TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Department of Software Science

Aivo Olev 177994 IAPM

WEB APPLICATION FOR AUTHORING SPEECH TRANSCRIPTIONS

Master's thesis

Supervisor: Tanel Alumäe, PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Aivo Olev 177994 IAPM

VEEBIRAKENDUS KÕNESALVESTISTE TRANSKRIPTSIOONIDE LOOMISEKS

Magistritöö

Juhendaja: Tanel Alumäe, PhD

Tallinn 2019

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aivo Olev

07.05.2019

Web Application for Authoring Speech Transcriptions

Abstract

Creating and authoring transcriptions is a growing need in a variety of industries due to the ease and growing need to create audio and video content and make it then be readable by both humans and machines. Automatic speech recognition systems have improved to a level where the resulting transcriptions are useful for human transcribers of all levels.

The thesis presents a new web application for transcribing and editing Estonian speech audio and video recordings. It combines an audio player and a text editor to offer an optimised workflow for manually correcting the mistakes in the initial transcription and for adding basic text formatting and styling.

In addition, an efficient constant-time algorithm for colouring words in editable text during audio playback was developed which can be used in other similar applications.

Experienced users of an automatic transcription service were surveyed for evidence of the tool's usefulness and for suggestions for improvements. The results showed that respondents were satisfied with the tool and its features and preferred to use it in the future compared to the alternative.

Present thesis is written in English and is 64 pages long, including 6 chapters, 2 tables and 11 figures.

Veebirakendus kõnesalvestiste transkriptsioonide loomiseks

Annotatsioon

Heli ja video meediumite kasvav populaarsus paljudes erinevates valdkondades tingib suurenevat vajadust kõne ümberkirjutamise järele, et muuta seda loetavaks nii inimestele kui masinatele. Automaatse kõnetuvastuse süsteemid on järjepidevalt arenenud ja suudavad enamasti luua transkriptsioone, mis on piisavalt heaks lähtepunktiks nii algajate kui ka professionaalisete transkribeerijate jaoks.

Käesolev magistritöö esitleb uut veebipõhist rakendust, mis pakub optimeeritud kasutajaliidest heli ja video salvestiste ümberkirjutamiseks. Sellesse on integreeritud omavahel tihedalt seotud helimängija ning lihtne tekstiredaktor. Rakendus kuvab ka automaatselt tuvastatud kõnelejate vahetumist ühe salvestise lõikes ning infot kõnetuvastuse ebakindluse tasemetest üksikute sõnade lõikes. Need kombineeritult pakuvad kasutajale optimeeritud töövoogu automaatselt tuvastatud teksti korrigeerimiseks ja vormindamiseks.

Rakenduse loomise käigus loodi konstantses ajas toimiv algoritm heli mängimise ajal sõnade värvimiseks, mida saab kasutada ka teistes rakendustes, kus tekst võib samal ajal muutuda.

Rakenduse empiiriliseks hindamiseks küsitleti kogenud automaatse transkribeerimise teenuse kasutajaid, eesmärgiga saada tõestust toovahendi kasulikkusest, tagasisidet juba arendatud võimalustele ning et ka leida kasulikke uusi suundi, mida rakendus veel ei kata, kuid mille vastu kasutajatel on huvi. Tulemustest võib järeldada, et rakendus ning selle pakutavad võimalused on sihtrühmale kasulikud ning nad eelistavad tulevikus selle kasutamist. Töö teeb ka järeldusi rakenduse põhilistest kasutajagruppidest ning seab tagasisidest tulenevad eesmärgid tuleviku edasiarendusteks.

Töö on kirjutatud inglise keeles ning sisaldab teksti 64 leheküljel, 6 peatükki, 2 tabelit ja 11 diagrammi.

List of Abbreviations and Terms

API	Application Programming Interface
DOCX	Open XML document format
DOM	Document Object Model
GraphQL	Graph data query and manipulation language
JSON	JavaScript Object Notation
REST	Representational State Transfer
WER	Word error rate
Virtual DOM	Virtual Document Object Model

