ASSISTED INTERVIEW TRANSCRIPTION FOR QDAcity

MASTER THESIS

HUGO IBANEZ ULLOA

Submitted on 2 November 2021

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

Supervisor:
Dr. Andreas Kaufmann
Prof. Dr. Dirk Riehle, M.B.A.

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Nuremberg, 2 November 2021

License

Nuremberg, 2 November 2021

Abstract

Qualitative research deals with a wide array of unstructured input data. One common technique for data gathering is performing interviews which are recorded and subsequently transcribed for analysis. While QDAcity, a cloud-based solution for qualitative data analysis (QDA), already supported the analysis stage of this process, the transcription had to be performed either manually or with an external tool or service.

Transcribing research data already involves crucial decisions, like deciding what and how to transcribe (i.e., filler words, pauses). These decisions affect the following analysis, and consequently, the results. Therefore the researcher is often well-advised to either transcribe the interview themselves or at least carefully correct the transcription.

In this thesis, we present an extension of the current capabilities of QDAcity focused on the automation and correction of transcription content based on interview media.

In our approach, first, an automated transcription from cloud-based services is obtained. This transcription output can be corrected with a user interface that aids this task. Subsequently, as the last step, it can be exported into the data format needed for further domain analysis.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

API	Application Programming Interface
ASR	Automatic Speech Recognition
CRUD	Create, Read, Update, and Delete
FLAC	Free Lossless Audio Codec
GCP	Google Cloud Platform
GAE	Google App Engine
GCS	Google Cloud Storage
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MP3	MPEG-1 Audio Layer III
QDA	Qualitative Data Analysis
SLA	Service Level Agreement
STT	Speech To Text
UI	User Interface
UML	Unified Modeling Language

URL Uniform Resource Locator

WAV Windows Audio Waveform

1 Introduction

Qualitative data analysis (QDA) methods focus on extracting the relevant information from qualitative data, interpreting it, and abstracting it. These methods consist of different steps. One of them is the transcription process.

The transcription process is a necessary process involving decision-making that can impact the outcome of the research. Atkinson et al. (1984) stressed that the production and use of transcripts are "research activities" and should not be approached as merely a "technical detail" that precedes analysis. The decision on how to proceed already impacts the analysis. It is a critical stage yet time-consuming. It is recommended that the researcher does it.

Currently, the transcription process is a very time-consuming task. What to include in the transcription should always be driven by the research question, and different ways to approach this problem exist. Researchers need more support to make this process more efficient.

QDAcity, a web application for qualitative data analysis, integrates multiple different types of input data and handles the analysis of interviews that already have been transcribed. The prevalence of interview transcription as a data source for qualitative research has been extensively discussed in the literature (McLellan et al., 2003). In this thesis, a module to automate part of the transcription process and aid the researcher to produce an analyzable transcription is developed.

Chapter 1 of this thesis presents an introduction to the topic, followed by the definition of the software requirements in chapter 2. The chapter 3 handles the architecture and design of the developed component, and chapter 4 examines the implementation. As a final part in chapter 5, the evaluation of the developed tool draws upon the previous requirements and then in chapter 6, the conclusion of this thesis.

1.1 QDAcity

Computer-assisted QDA software exists in a wide arrange of options such as Atlast/ti, The Ethnograph, QSR N4, QSR N5, to name a few. The Professorship for Open Source Software at the Friedrich-Alexander-Universität Erlangen Nürnberg developed QDAcity, a cloud-based QDA software.

QDAcity runs powered by the Google Cloud Platform (GCP), and it has a client-server architecture that communicates through Hypertext Transfer Protocol (HTTP) requests and WebSocket messages. Researchers can use QDAcity to perform qualitative data analysis. QDAcity provides useful features like real-time collaboration, a teaching platform and supports different data sources. The extension proposed in this thesis aims to expand the current capabilities and support the researcher during the transcription process.

A QDAcity project contains textdocuments which need to be analyzed. For analysis of these textdocuments, a code system is used, consisting of the relevant codes of the project's domain. The code system can be defined and redefined by the users of the project. A number of users act as raters, which means that they individually apply the codes to specific text segments of the documents in a project. This process is called coding. In the end of a project, users can write a theory about the coded textdocuments, which is the result of the theory building research (Schöpe, 2017).

1.2 The Role of Transcription in Qualitative Research and Data Analysis

According to MacQueen and Milstein (1999), inappropriate or inadequate data preparation decisions can delay or negatively affect the analysis process. This sensitive step has challenges. Speech elisions (the omission of a sound between two words, usually a vowel and the end of one word or the beginning of the next), incomplete sentences, overlapping speech, a lack of clear-cut endings in speech, poor audiotape quality, and background noises are just a few of the issues that a transcriber encounters. In addition, he or she must carefully determine where and when punctuation is required so as not to change the intent or emphasis of an interviewee's response or comment (MacQueen & Milstein, 1999).

The decision on what and how to transcribe may significantly affect and impact the analysis; several decisions are involved, like the translation of content fillers, semantics and even the decision to transcribe parts of the content. It is an important part, and ideally, the researcher should do it instead of outsourcing this step.

1.3 Automatic Speech Recognition

Automatic Speech Recognition (ASR) has a long history of being one of the complex problems in artificial intelligence and computer science. Since speech is the primary means of communication between people, the field has attracted a great deal of attention over the last decades. Since the 1930s, when Bell Laboratories proposed a system model for speech analysis and synthesis (Juang & Rabiner, 2005), small yet steady improvements have characterized the field.

The early versions of automatic speech recognition systems were able to identify isolated words from small vocabulary datasets, moving forward to the identification of continuous speech by using pattern recognition. The recognition of continuous speech progressed to the identification of spoken dialogue and the adoption of machine learning models (Juang & Rabiner, 2005).

Today, speech technologies are commercially available for a wide arrange of use cases. State-of-the-art accuracy and the most advanced deep learning neural networks algorithms for ASR can transcribe content with accurate captions providing a flexible deployment and integration. These speech technologies building voice-powered applications can be open source and provide the code available to download and use or commercial speech recognition systems with inaccessible code and mechanisms.

To provide a practical example of the progress in the field, the Google API (Application Programming Interface) is integrated into a broad range of Google products like YouTube transcription, Translate and voice search. Google achieved in 2015 an 8% error rate, which is a reduction of more than 23% from the year 2013 (Këpuska & Bohouta, 2017). In recent years, Google has acquired several deep learning companies over the years, including DeepMind, DNNresearch and Jetpac.

2 Requirements

The main goal of the thesis is to integrate a speech-to-text API into QDAcity and provide a user interface for correction of it. This integration shall enable users to obtain an automated transcription based on interview media. Given that the automated transcription will most likely contain errors, it is imperative to provide a user-friendly interface to correct and approve the transcription. The detailed requirements have been divided into functional and non-functional requirements.

2.1 Functional Requirements

The following functional requirements are expressed using the FunctionalMASTeR template by Rupp et al. (2014). As shown below in figure 2.1. In this illustration, the words in italic are used as a placeholder, and the dashed boxes are optional.

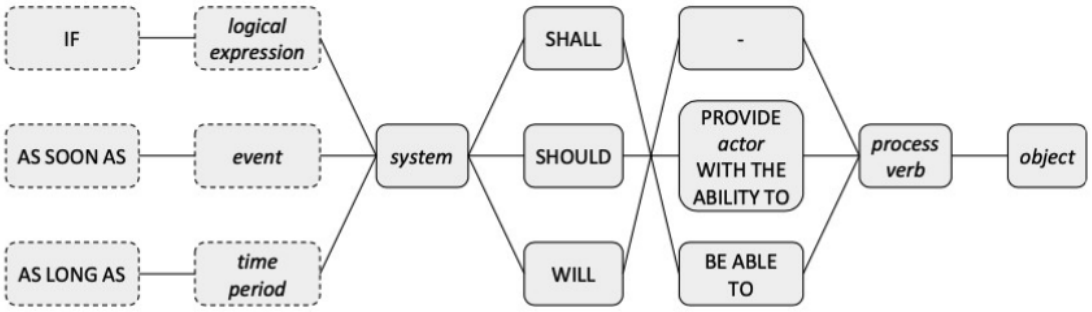

Figure 2.1: FunctionalMASTeR template based on Rupp et al. (2014)

The semantic definition of the keywords words *shall, should* and *will* can be found in table 2.1. The used terms are semantically defined in table 2.2.

