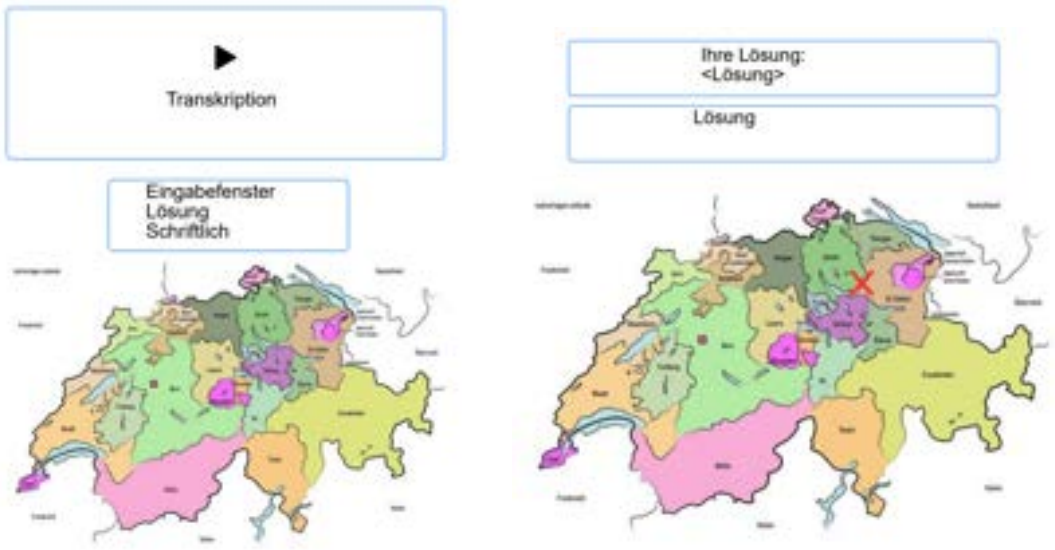
Personen mit anderer Muttersprache als Schweizerdeutsch Diese Nutzergruppe bringt weniger und vornehmlich passive Vorkenntnisse der Schweizer Dialekte mit. Im Umfeld der Dialektsammlung kann aber von einem Interesse an den Dialekten bzw. am Kennenlernen derselben ausgegangen werden.

7.4.3 User Story 1: Raten auf Kantonsebene

User Story Als Nutzer mit anderer Muttersprache als Schweizerdeutsch möchte ich den Ursprung eines schweizerdeutschen Sprechers auf Kantonsebene bestimmen können, um besseren Smalltalk mit schweizerdeutschen Sprechern betreiben zu können.

Tasks

- Ich kann ein Dialekt-Sample abspielen lassen.
- Ich kann den vermuteten Herkunftskanton angeben.
- Nach der Eingabe wird mir der korrekte Kanton angezeigt.



(a) Ansicht vor der Eingabe **(b)** Ansicht nach der Eingabe

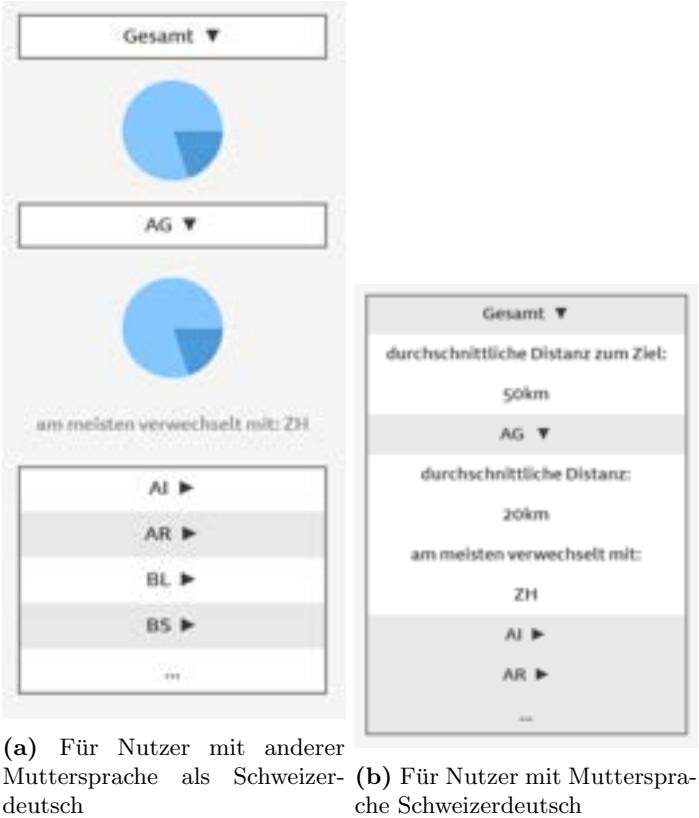
Abbildung 18: UI Sketches für User Story 1.

7.4.4 User Story 2: Selbstvergleich

User Story Als Nutzer möchte ich wissen, wie gut ich die Herkunft eines schweizerdeutschen Sprechers errate, damit ich mich im Raten verbessern kann.

Tasks

- Es wird eine Statistik angezeigt.
- Ich kann sehen, welche Kantone ich besonders gut errate.
- Ich kann sehen, wie ich mich über die Zeit verbessert habe.
- Nutzer mit Muttersprache Schweizerdeutsch: Ich kann sehen, wie weit (geographisch) ich im Schnitt danebenrate.



(a) Für Nutzer mit anderer Muttersprache als Schweizerdeutsch

(b) Für Nutzer mit Muttersprache Schweizerdeutsch

Abbildung 19: UI Sketches für User Story 2.

7.4.5 User Story 3: Vergleich mit anderen

User Story Als Nutzer möchte ich wissen, wie gut ich im Dialekteraten im Vergleich zu anderen abschneide, damit ich meine Performance einschätzen kann.

Tasks

- Jedes bereits geratene Sample kann ich in einer Übersicht aufrufen.
- Für jedes Sample wird angegeben, wie ich und andere Nutzer geraten haben.
  - Angabe der durchschnittlichen Distanz (von Schweizer Muttersprachlern)
  - Angabe der x meistgeratenen Kantone in Prozent (von Nicht-Muttersprachlern)

Abbildung 20: UI Sketch für User Story 3.

7.4.6 User Story 4: Wettkampf

User Story Als Nutzer möchte ich mich mit anderen Nutzern messen können, damit meine Anstrengungen im Dialekte-Lernen sich auszahlen.

Tasks

- Es wird eine Rangliste angezeigt.
- Es wird der Score jedes Nutzers angezeigt.

Abbildung 21: UI Sketch für User Story 4.

7.4.7 User Story 5: Vergleich der Dialekte

User Story Als Nutzer möchte ich verschiedene Schweizer Dialekte miteinander vergleichen können, damit ich sie besser unterscheiden lerne.

Tasks

- Im Vorgang wurde ein Sample falsch geraten.
- Ich kann das originale Dialekt-Samples aus der Umgebung abspielen.
- Als Nutzer mit Muttersprache Schweizerdeutsch: Ich kann andere Samples aus der Umgebung des falsch geratenen Ortes abspielen.
- Als Nutzer mit anderer Muttersprache: Ich kann das Sample im falsch geratenen Dialekt (Kanton) abspielen.

(a) Für Schweizer Muttersprachler **(b)** Für Nicht-Muttersprachler

Abbildung 22: UI Sketches für User Story 5.