Contents

1 Introduction **11**

1.1 Motivation 12

1.2 Problem statement 13

2 Background **15**

2.1 Related Work 16

2.1.1 State of the Art in Estonian Speech Transcription 16

2.1.2 Similar Software Tools 17

2.1.3 Contributing Research 19

2.1.4 Future Trends and Possible Extensions 20

2.2 Implementation Outline 20

2.3 Functional Requirements 23

2.4 Software Libraries and Tools 24

2.4.1 Front-end Libraries and Tools 24

2.4.2 Back-end Server Libraries and Tools 25

3 Implementation **27**

3.1 Editor Library Evaluation 27

3.1.1 Evaluating the Rich Text Editor Draft.js 29

3.2 Data Representation 31

3.3 Word Coloring Algorithm 33

3.4 Data Caching 36

3.4.1 Caching JavaScript and Requests 36

3.4.2 Server-Rendering 37

3.4.3 Caching User Data 37

3.5 Implemented features 38

4 Empirical Study **41**

4.1 Methodology 41

4.1.1 Survey Data Collection 42

4.2 Study Results 43

5 Discussion **52**

5.1 Future Work 55

6 Conclusion **57**

List of Figures

1 Screenshot of the text editor built using Draft.js 30

2 Screenshot of the Draft.js editor state object 31

3 Screenshot of the completed text editor 40

4 Study results - Previous respondent experience (translated from Estonian) 44

5 Study results - Used for work or school (translated from Estonian) 45

6 Study results - Most useful features (translated from Estonian) 46

7 Study results - Satisfaction with the tool (translated from Estonian) 47

8 Study results - Requested additional features (translated from Estonian) . 49

9 Study results - Likeliness to recommend (translated from Estonian) 50

10 Study results - Use cases for an automated transcription service (translated from Estonian) 51

11 Study results - User interface user-friendliness (translated from Estonian) 51

List of Tables

1 Word coloring algorithm - Adding a DOM reference to the map object for every millisecond of each word 35

2 Predicted mean satisfaction score for the whole target group 48

Chapter 1

Introduction

Creating and authoring transcriptions of speech recordings is a growing need in a variety of industries due to the ease and growing need to to create audio and video content and make it then be readable by both humans and machines.

There have been great advances in speech recognition which can be utilised to automate the speech transcription process. Microsoft has already claimed to have reached human parity [1]. The latest version of an Estonian speech recognition system has achieved a word error rate of *8.1%* for broadcast conversations and *22.7%* for various user-made recordings [2].

The purpose of the thesis is to create a speech to text transcription authoring tool for audio and video files containing Estonian speech, which uses the advantages of automatic transcription based on machine learning models. It should enable users to edit the auto-generated transcription so that it would stay in sync with the audio track. The tool is built using the latest web technologies to enable easy distribution and access and to have an intuitive user interface. The thesis evaluates possible front-end architectural options and attempts to establish one that is suitable for the current use cases as well as more advanced functionalities in the future.

To an extent similar commercial and open-source tools (although not as user-friendly) exist for that purpose for other languages but to author's knowledge, no publicly available option currently exists for the Estonian language.

The authoring tool is a web application that allows users to upload audio and video files containing Estonian speech, have those automatically transcribed, edit those transcriptions in a text editor along with an integrated audio player. The application is validated through a survey of users of an automatic transcription service and conclusions and suggestions are inferred from its results.

The thesis is organised as follows. *Chapter 1* defines the problem statement. *Chapter 2* provides an overview of used software tools, describes the application implementation process and analyses related research. *Chapter 3* presents results from the application implementation phase. *Chapter 4* defines the methodology of an empirical study and presents its results. *Chapter 5* discusses the results form the implementation and the study. *Chapter 6* concludes the thesis.

1.1 Motivation

Manual transcription of spoken content has many downsides and in many situations does not make sense economically. That is because its costs are directly proportional to the man-hours spent on the transcription process. There are also areas, for example optimising content for web search engine rankings, where perfect transcription quality is not required and a lower accuracy rating is almost as valuable.

Transcriptions are also created in different industries by non-professionals for whom the task is tedious and for whom any tool that would assist them in that would be well accepted, even if the later effort of correcting the transcription could objectively take longer than starting from a blank slate.

Professional transcribers could use the benefits of automation as well. There is an inflection point from where the auto-generated transcription's accuracy must exceed the time it takes to make manual correction to achieve the desired accuracy level. There has been scientific research to estimate that in terms of an accuracy percentage.

In cases where the transcription has to be in sync with the audio track the desired accuracy percentage is much lower because to manually transcribe and time-code every word or blocks of words (in the case of creating subtitles) takes extra time which an automatic transcription editing tool can optimise.

1.2 Problem statement

The first goal of the thesis is to build a web application which benefits professional speech transcribers in their every-day workflows. It should also benefit amateur transcribers, for whom transcription creation tasks are not their main responsibility. Suggestions for the design of the user interface and functionalities are taken from earlier research and other similar tools.

The second goal of the thesis is to validate whether building such a tool with modern web technologies results in a usable and intuitive user interface that is appreciated by users.

The third goal is to establish a software architecture which performs well under the conditions where text is being edited, audio is played back and the currently spoken words are highlighted all at the same time. That creates unique challenges because one depends on the other and since playing back audio generates events many times per second, the application state calculations must be fast and efficient so that it could be done even on mobile devices. The architecture should ideally also enable real-time editing by multiple users and rich text annotations. That would mean that the same application can be extended to enable users to annotate text and collaborate in real time.

The main functionalities of the application shall be logging in, uploading and managing existing audio and video files, generating new transcriptions for those, authoring transcriptions, doing common word-processing activities, time-coding transcriptions and exporting those into different formats. In addition, the research looked for other possible features from existing research literature and open-source and commercial applications.