The requirements are divided into different sections based on their relationship with these concepts. Accordingly, the requirements are divided into the following sections:

Keyword	Semantic Definition of the Keyword
Shall	A requirement that has to be fulfilled for the project to succeed.
Should	A requirement that is important but not necessary for the software to work correctly.
Will	A requirement that is not necessary but desired.

Table 2.1: Definition of the requirement keywords

Term	Semantic Definition of the Term
User	In QDAcity, a *user* shall be defined as a physical person that is using the QDAcity application.
Document	In QDAcity, a *document* shall be defined as a data structure that has the potential to be used for qualitative data analysis.
Codeable format	In QDAcity, a *codeable format* shall be defined as a type of document that allows for parts of the data to be associated with a code within the codesystem.
Code	In QDAcity, *code* is a strategy of QDA in which labels are assigned to parts of the examined data.
Codesystem	In QDAcity, a *codesystem* shall be defined as a a hierarchical set of codes.
Assisted transcription component	*Assisted transcription component* shall be defined as the module developed in this thesis.
Transcription document	In QDAcity, a *transcription document* shall be defined as a type of document that is not ready to be coded and its format is still an STT (Speech-To-Text) format.
STT format	In QDAcity, a *STT Format* shall be defined as a format that holds the transcription and transcription metadata.

Table 2.2: Definition of the requirement terms

- Section 2.1.1 (Speech to Text Process) defines requirements related to the first step of the process, to upload the media object to Google Cloud Buckets and transcribe it.
- Section 2.1.2 (Review, Correction and Approval of Automated Transcrip-

tion) defines requirements related to reviewing and correcting the content from the Google API. Given that the transcription process is a sensitive pre-processing step of the data analysis, the researcher will have complete control over this process with the aid of tools to allow this process to be easier and faster.

- Section 2.1.3 (Storage of Changes) defines requirements related to data storage. The revision and correction of transcription can be resumed in different sessions, allowing the researcher to store the changes along the transcription process.
- Section 2.1.4 (Export of the Corrected Transcription) defines requirements related to the last step of the process. After the transcription satisfies the user, it can be parsed and exported for further analysis.
- Section 2.1.5 (Authentication and Authorization) defines requirements related to the process of verifying who a user is and what elements should be allowed to access.

2.1.1 Speech to Text Process

The outcome of the transcription process using an API will be a transcription text based on interview media. For this process, the following requirements have been defined:

FR-1: The assisted transcription component shall allow the user to upload and transcribe single-channel (mono) audio files in wav format up to a maximum file size of 500 MB.

FR-2: The assisted transcription component shall allow the user to upload and transcribe single-channel (mono) audio files in mp3 format up to a maximum file size of 500 MB.

FR-3: The assisted transcription component shall allow the user to upload and transcribe single-channel (mono) audio files in flac format up to a maximum file size of 500 MB.

FR-4: A selected and visually highlighted position in the transcript shall be synchronized with the corresponding temporal position in the audio file.

FR-5: The assisted transcription component shall provide the ability to transcribe audio in German or English.

FR-6: The assisted transcription component shall provide the ability to visualize the speaker diarization of the automated transcription.

FR-7: In the event of any error during the transcription process, the assisted transcription editor shall delete the media object from the cloud storage and provide user feedback.

FR-8: The assisted transcription component should provide the ability to transcribe audio in other languages supported by the API beyond the scope of FR-5.

FR-9: The assisted transcription component should provide the user with the ability to visualize the accuracy of the confidence value of the algorithm on the accuracy of the automatically transcribed words.

FR-10: QDAcity document list viewer should provide the user with the ability to differentiate visually when a transcription document is still not ready to be coded.

FR-11: The assisted transcription component will transcribe video files in mp4 format up to a maximum file size of 500 MB.

FR-12: As long as there are no transcription documents, QDAcity shall not display the transcription editor.

2.1.2 Review, Correction and Approval of Transcription

The users need the ability to correct the transcription provided by the API. These corrections will include particular characteristics. On this basis, these requirements can be defined:

FR-13: The assisted transcription component shall provide the user with the ability to modify the text output of the automated transcription.

FR-14: The assisted transcription component shall provide the user with the ability to customize the names of the speakers.

FR-15: The assisted transcription component shall display the speaker code and timestamp next to the text paragraph.

FR-16: The assisted transcription component shall be integrated within the layout of the current editors of QDAcity.

FR-17: The assisted transcription component shall provide the user with the ability to jump 15 seconds backward in the media.

FR-18: The assisted transcription component shall provide the user with the ability to insert exact the time position for the media player to reproduce the audio.

FR-19: The assisted transcription component should provide the user with the ability to jump to the position of the audio by selecting a position on the text transcription.

FR-20: The assisted transcription component will provide the user with the ability to reproduce the interview media at different speeds.

FR-21: The assisted transcription component will provide the user with the ability to mute the reproduction of the audio.

FR-22: The assisted transcription component will provide the user with the ability to use keyboard shortcuts to interact with the media player.

2.1.3 Storage of Changes

The corrections to the automated transcription will have to be stored. To achieve that, the following requirements are defined:

FR-23: QDAcity shall provide the user with the ability to store changes during the correction of the transcription.

FR-24: As soon as a user modifies the transcription, QDAcity should persist the change.

FR-25: As soon as a user removes a part of the transcription, QDAcity should persist the change.

FR-26 QDAcity shall provide the user with the ability to delete the automated transcription.

FR-27: As soon as a user removes a transcription, QDAcity should persist the changes and delete the interview media.

2.1.4 Export of the Corrected Transcription

Users need to be able to code the automated transcription; therefore, the following requirements have been defined:

FR-28: QDAcity shall provide the user with the ability to export the correct transcription to a codeable format.

FR-29: QDAcity shall provide the user with the ability to export the correct transcription to an editable format.

FR-30: QDAcity should provide the user with a confirmation prompt to avoid accidental export.

FR-31: As soon as the user confirms the export of the transcription, QDAcity shall provide the user with the ability to further code the document.

2.1.5 Authentication and Authorization

On a document level, authentication and authorization are already implemented into QDAcity. Now QDAcity will also be handling media objects. To achieve this, the following requirements have been delimited:

FR-32: The authorization mechanism shall allow the user to upload and interact with interview media only for the members of the concerning QDAcity project.

FR-33: The authentication mechanism shall allow the upload and interaction with interview media to only authenticated and authorized users.

FR-34: The authentication mechanism shall allow upload and interaction with interview media even if the user does not have a Google account.

FR-35: The authentication mechanism should be handled by service account keys.

2.2 Non-functional Requirements

The delimitation of the non-functional requirements is based on the PropertyMASTeR and the EnvironmentMASTeR templates by Rupp et al. (2014) as shown in Figures 2.2 and 2.3.