Two promising features found from research were the ability to add a speech-to-text dictation option and to highlight word recognition confidence levels.

Chapter 2

Background

Automatic speech recognition is one of the fastest growing areas in the framework of speech science and engineering. Research in speech processing and communication, for the most part, was enthused by people's desire to build mechanical models to follow human verbal communication capabilities [3]. The two main technical challenges in adopting speech recognition are a) making speech-recognition systems robust in noisy acoustic environments, and b) creating workable speech-recognition systems for natural speech. Although great progress has been made the problems remain unsolved.

When it comes to the nature of data being used and analysed in scientific research, it has undergone at least two big changes in the last decade. Firstly, in terms of the disciplines that the development of speech and language corpora brings together, which encompass at the very least acoustic phonetics, speech technology, natural language processing, lexicography, socio-linguistics and linguistics more generally, but also include aspects of psychology and musicology. Secondly, much bigger quantities of data can be processed because of advances in hardware [4]. This means that the speech and language disciplines require effective and semi-automated applications to assist working with increasing quantity of data. Those tools also should be relatively easy to use so that researchers from various disciplines have a smaller barrier to entry just in terms of usability of the tools.

2.1 Related Work

Automatic speech recognition efficiency has been studied in various fields [5]. A study of professional transcribers looked into the advantages and disadvantages of using automatic speech recognition and have studied the workflows of professional transcribers to identify the operations that such tools can benefit. The conclusion was that it had the potential to increase the productivity and creativity of the professionals, but the advantages can be overshadowed by a reduction in transcription quality unless thorough revision processes are in place [6].

Professional transcribers can have lengthy standards that they have to adhere to and that no current automatic transcription tool can currently follow. Another study looked at three examples of automated transcription of audio recordings from different contexts: an interview, a public hearing and a classroom setting, and compared them against *manual* transcription techniques in each case [7]. They concluded that an automated solution could provide a useful first draft for use in later cycles of the process of speech transcription, provided that the quality of the automated transcript is sufficient to serve as a foundation for further editing, addition and improvement. The mismatches observed between automated and manual transcripts could mostly be easily identified and rectified in a review and edit of the automated transcript.

Regardless of the type of transcript required, it is commonly accepted that the process of transcription is likely to require multiple rounds of engagement with the audio file [7]. That means it is vital that a good transcription editing tool make such an interaction convenient and painless for it's users.

2.1.1 State of the Art in Estonian Speech Transcription

Current conversational language recognition in Estonian and Finnish languages has achieved a word error rate (WER) of 21.9% in one study [8]. Another studies from Estonia have established an estimated word error rate of 26.4% for general-domain dictation application and 13.7% for radio broadcast dictation in 2014 [9]. The latest results are even better at a word error rate of 8.1% WER for broadcast conversations, 12.9% WER on conference speeches and 22.7% WER on various user-made recordings [2].

This thesis will use that system as a web service for automatic transcriptions. That system also performs punctuation recovery and speaker identification. Speaker identification is useful for radio broadcast recording but is less useful for other recordings since it requires the presence of speaker models. It can still recognise when the same person is talking in the same recording with decent accuracy.

2.1.2 Similar Software Tools

Web applications for automating transcriptions editing and with embedded audio and text alignment have been created for other languages [10], [11]. For example, the following paper [12] describes an application called SPA with a front-end architecture that supports various audio and text processing modules: automatic transcription, speech disfluency classification, emotion detection, dialog act recognition, age and gender classification, non-nativeness detection, hyperarticulation detection, dialog act recognition, and external modules for feature extraction and dual-tone multi-frequency detection. It also provides a REST API for integration with other applications. They conclude that such a platform is flexible and scalable.

The current thesis will also use a similar front-end architecture but will not at first implement as many processing features. In addition, it will attempt to also integrate basic functionalities of word processors into the application, which most tools have not implemented.

Partially similar software tools have been actively implemented to enable users to annotate text [13][14]. There are a multitude of things that can be labelled about speech, for example words with an emotional valence [15]. Many alternative tools exist though most have been developed using server-side languages like Java and Python, relying less on front-end technologies, and many don't utilize the web platform [16].

DialogueView program for example offers users 3 views of the speech dialogue: word view, which shows the transcribed words time-aligned with the audio signal; utterance view, which shows the dialogue line-by-line as if it were a script for a movie; and block view which shows an outline of the dialogue [17]. That kind of abstraction to show or hide details can helps users focus on their annotation task and can be a great user interface design feature. For the tool developed during this thesis it is relevant for example for enabling users to refine and export subtitles out of the time-coded text. Those users might probably need to see the text in subtitle-size chunks and might want to refine the time-coding of those chunks.

An overview of common annotation tools concludes that many suffer from common problems which should be avoided. The authors list six most common ones: 1) inadequate importing, exporting or conversions; 2) lack of documentation or support; 3) difficult to learn; 4) poor user interface; 5) difficult installation; 6) unstable, slow or buggy [18].

Text annotations can be just notes or comments written for reader's own needs or shared annotations written for the purposes of collaborative writing and editing. Annotations can also be created for natural language processing and research needs. The tools used for those needs are very relevant for the current thesis because it attempts to enable text basic text annotations that are common in word processors. Its architecture should also be able to support more specialised text annotation workflows that are useful for labelling text for research purposes.

The advances in web-technologies have resulted in the most popular development environment for software developers being Visual Studio Code built using web technologies [19]. The web platform has also become a paradigm shift for developing software tools for speech and language corpora because it has made it easier to seamlessly deliver an up-to-date version of the application to users [20]. This can bring both benefits as well as drawbacks for building such applications. Because software runs in the web-browser, it does not depend on the computing platform. The new version of the application is also immediately available without software reinstalling needed to be performed. There are also some disadvantages like the requirement for a moderately fast internet connection, internet-browser incompatibility issues, less access to the local machine for the application and possible data privacy issues.

Despite the downsides, this thesis sees a great opportunity targeting the web and using the vast availability of open-source technologies and libraries to build a feature-rich and intuitive tool with an architecture that can easily be extended for more natural language and speech processing features.

2.1.3 Contributing Research

The thesis looks for inspiration in how to make the user interface more effective for users. Specific ways to aid transcription editing have been researched in [21]. Also related topics like transcriber performance and fatigue are established in literature [22]. There are also beneficial insight into the effectiveness of common text highlighting techniques [23], which are considered in the design of the user interface in the current thesis.

Some users might also want just statistics and detailed information on the text content in the transcript. There is such a tool built for the Swedish language [24] and if something similar existed for Estonian, it would be rather simple to integrate via a web service.

Since the thesis is interested in establishing an architecture that could in the future support other speech and language processing features, other such tools are relevant. Such are sounds in the speech and physical gestures of speakers. Annotating these multi-modal recordings presents a different set of requirements than the annotation of text. Special purpose tools have been developed and reviewed which display video and audio signals and allow the creation of time-aligned annotations [25]. There are also web-based speech processing tools being built with advance audio visualisation capabilities, speech segmentation information and functionality to manually correct derived speech signals such as formant trajectories [26].

2.1.4 Future Trends and Possible Extensions

Looking further into the future, an overview of current trends concludes that exclusively time-based segments in many annotation tools prevent modelling the complex non-linear relationships in representing speech and language. Those more complex relationships require a data model to be used. And there has been for in coming up with more standard models so that work from different researchers can be used and their finding more easily repeated [4]. That is something that is interesting to consider even here although for this thesis a linear model is good enough. But to support other future editing features, adding such a data model might become a necessity.

Reaching good quality transcriptions allow for interesting additional features like augmenting a video lecture transcription automatically with screenshots [27] or inserting additional text via dictation [28] and even generating and inserting speech into the linked audio file which resembles the original speech [29]. There they also allow users to edit using the waveform of the audio file, which will be used also in the thesis.

Transcripts can also be very beneficial in video editing [30]. Such applications can inspire further extensions or integrations with the tool implemented in the thesis.

2.2 Implementation Outline

The implementation of the application was done in several stages. The following list describes these as steps to take to reach a finished application:

1. Define the functional requirements.

2. Choose the right tools:

   (a) Choose the appropriate programming languages.

   (b) Compare the existing libraries.

    (c) Evaluate the libraries that meet the requirements.

    (d) Choose other technologies.

3. Data representation:

    (a) Integrate with the transcription service and process the result.

    (b) Extend the data representation format of the editor.

    (c) Implement the back-end server data model.

    (d) Implement an efficient data exchange and caching between client and server.

    (e) Create an editor toolbar.

4. Audio player integration:

    (a) Invent an efficient algorithm for the word colouring problem during playback.

    (b) Implement global state management to support seeking audio from the text editor.

    (c) Create an audio player toolbar.

5. Supporting back-end server work:

    (a) Support for user accounts, authentication, file upload.

    (b) Implement exporting to the *DOCX* format.

6. Supporting front-end work:

    (a) User login, logout, registration and password reset flow.

    (b) Uploading and managing files.

    (c) Other user interface related work.

7. Deployment and publishing:

    (a) Set up and configure the web, database, email, back-end API and front-end servers and a reverse proxy.

    (b) Publish securely on a new internet domain.

Before anything was implemented, the functional requirements were defined because those affect the infrastructure, language and library choices. After the requirements were clear, languages and their libraries needed to be evaluated because those could set severe limitations to the whole project either already during the current project or somewhere down the line.