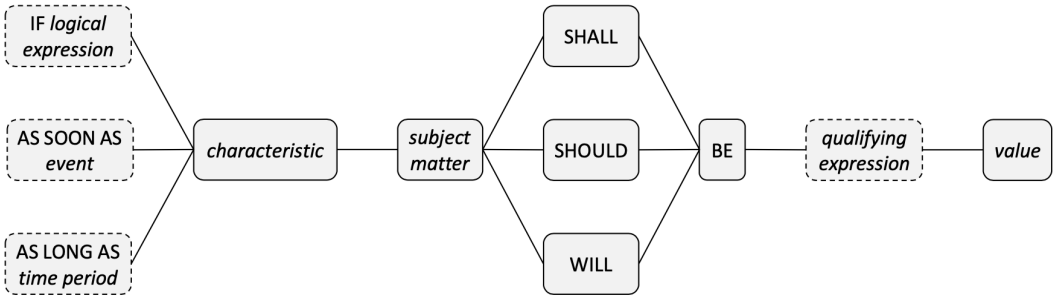

Figure 2.2: PropertyMASTeR template based on Rupp et al. (2014)

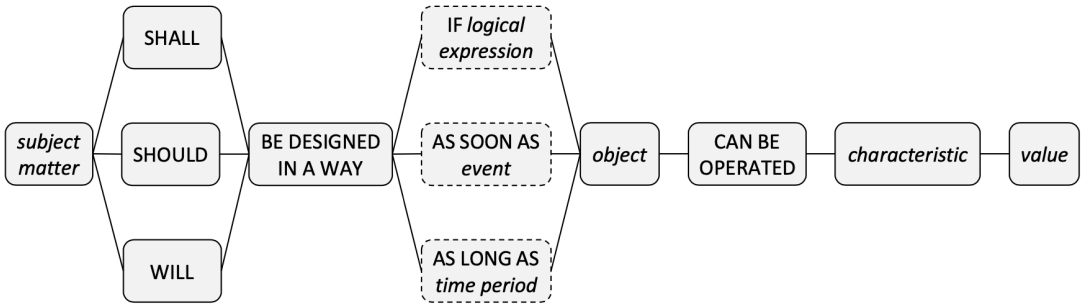

Figure 2.3: EnvironmentMASTeR template based on Rupp et al. (2014)

The non-functional requirements are assembled in the following section called Quality Metrics. To verify that the implementation achieves quality metrics of software, the requirements are grouped following the guidelines of the ISO-25010 Estdale and Georgiadou (2018).

The ISO/IEC 25010: 2011 - Systems and software Quality Requirements and Evaluation (SquaRE)– System and software quality model present the leading quality models for software products and computer systems known as 'software-intensive computer systems' (Estdale & Georgiadou, 2018).

2.2.1 Quality Metrics

According to the standard ISO-25010, the characteristics to which a high-quality software system should adhere are divided into eight categories and their cor-

responding subcategories as expressed in the figure 2.4. The characteristics are described in the following sections.

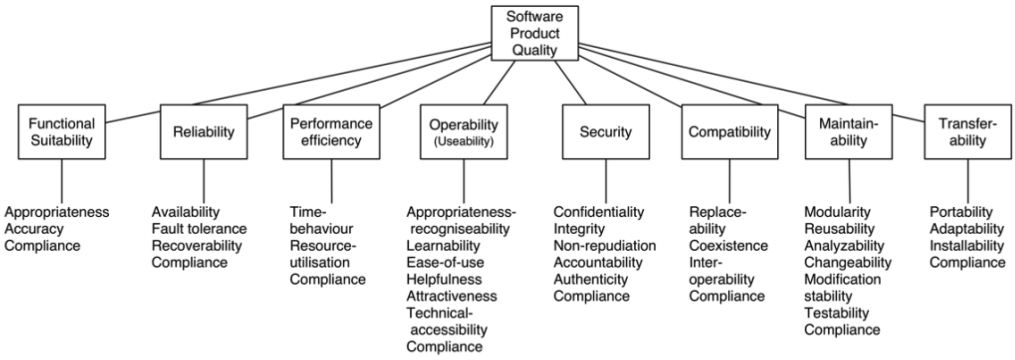

Figure 2.4: ISO/IEC 25010:2011 - Categories

Functional Suitability

This characteristic indicate that the implemented features should be fully functional. The main objectives should be achieved, and the user should be able to perform the designated task.

Reliability

This subsection indicates that QDAcity should be operative under normal conditions all the time. Error handling should be implemented, and the operation of the software should run without interruptions. To fulfill and expand on this characteristic, the following requirements have been defined:

NFR-1: The assisted transcription component should be designed in a way that it can be operated via the current Google Cloud Console.

NFR-2: Transcription services should be able to resume the upload if the connection is not stable.

Performance Efficiency

Performance efficiency is composed of three sub-characteristics. Time behaviour, resource utilization and capacity. To achieve this, in the context of this thesis, QDAcity performance should not be affected, the software should keep working without any drastic increase in the earlier mentioned parameters.

Operability and Usability

In the context of this thesis, this characteristic focuses on following the implemented color schemes, UI guidelines and localization strategies. The user experience of the new component should be aligned with the previously implemented elements. The deliverable should provide a similar quality and ease of use. To accomplish this, the following requirements have been defined:

NFR-3: The assisted transcription component user interface elements that contain text shall comply with the app localization strategies.

NFR-4: The assisted transcription component user interface elements shall be designed in a way that follows the current color scheme for QDAcity.

NFR-5: The assisted transcription component user interface elements shall be designed in a way that complies with the current UI guidelines for QDAcity.

NFR-6: As soon as the upload begins, the assisted transcription component should provide visual feedback to the user.

NFR-7: As soon as the transcription begins, the assisted transcription component should provide visual feedback to the user.

NFR-8: The assisted transcription component will be designed in a way that it can provide real-time collaboration to multiple users.

Security

This characteristic indicates that only authorized and authenticated users should be allowed access to the resources.

Compatibility

Compatibility refers to the degree to which a product, system or component can exchange information with other products, systems or components, and perform its required functions while sharing the same hardware or software environment. In the context of QDAcity, the new features should be compatible with the previously existent elements.

Maintainability

This characteristic addresses the future maintainability of the created components. The proposed way to achieve this is to create tests; therefore, the following requirement has been defined:

NFR-9: All API methods in the new endpoint classes should be covered with unit tests.

Transferability

In the context of this thesis, this characteristic focuses in the ability to use QDAcity in different browsers and operating systems. The assisted transcription module, should be compatible with the previously supported browsers and operating systems.

3 Architecture and Design

Based upon the defined requirements in Chapter 2, this chapter introduces the architecture of the assisted interview transcription component. The first part states the current system design, moving forward to elaborate on some aspects of the implemented architecture in more detail. Furthermore, there is a description of the client-side, composed of an overview of the library and the components hierarchy. As a conclusion of the chapter, a description of the services server-side can be found.

3.1 Existing System

QDAcity has a client-server architecture that communicates through HTTP requests and Web-socket messages. QDAcity runs on GCP.

The backend is mainly responsible for the persistent storage, authentication and authorization; moreover, it provides services of a typical web backend in the form of REST endpoints written in Java.

The frontend is using React, which is a declarative, component-based framework. React allows the creation of complex UIs.

3.2 Selection of Storage Provider and Speech-to-text API

With the focus on increasing maintainability and reducing implementation time, the products of GCP [1] are being used. By using this first choice, the application can also have a plethora of other benefits, such as the simplification of monitoring and administration and easier handling of authentication and authorization processes.

[1] http://cloud.google.com

Using the Google speech-to-text API [2], the app has access to Google's most advanced deep learning neural network algorithms for automatic speech recognition. It supports more than 125 languages and variants and can handle noisy audio from many environments without requiring additional noise cancelation.

The API provides as a beta feature two features that will be implemented: automatic punctuation and speaker diarization.

Regarding the cloud storage provider, Google Cloud Storage (GCS) [3] enables reliable and secure object storage with scalability and reliability. Given that we will be storing interview media that should be available during the correction process, GCS provides safe methods for upload and download of the media. The authentication and authorization to the media will be handled by GCP and our backend.

Both cloud services provide documentation and code examples to kickstart implementations.

3.3 Evaluation of Transcription Correction Library

To fulfill the task of correction of the automated transcription, 3rd party libraries were considered. Implementing a new component from the ground up was also considered, but this option was discarded, given the possibility of reasonable existing solutions.

The criteria presented in table 3.1. was considered when evaluating the use of the libraries. Two main options were selected and evaluated. Both libraries provide a license model compatible with QDAcity, are open source and provide react support.