There were two algorithmically most challenging features and those were tackled first. The first was extending the data representation in the text editor and finding an efficient method of saving data to the server. There was also a need for a method for appropriately storing data in a database and in a file system.

The second challenge was to find an algorithm to efficiently colour individual words in the text editor during audio playback which would not freeze mobile web browsers even when the text is over 100 000 words long, as audio playback events can fire over 10 times per second.

The remaining work was building a reliable back-end server and a front-end user interface to support other use cases besides just text editing. The final goal was to complete a system that was ready to be used by users of all experience levels and computer skills and was ready to be evaluated with a user survey. After the completion the system was published on a new virtual server with a custom configuration.

2.3 Functional Requirements

The functional requirements for the web application stem form the research goal to build a useful web application for transcription editing. Some of the requirements also look further into the future and expect the application to be extendable to support more advanced features from other web-based text editors and other natural language processing workflows like adding annotations and creating subtitles.

1. Base requirements which are necessary for transcription editing:

   (a) Each word in the text can be linked to a section in the audio file, defined by a start and an end time.

   (b) User must be able to edit text freely.

   (c) By knowing the current audio playback progress the currently playing word must be highlighted in the transcribed text.

   (d) The editor must support long texts of at least 100 000 words.

   (e) The system must reliably determine changes to the editor contents and calculate a new state based on the old state and the change.

   (f) Representation of the editor state and the state change must be space-efficient.

   (g) Media player component must back common audio and video files.

2. Additional functional requirements that are expected to contribute to a good editing experience:

   (a) Editor must have basic text processor capabilities with a possibility extend to support any rich content.

   (b) Detected speaker info and word recognition confidence levels must be shown inside the editor.

3. Future functional requirements to take into consideration:

   (a) User must be able to add and edit comments on portions of the text. Comments should be part of the editor's contents.

   (b) Editor must support real-time editing of text editor contents in two or more web-browsers simultaneously.

2.4 Software Libraries and Tools

This section provides an overview of the software tools used for its implementation. It is presented in two sections: front-end libraries and tools, back-end server libraries and tools. Front-end technologies were more critical for reaching the research goal.

2.4.1 Front-end Libraries and Tools

The architecture of the front-end of the application is based on the Next.js library, which is a way to server-render React.js applications on initial page load on a Node.js server. Next.js does it by using a build process which separates the application into pages (otherwise a React.js application would be a single-page application). The building process puts the source code into multiple code bundles, some of which are shared between these pages, some are only specific to individual pages.

React.js framework is itself notable as well. It is currently the most popular framework to build single-page applications. This choice does not set any limitations for using just regular JavaScript libraries or approaches, it just encourages a standard approach and structure for the application, which one can deviate from when needed an it was required in this thesis. In general, React.js mainly makes it simple to build front-end applications and the structured approach will become very useful once the application's code-base grows and maintainability and extendibility become more important.

Other important JavaScript libraries that were used were:

1. Quill.js, which ended up being the preferred rich text editing library. There is a separate project in GitHub named react-quill, through which Quill was imported into the project. However, more advanced use cases required using Quill directly.

2. Wavesurfer.js, which is a well-established and powerful speech tool, audio player and waveform visualizer, first published in 2000 [31]. It is now also available as a JavaScript library.

3. React-Apollo makes it easier to fetch data from the GraphQL API, provided by the back-end server. It also simplifies caching of that data and keeping the state of the application globally available throughout the front-end application.

2.4.2 Back-end Server Libraries and Tools

Technical choices for building the back-end server were easier to make than for the front-end client because the functional requirements were not demanding for any mainstream back-end technology. Since it was the first time for the author to build a back-end server, Node.js, a JavaScript runtime was preferred because of good familiarity with the JavaScript language.

At the time of writing the default approach to build an API for a front-end application is to use the representational state transfer (REST) architectural style. A newer approach is to have a single API endpoint and use GraphQL, a query language to describe the contents of a request. While both approaches have their strengths, those are hardly relevant here to compare. GraphQL queries were mainly used in the web application of the thesis, although a couple of REST API endpoints were eventually added as well to support some file transfers.

Node.js has some great tools to build GraphQL servers and some of those were utilized here. The most important back-end server libraries and technologies were:

- Node.js - a JavaScript runtime.
- Express.js - a minimalist library for building web servers.
- GraphQL Yoga - a wrapper around Express.js which offers a default configuration for building a GraphQL server.
- Prisma - a technology and an abstraction around databases. It generates a JavaScript API for communicating with the database. Thanks to that API, no direct database queries were required to write during the application development.
- PostgreSQL - the chosen database technology.
- Docker - a container management software used here to deploy the database server. It was very useful for ensuring an identical environment for development and production environments.

Chapter 3

Implementation

The implementation phase of the research was about analysing and experimenting with available software libraries to find those that support the kind of architecture that was needed to reach the architectural goals set in the thesis. The selection was mainly done by ruling out libraries based on their documented features. Two libraries were then selected for prototyping and performance measuring.