Considering the last three parameters, the option taken was the library named React Transcription Editor due to the fact that offers more documentation, the UI was better aligned with the current interface, and the health status of the library reflects more activity.

In Table 3.2. a detailed overview of the library is provided. It uses an MIT License, it is free, provides documentation and the dependencies are compatible with previously implemented ones. The main features of the library will enhance the transcription process and provide a good user experience.

[2]https://cloud.google.com/speech-to-text/
[3]https://cloud.google.com/storage

Criteria	React Transcription Editor	Slate Transcript Editor
License model	MIT License	MIT License
Open source	Yes	Yes
React support	Yes	Yes
Documentation	Demo project, code examples, documentation.	Demo project, code examples, documentation.
Ease of use and personalization	Clear and clean UI, does not allow theming outside the box.	User interface allows theming, more complex UI.
Health status of the library	338 stars, 112 forks. Latest release on June 28, 2021. Latest commit on Jun 28, 2021.	34 stars, 17 forks. Never released. Latest commit on Oct 14, 2021.

Table 3.1: Comparison of transcription edition libraries

Criteria	BBC/ React transcript editor V. 1.4.0
Open source. Is the project open source? Which license is being used?	MIT License (MIT)
Costs. Are there any costs for use?	Free
Documentation. Is there enough documentation?	Demo project, code samples. Overview of basic use case and advanced use cases
Main dependencies.	Fortawesome, draft-js, mousetrap
Main features.	Advanced player controls, keyboard shortcuts, interactivity features to support transcript correction, supports speaker diarization.

Table 3.2: React Transcript Editor library characteristics

3.4 Transcription Formats

GCP provides the google-stt format. This format contains different fields depending on the needs of the implementation. The characteristics of the needed output can be seen on table 3.3. The first characteristic, namely speaking diarization assigns values to paragraphs in correspondence to the speakers. Each word comes

from the API with a confidence value that reflects the accuracy of the algorithm towards the selection. As a last parameter, we have the spoke punctuation which is the ability to identify parts of the speech and assign punctuation to them.

Characteristic	**Summary**
Speaker Diarization	The characteristic to automatically detect different speakers.
Word-level confidence	The measured degree of accuracy of the response. It is a number between 0.0 to 1.0.
Spoken punctuation	The ability to detect spoken punctuation.

Table 3.3: Required characteristics of transcription output format

3.5 Supported Languages and Media Formats

For best results [4], the audio source should be captured and transmitted using a lossless encoding (FLAC or LINEAR16). The accuracy of the speech recognition can be reduced if lossy codecs are used to capture or transmit audio, particularly if background noise is present. FLAC (Free Lossless Audio Codec) is the recommended encoding because it is lossless– therefore recognition is not compromised; however it is worth mentioning that FLAC files are comparatively bigger than for example an MP3 file. The supported formats by the app are: FLAC, WAV and MP3.

Regarding the supported languages [5] , the chosen API service currently supports the ones in table 3.4. There is indeed support for more languages; however at the moment, Google doesn't provide diarization for them therefore the assisted transcription component will be limited to those that fulfill all the requirements.

Supported Languages
Chinese, Mandarin (Simplified, China)
English (India)
English (Singapore)
English (United Kingdom)
English (United Kingdom)
English (United States)
French (France)
German (Germany)
Italian (Italy)
Japanese (Japan)
Portuguese (Brazil)
Russian (Russia)
Spanish (Spain)

Table 3.4: Supported Languages

[4]https://cloud.google.com/speech-to-text/docs/best-practices
[5]https://cloud.google.com/speech-to-text/docs/languages

3.6 System Architecture

As a result of the previous decisions, a high level illustration of the system architecture can be drawn. Figure 3.1 illustrates the position of the new module in regards to the current QDAcity application.

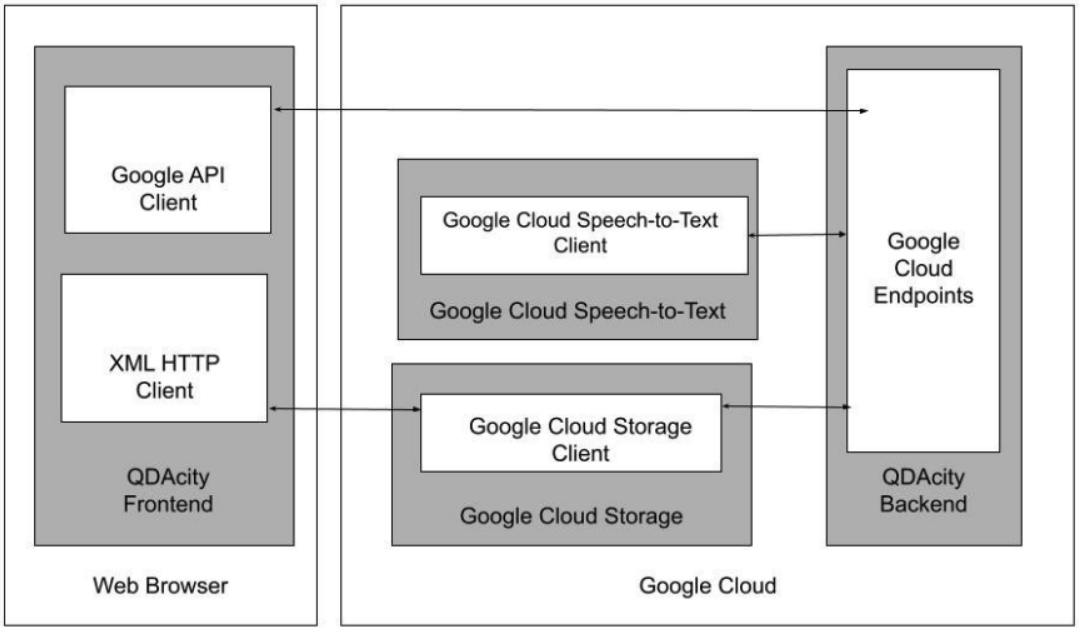

Figure 3.1: High level system architecture

In figure 3.2, a Unified Modeling Language (UML) component diagram can be found. At first glance, the four main components can be observed. The QDAcity frontend and backend and two Google services. The frontend with its HTTP methods is divided into two sub-components, the Google API and an XML HTTP client.

The QDAcity backend provides 4 endpoints:

- `generateUploadSignedURL` creates a signed URL for the upload of the media object. The upload will then be performed by the XML HTTP Client directly.
- `getTranscriptionDocument` starts the transcription process. Receives the object media name stored in GCS, stablishes connection with the Google Cloud Speech-to-text component and receives the JSON transcription object.
- `deleteInterviewMedia` retrieves the information of the object and communicates with GCS directly to delete the interview media.

- `getSignedURL` generates a signed URL for interview media retrieval during the correction process.