After the software libraries were chosen, a fully working web application was built and published so that it could be assessed by means of empirical research.

This chapter describes the results gained during the implementation of that application. It starts with an evaluation of rich text editing libraries and details how one of the candidates was evaluated. It then continues through the notable features which were implemented and describes the solutions found along the way.

3.1 Editor Library Evaluation

Adding support for text editing was one of the most challenging aspects of building this transcription editing software.

Web browsers render web applications as HTML documents, represented as a single document object model (DOM) tree. In order to satisfy the requirements to store meta-data for each transcribed word and to insert arbitrary rich content into the editor means that it must be possible to manipulate the inner DOM elements of the editor's parent element.

HTML has the ContentEditable native widget for making a DOM element's content editable [32]. In the current case the text editor parent element should be made editable so that child elements like words can be freely edited and removed. It has however severe cross-browser and other inconsistency problems which make using a helper library a necessity, as recommended even in the official specification [32]. The creators of the Medium.com editor propose that a good rich text editor should satisfy the following mathematically defined axioms (ContentEditable satisfies none of those) [33]:

1. The mapping between DOM content and visible content should be well-behaved (should be a morphism).

2. The mapping between DOM selection and visible selection should be well-behaved (should be a morphism).

3. All visible edits should map onto an algebraically closed and complete set of visible content.

There exist well established rich text editing libraries for JavaScript like CKEditor and TinyMCE. These however are heavily reliant on browser DOM and the ContentEditable widget and don't offer an API to define new rich content entities [34]. Although there is now a WebComponents standard which could be enough to extend the DOM by defining new HTML elements so it's possible that this is no longer a strong argument.

There is a newer library Draft.js, open-sourced by Facebook, which seemed very promising as it has proven itself in many of the Facebook's own products. It is dependent upon React.js but since this thesis' application also uses it, it is not an extra burden.

There are Trix, Quill and ProseMirror libraries which implement and maintain a data model to operate on through their API-s instead of allowing users to modify the DOM directly. These also treat the ContentEditable widget as only an input source. Quill and

ProseMirror are also suitable for real-time collaboration features [34]. Of these three, Quill seemed to offer the most modular approach and appeared at least on paper to be easier to extend. That is why it was chosen for this thesis to evaluate and implement.

3.1.1 Evaluating the Rich Text Editor Draft.js

Draft.js was evaluated because it's built on top of React.js which would make it easier to integrate into a React.js web application.

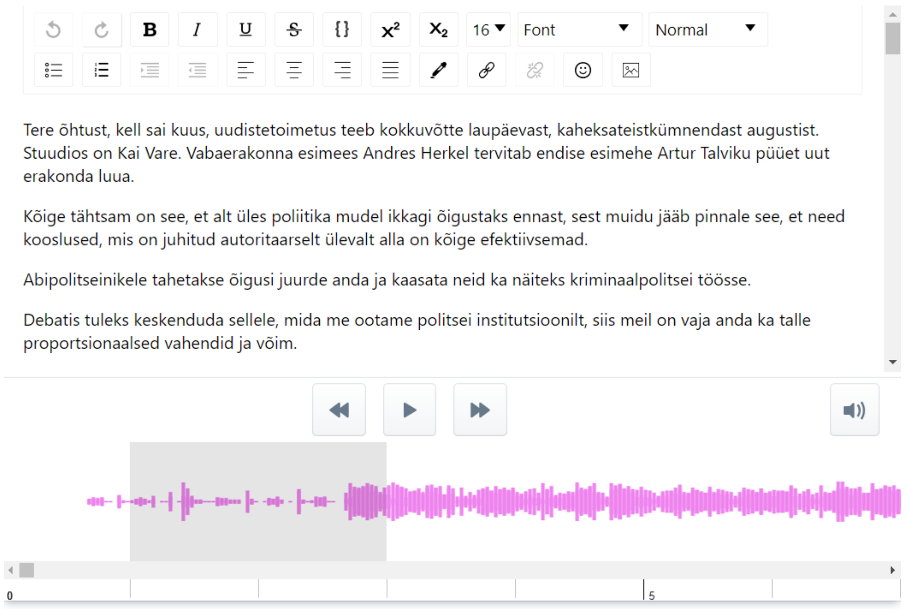
The following steps were taken during the evaluation:

1. Create a React.js application and create a Draft.js editor component.
2. Transform a sample transcription JSON object into the Draft.js format and insert it into the editor.
3. Create a custom entity to hold metadata for each word.
4. Integrate an audio player and control audio from the editor.
5. Measure the state representation of the editor contents using the Redux.js library.

These steps were relatively easy to accomplish and many of the basic functionalities were implemented, as illustrated on Figure 1.

The last step however revealed severe problems due to which that application was discontinued. Firstly, Draft.js does not keep custom entity data in the editor DOM but all state is in a separate big state object as shown on Figure 2.The sample transcription text had 1588 words from an audio clip with a length of $13min13s$. A state object to represent that had 132183 rows, almost 83 times more than the number of words.

Although many of those rows were just empty lines with brackets, it's still a lot. The whole document was represented as a structure of blocks (equivalent to paragraphs in this example), entities (here equivalent to words) and characters. A block held a list of objects



Tere õhtust, kell sai kuus, uudistetoimetus teeb kokkuvõtte laupäevast, kaheksateistkümnendast augustist. Stuudios on Kai Vare. Vabaerakonna esimees Andres Herkel tervitab endise esimehe Artur Talviku püüet uut erakonda luua.

Kõige tähtsam on see, et alt üles poliitika mudel ikkagi õigustaks ennast, sest muidu jääb pinnale see, et need kooslused, mis on juhitud autoritaarselt ülevalt alla on kõige efektiivsemad.

Abipolitseinikele tahetakse õigusi juurde anda ja kaasata neid ka näiteks kriminaalpolitsei töösse.

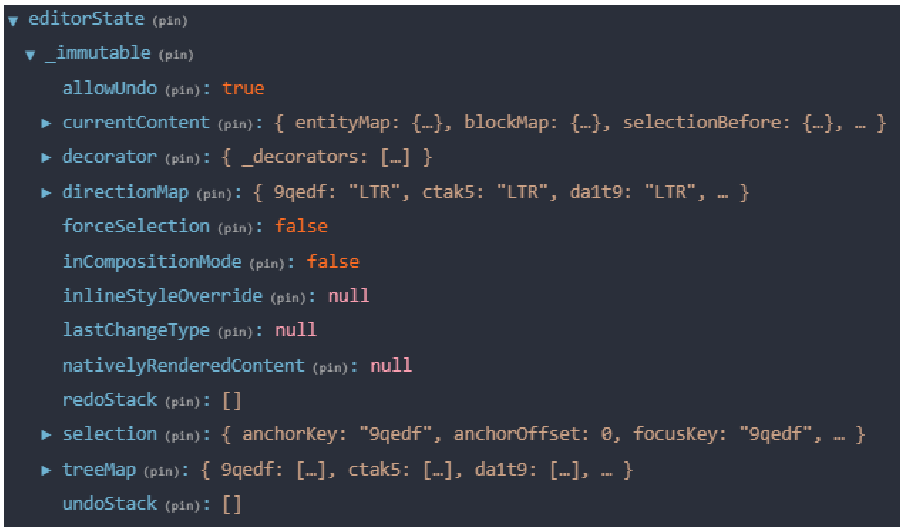
Debatis tuleks keskenduda sellele, mida me ootame politsei institutsioonilt, siis meil on vaja anda ka talle proportsionaalsed vahendid ja võim.

Figure 1: Screenshot of the text editor built using Draft.js

which defined every character in that block and their relation to an entity in a separate *entityMap* object, if that existed. And there were other objects like *treeMap* which held the actual custom data for each entity. While the optimal representation for the current case would have been to represent state by default in word-sized pieces, Draft.js bloated the state by always creating an object for each character in addition to the block and entity level mappings.

This problem was compounded by the fact that a new such state object was created even when user changed the cursor position or changed focus from the editor. And while a cursor change did not create a very different state object (one or two parameters were changed), whenever user removed or added a word to the beginning of the document, all entity mappings were re-indexed, the indexes for all character objects in the same paragraph (block) were recalculated etc. So an experiment on the same sample data showed that deleting a word from the beginning of the document would need to be represented by a state change object with 49691 rows, 38% of the full state object's length.

That is hardly optimal for sending and receiving state updates to and from the server, not to mention the computational cost of calculating those state differences. Thus it became apparent that Draft.js was not suitable for long text documents and certainly not for



```
▼ editorState (pin)
  ▼ _immutable (pin)
      allowUndo (pin): true
    ▶ currentContent (pin): { entityMap: {…}, blockMap: {…}, selectionBefore: {…}, … }
    ▶ decorator (pin): { _decorators: […] }
    ▶ directionMap (pin): { 9qedf: "LTR", ctak5: "LTR", da1t9: "LTR", … }
      forceSelection (pin): false
      inCompositionMode (pin): false
      inlineStyleOverride (pin): null
      lastChangeType (pin): null
      nativelyRenderedContent (pin): null
      redoStack (pin): []
    ▶ selection (pin): { anchorKey: "9qedf", anchorOffset: 0, focusKey: "9qedf", … }
    ▶ treeMap (pin): { 9qedf: […], ctak5: […], da1t9: […], … }
      undoStack (pin): []
```

Figure 2: Screenshot of the Draft.js editor state object

applications that want to support real time collaboration.