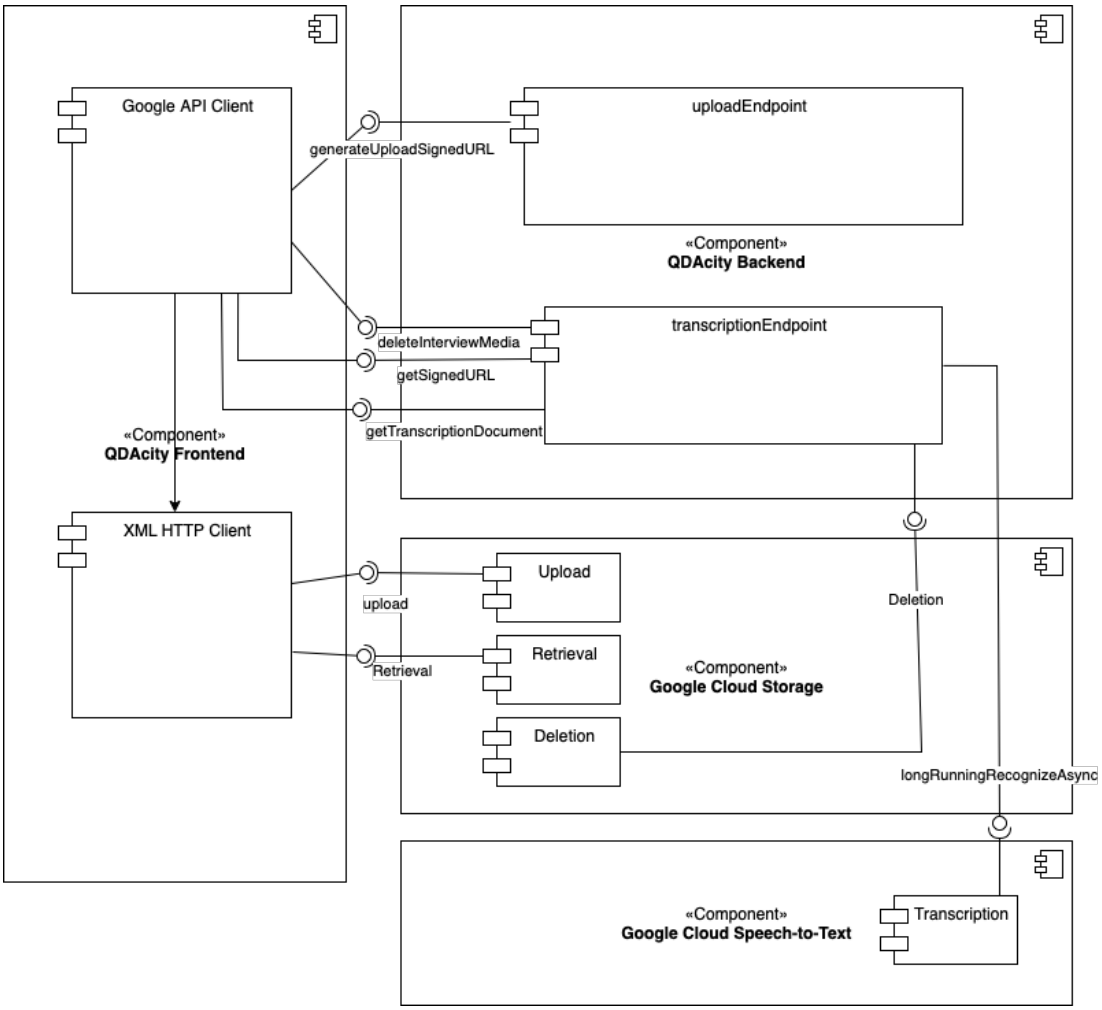

Figure 3.2: Component architecture

4 Implementation

This chapter describes the most important details about the components of the assisted interview transcription. It starts with an overview of the implementation of the primary use cases. This overview encompasses the media upload, the creation of the transcription document, followed by the storage of changes, deletion and export. Furthermore, there is a description of the server-side which is composed by the CRUD operations, signed URLs generation and interactions with the transcription API.

4.1 Overview of Use Cases

This section describes the behavioral view of the assisted interview transcription component.

4.1.1 Media Upload

In order to comply with the security requirements delimited on the Authentication and Authorization section of the functional requirements (Cf. FR-32 to FR-36 in section 2.1.5), the usage of signed URLs was implemented. This upload method allows the system to perform a seamless upload of objects to GCP without the need for the user to have a Google account. The frontend is in charge of uploading directly to the bucket, and the backend is used only to re-retrieve an authorized upload signed URL.

At first, the alternative of going through the backend was considered but discarded after some performance tests. A timeout was reached while uploading bigger files due to limits imposed by the app's infrastructure.

As depicted in steps 1, 2 and 3, the frontend prepares a request for a signature. This GET call retrieves an authorized URL from the frontend to proceed with the upload. Steps 4 to 7 are being done from the frontend directly to GCS. The first one is a POST request, and the following is a PUT request sending the object to upload. In figure 4.1, a UML component diagram can be found. At first glance,

the four main components can be observed. The QDAcity frontend and backend and two Google services. The frontend with its HTTP methods is divided into two sub-components, the Google API and an XML HTTP client.

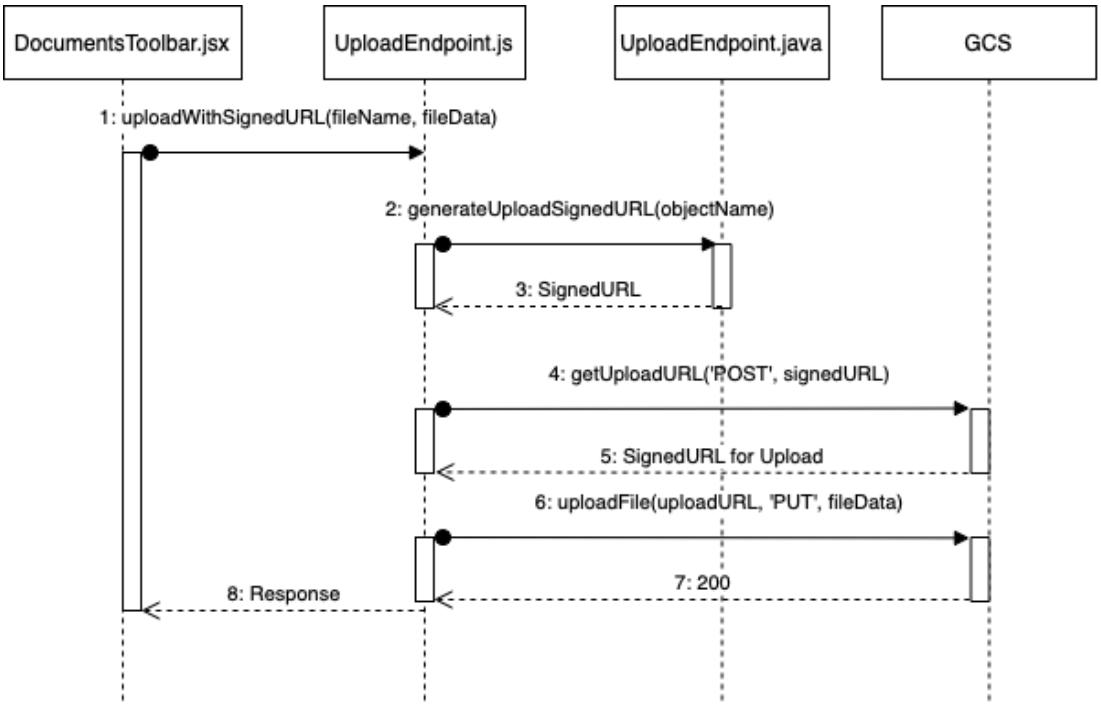

Figure 4.1: Upload sequence diagram

4.1.2 Creation of the Transcription Document

As soon as the upload is performed, a request to the endpoint `getTranscriptionDocument` is performed. This endpoint is in charge of the communication with the Google speech-to-text service. The implementation is written in Java using the provided Google libraries and packages. The endpoint contains some mandatory parameters as well as parameters to improve the output of the transcription.

The class `TranscriptionDocument` is an extension of BaseDocument as can be seen in Figure 4.2. The extension expands two new fields: `gcsUri` (String) that stores the path where the media file will be and `sttType` (String), which stores the format of the current transcription. The rest of the fields that comprise BaseDocument are reused in the same way as other documents inside the app.

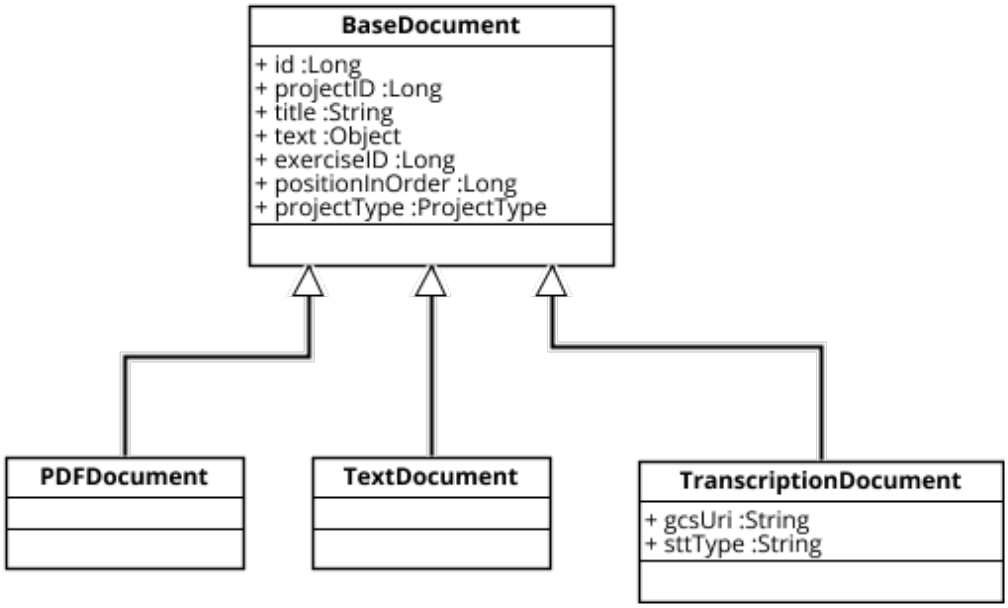

Figure 4.2: Transcription Document Class Diagram

4.1.3 Storage of Changes and Deletion of the Transcription Document

Building upon the previous structure, the already existing endpoints handle the changes within the documents. Since TranscriptionDocument is an extension of the previous document objects, these methods were found to be highly reusable and are in place.

Regarding the deletion of interview media, the process starts with deletion request of the transcription document on the frontend, an endpoint handles the removal of the object in Google cloud storage and the data stored in Google datastore. Within the class `TranscriptionEndpoint`, the endpoint `deleteInterviewMedia` retrieves the object's name to be deleted and proceeds with the interaction with Google Storage Services.

4.1.4 Export

An export of the transcription can be triggered by the user at any point. This export converts the transcription information into a regular QDAcity document. This process can be seen in detail in figure 4.3. After the parsing in step one, a removal of the previous `transcriptionDocument` happens and a new `TextDocument` is created in steps 4 to 13. Since this is an irreversible process,

a prompt is being shown to the user so he can double check if the action should proceed.

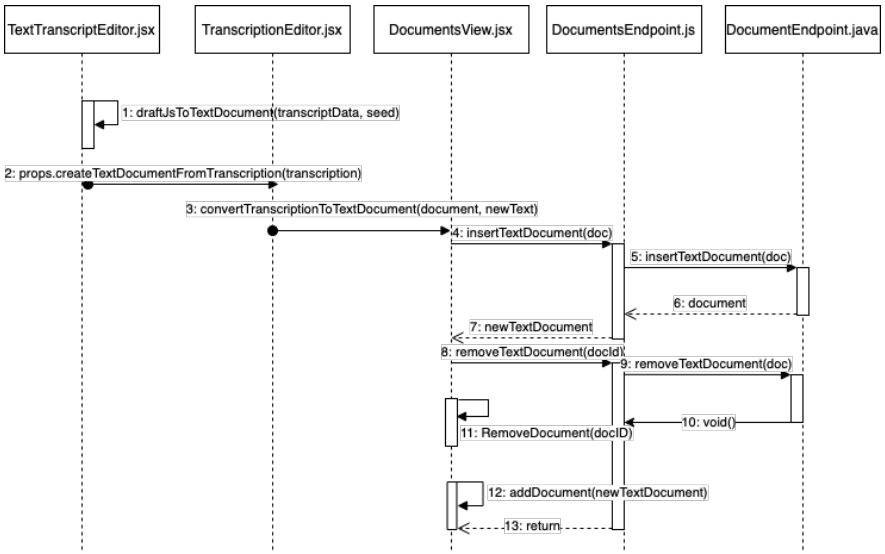

Figure 4.3: Export of Corrected Transcription - Sequence Diagram

4.2 Transcription Services Client

4.2.1 Components Hierarchy

The entry point to the transcription editor is the component called `CodingEditor`; inside this previously existing component, the new implementation begins. In this way, there is an integration of the new editor to the previously implemented editors.

Figure 4.4. illustrates the main components of the assisted transcription frontend. The component `TextTranscriptEditor` aggregates the sub-elements and handles exporting and storage methods. It is the core of the component. The media player is embedded into the Header and provides the media player UI and interaction with the media object. The `VideoPlayer` is a hidden component that deals with the reproduction of the media, and the `TimedTextEditor` component houses the slate components of the text editor.

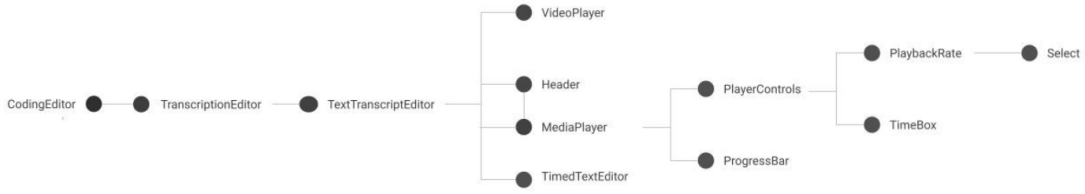

Figure 4.4: Implemented React components hierarchy

4.2.2 User Interface Changes

As it can be observed in figure 4.5. The `TranscriptionEditor` component is embedded and very well integrated into the place of previously implemented editors, blending in and providing a seamless interaction with the rest of QDAcity functionalities.

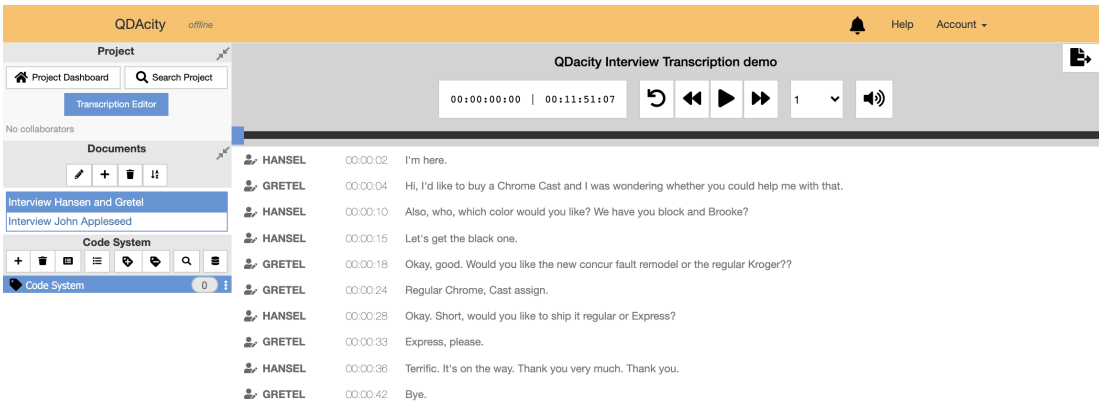

Figure 4.5: Assisted interview transcription integration into QDAcity

Following on figure 4.6, we can appreciate an example of how user interface elements have been adapted to comply with the color and user interface guidelines of QDAcity. An adaptation to `styled-components` [1] was also in place to comply with the user interface guidelines on a visual and implementation level.

[1] https://styled-components.com

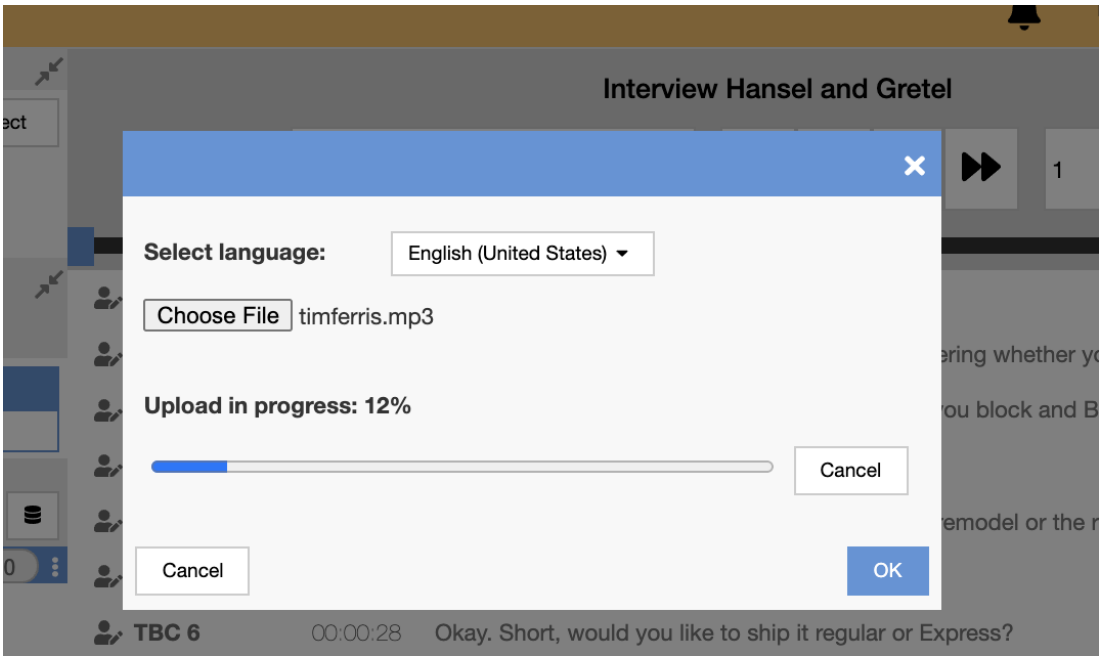

Figure 4.6: Assisted interview transcription upload dialog

4.3 Transcription Services Server

The server-side component is built upon the previous structure and architecture with a few expansions mainly to support the new `TranscriptionDocument` and to hold the authorization towards Google Cloud Buckets and generate the Signed URLs.

4.3.1 CRUD operations

The previous structure for CRUD operations is being re utilized with some extensions to interact with the added fields of the class `TranscriptionDocument`. As expressed in figure 4.2, since the transcription document is an inherited class from `BaseDocument`, the previous methods are simply reused.

4.3.2 Media Upload and Connection with Speech-to-Text API

The upload is handled by the frontend as explained better in the section 4.1.1; however the backend generates the signed URL for the upload since the credentials for authentication are stored on the server-side.

Once the upload is performed, the endpoint `getTranscriptionDocument` inside the `transcriptionEndpoint` proceeds with the connection to the API,

gathers all the necessary parameters and retrieves the transcription.

The Google STT Format received from the Google Speech-to-text API can be found on Figure 4.7. It provides different alternatives, the last one being the one with a higher confidence value and the one used inside the transcription editor. It provides a confidence value per sentence and metadata by word.

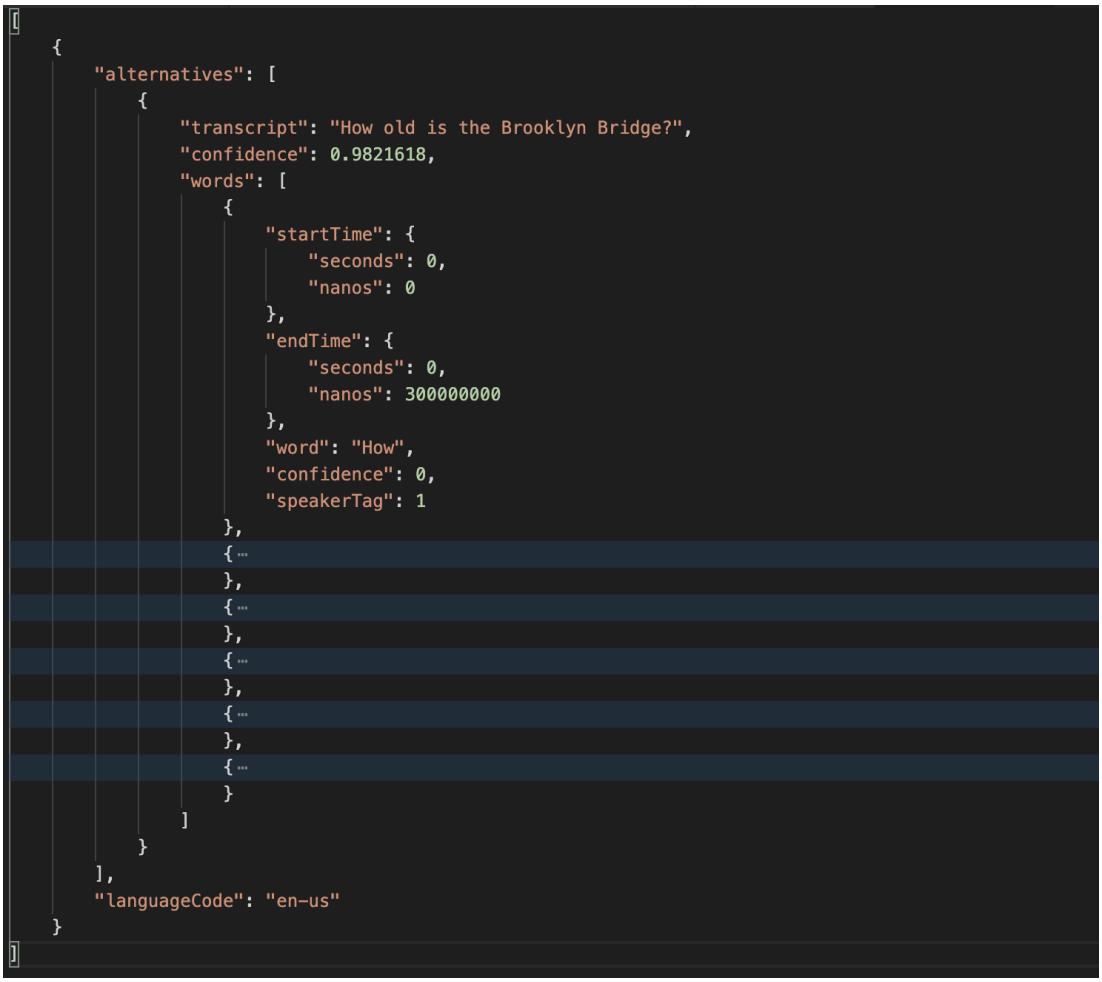

Figure 4.7: Example of the Google STT Format

5 Evaluation

In this section, a review of the requirements from Chapter 2 is performed. Building on this review a comparison of the requirements and the implementation described in chapter 4 takes place. The evaluation is grouped into functional requirements and non-functional requirements. The main goal of this chapter is to provide an assessment of their fulfillment.

5.1 Functional Requirements

This section evaluates the functional requirements. These requirements focus on the speech to text process, modification, storage, export and authorization processes.

5.1.1 Speech to Text Process

The requirements regarding the files' upload, formats, and size FR-1 to FR-3 have been successfully accomplished. The user can now upload interview media with the formats stated in the requirements and get the transcription text. FR-4 highlights a connection between the text editor and the media player and has also been accomplished. The editor has a visual element that highlights the text in relation to the interview media.

Requirements FR-5 and FR-8 have also been implemented. The user can transcribe interview media in German and English. Support for Mandarin Chinese, French, Italian, Japanese, Portuguese, Russian and Spanish is also available.

The transcription API supports diarization for the previously mentioned languages; therefore, FR-6 was also completed. The output from the API provides the diarization, and the transcription text editor showcases clearly the different paragraphs and speakers.

User feedback in the case of an error with the transcription as stated in FR-7 is also in place, prompting an alert in the frontend with proper error handling in the backend.

In regards to word-level confidence values as expressed in FR-9, unfortunately, this requirement has not been achieved. The word-level confidence is stored in the transcription metadata, but the user interface does not interact with these values. These requirements are not prerequisites to any other requirement, and they are not necessary for the main tasks of the tool.

Requirement FR-10 has been achieved; the current document list viewer makes a visual differentiation for transcription documents.

The requirement FR-11 has not been achieved; support for transcription of videos is not implemented due to limitations with the speech-to-text API.

As the last requirement in this section, we have FR-12 that hides the transcription editor and the transcription editor selector button. This requirement was achieved successfully. If there are no transcription documents in the permanent storage, the transcription editor and its components remain hidden.

5.1.2 Review, Correction and Approval of Transcription

The correction of the API output is a critical component of the deliverable. It has to provide features that enhance and facilitate the process.

Requirements FR-13 to FR-14 have been successfully developed. The user can assign personalized names to the speakers. Every section of the interview is editable, text can be added or removed and new paragraphs can be created.

Concerning the integration within the layout and the information contained in the text editor, requirements FR-15 and FR-16 have been successfully implemented. The speaker code, timestamp and speech text are organized within the editor and the whole editor is well integrated into the UI of QDAcity.

Requirements that concern the interaction with media and the behavioral relationship between the text and the media player FR-17 to FR-22 have all successfully been achieved. Actions that facilitate the navigation within the media object like keyboard shortcuts, media playback at different speeds, or having a one-click jump backwards are well integrated and in place. The synchronization between the media player and the text on the text editor works, and the user can select a word in the transcript to jump to that moment in the media player.

5.1.3 Storage of Changes

The storage of changes during the editing allows the user to perform corrections in different correction sessions. The whole transcript is a work in progress until the user decides to export it. These corrections are being stored immediately to the permanent storage and retrieved on demand.

Requirements FR-23 to FR-25 have been successfully implemented. The changes during the transcription correction process are stored onto the permanent storage.

The requirements FR-26 and FR-27 were also successfully developed. The user can delete the whole transcription document. This deletion of the transcription document will also delete the interview media from the Google Cloud Bucket.

5.1.4 Export of the Corrected Transcription

When the user decides, an export of the transcription can take place. This export parses the transcription from a transcription format to a HyperText Markup Language (HTML) for further analysis.

Requirements FR-28 and FR-29 have been implemented successfully. The user can export the transcription to a codeable, editable format.

As stated in FR-30, since this is an irreversible step, the user will be prompted for confirmation. This requirement was also accomplished.

Requirement FR-31 concerning the flow between the transcription editor to the coding editor was also achieved successfully. As soon as the user exports the transcription, the coding editor is shown and the transcription editor is hidden.

5.1.5 Authentication and Authorization

Requirements FR-32 and FR-33 have been successfully achieved. The upload and download of interview media are allowed to only the authorized user; furthermore the media is only available to users that belong to the project assigned to the document.

Requirement FR-34 has also been successfully achieved. With the implementation of signed URLs, the user can upload and download objects from Google Cloud Storage even if the user does not hold a Google account. The current implementation allows the same level of security to both Google and non Google users.

The last functional requirement, FR-35 was also achieved successfully. The authentication is handled by the backend. A proper authentication mechanism towards Google Cloud Buckets was implemented, and the keys remain safe on the server-side.

5.2 Non-functional Requirements

The evaluation of non-functional requirements is performed in this section. The implementation of the features which were developed during this thesis is meas-

ured against the eight quality metrics introduced in section 2.1.1.

5.2.1 Quality Metrics

Functional Suitability

This criteria has been met. Manual tests were performed on QDAcity and everything works as expected. Furthermore, unit tests are in place and reporting no issues.

Reliability

Given that QDAcity runs inside Google App Engine, a metric to use and validate the compliance with this requirement is the App Engine Service Level Agreement (SLA). Google provides a montly uptime percentage of 99,95% (Google, 2020).

Furthermore, the requirement NFR-1 has been succesfully achieved. Given that the transcription module relies on Google, all its components can be operated within the scope of the Google Cloud Console.

Requirement NFR-2 has unfortunately not been achieved due to time constraints; however, performance tests have been meet, and files upload has not reported any issues.

Performance Efficiency

The three sub-characteristics of this section, time behaviour, resource utilization and capacity, have not been modified by the improvements provided in this thesis. The usage of signed URLs to upload and download objects situates the heavy tasks of the transcription process in the user browser.

Operability and Usability

Requirements NFR-3, NFR-4 and NFR-5 have been succesfully achieved. The components presented in this thesis comply with previous localization strategies and color and UI guidelines.

Requirements NFR-6 and NFR-7 have also been achieved. The user receives feedback during upload and transcription.

The last non-functional requirement concerning the real time correction collaboration, NFR-8, has not been achieved.

Security

The usage of service account keys and proper validation methods inside the endpoints ensures that the security characteristics are achieved. The authorization and authentication checks for all new API methods confirm that this criteria is met.

Compatibility

All the components presented in this thesis are compatible with previously existing components within QDAcity. The successful implementation of them can attest to their compatibility.

Maintainability

Requirement NFR-9 has been achieved, the new endpoints are covered with unit tests and the good test coverage ensures that any breaking changes will be potentially identified.

Transferability

The new features have been tested on different operating systems (OSX and Windows) and different browsers (Google Chrome, Firefox, Safari).

6 Conclusion

The transcription process is a vital but time consuming aspect of qualitative research, this process helps the researcher make sense and understand the interviewees. It very much influences the whole analysis process. By augmenting the capabilities of QDAcity to support the transcription process, the researcher can benefit from seamless integration into our QDA software tool, allowing a more efficient and smoother process from data gathering to analysis and expanding the boundaries of computer-assisted QDA software.

The functional and non-functional requirements lay the ground of the proposed architecture and further implementation. According to the previously detailed evaluation, out of 35 functional requirements, 33 have been successfully achieved. All the requirements with the keyword shall and should, required for the project to succeed and not necessary but important have been achieved. The requirement FR-9 concerning the visualization of the confidence value on the transcription editor has been partially accomplished, and the requirement FR-11 has not been accomplished given the complexity of transcribing video files. Out of nine non-functional requirements, seven have been successfully achieved. Requirement NFR-2 in the matter of upload re-connection and NFR-8 concerning real-time collaboration correction have not been achieved. They were not necessary but desired requirements.

Through the use of an automated transcription service integrated into QDAcity, we have sped up a tedious part of the research process, making it more time and cost-efficient, while eliminating the need for another third party tool or service in order to get from recording interviews to analyzing them.

Researchers can use QDAcity to obtain an automated transcription using state-of-the-art Google deep learning neural network algorithms for automatic speech recognition and partake in the correction process with proper software tools.

References

Atkinson, J. M., Heritage, J. & Oatley, K. (1984). *Structures of social action*. Cambridge University Press.

Estdale, J. & Georgiadou, E. (2018). Applying the iso/iec 25010 quality models to software product. In X. Larrucea, I. Santamaria, R. V. O'Connor & R. Messnarz (Eds.), *Systems, software and services process improvement* (pp. 492–503). Springer International Publishing.

Google. (2020). App Engine Service Level Agreement (SLA) [[Online; accessed 27-October-2021]].

Juang, B.-H. & Rabiner, L. R. (2005). Automatic speech recognition–a brief history of the technology development. *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*, *1*, 67.

Këpuska, V. & Bohouta, G. (2017). Comparing speech recognition systems (microsoft api, google api and cmu sphinx). *Int. J. Eng. Res. Appl*, *7*(03), 20–24.

MacQueen, K. M. & Milstein, B. (1999). A systems approach to qualitative data management and analysis. *Field Methods*, *11*(1), 27–39.

McLellan, E., MacQueen, K. M. & Neidig, J. L. (2003). Beyond the qualitative interview: Data preparation and transcription. *Field methods*, *15*(1), 63–84.

Rupp, C. et al. (2014). *Requirements-engineering und-management: Aus der praxis von klassisch bis agil*. Carl Hanser Verlag GmbH Co KG.

Schöpe, M. (2017). *Qdacity quality metrics*. [Master's thesis]. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).