3.2 Data Representation

The chosen front-end library for the text editor was Quill.js. It has a format named Delta to represent operations to reach any editor state which is a strict subset of JSON and is very minimalist [35]. It consists of an array of operations which when performed from a starting state will always lead to a singular ending state. The start state can be an empty state, and then the operations will only have *insert* operations.

The starting state can also be any other state and then, besides *insert*, *retain* and *remove* operations are also used to describe only the part of the state that should be changed to reach the new state. This has the desirable result that to describe the change to a document, only the index values for spans of retained and deleted characters plus any newly inserted content have to be included. Such a minimal representation is ideal for synchronising state changes in a real-time collaboration setting and for minimising the payload during saving changes to the back-end server.

In the Delta format there is also only one way to represent any single DOM tree, which means that there is a (mathematical) morphism from a set of operations to a DOM tree and the other way around as well. This guarantees consistency between operations defined through API calls, which get triggered for example when user clicks a button on the editor toolbar, and modifications to the DOM tree made by a user directly in the editor.

By default the Delta format is restricted to only support a small subset of HTML elements and properties. Quill.js however has a Parchment child-project which allows to define arbitrary new HTML elements and properties [36]. Parchment is also Quill's document model. For the current application two new properties were defined to hold the start and end times of each word. In addition, a new element (or *Blot* in Parchment terminology) was defined to hold speaker data. It was defined as an element that can embed arbitrary DOM content and was used to render a React component holding a list of all speakers in the document plus the currently selected speaker.

This validated that Quill.js is flexible enough to represent any DOM trees while being quite optimal and consistent in representing the full state object as well as any state change.

The web application built in this thesis sends an audio file to the automated transcription service and receives back a large JSON file. Its transfer to the front-end was implemented also as a JSON file. There it gets transformed into the Delta format. After any modification by the user all later states of the transcribed text are represented in the Delta format. When including a timestamp together with the Delta operations of the changes made, it can be sent to the server and stored there. Since however it is slightly more optimal to describe a full state only in terms of *insert* operations, it is beneficial from time to time send a full editor state to the server. That full state should also include a timestamp, and overwrite the accumulated set of *insert*, *retain* and *remove* operations.

3.3 Word Coloring Algorithm

During audio playback the currently playing word should be highlighted or coloured. What makes it challenging is that audio playback events happen and should be reacted to many times per second and a document can contain many words, at least 100 000 was set as a target in the current thesis.

The naïve approach to this colouring problem would be to iterate over all the words in the document and as each word has the start and end time, a simple comparison could determine whether to colour the word. That process could then be cut short because it's reasonable to assume a limit that only one word can be spoken at the same time.

Since the playback event carrying the current playback progress can trigger 10 to 20 times per second and the document can contain 100 000 words and other non-textual content which still has to be filtered, it's easy to see that the naïve approach would be a very costly operation in a web-browser environment.

A declarative and the standard approach for passing information in the React.js framework would be very similar to the naïve approach. Data flow in React applications by default is in a single direction – from the parent component to the child component. Which in this case would be from the text editor component to the individual word components. Each word component would then be responsible to determine whether to update their state, in this case whether to render a highlighted word or not. That works well enough in cases where not all child-component are similar but not in this case because here the React run-time has no way of knowing how to optimise its reconciliation process and thus will end up always running a check for each word component each time the parent component changes the passed-in property value.

A more promising approach to determine the word that should be coloured is to **use heuristics** to find the right word quickly in most cases and only in minority of cases execute extra steps. It's quite easy to come up with heuristics for the current problem since a few assertions can be made: 1. Only one word can be played back at any point in time. 2. The total playback time and the current playback time are always known. Which means approximations like average word duration can be calculated. 3. Usually in the next iteration the word to highlight should be the same word or the next one. This does not hold when user seeks to somewhere else in the audio. 4. If the previous word was before the current player progress and the next word is after it then it's proven that no word should be coloured.

This approach was implemented and tested in this thesis but it did not end up being good enough and the frame-rate of the application suffered during audio playback. The problems were that the duration of individual words could vary and more importantly it was not possible to assert whether the speech was continuous and that ended up being the biggest problem – there could be big gaps in the audio where nothing was spoken and that meant that in the worst case the estimations could be tens of words off and had to be recalculated, all checked words had to be remembered at each run and the algorithm could end up jumping back and forth in the text trying to find the right word. An extra check is always added when no word is spoken at that moment – then the algorithm has to prove it by checking the two words around it.

The heuristic approach would be perfect when user could not seek to random places in the audio, as there would be no space-related cost and only 2 checks to do at each player progress event. In the current application however users can seek freely and as often as they please and so in the worst case each event could take multiple calculations and it can be shown that the heuristics are even less efficient the longer the text because the estimation accuracy could worsen.

An optimal algorithm for the given use case would be one that is the most time-efficient. Space-efficiency here is much less important because modern computers have quite a lot of free memory and the quantity of data per word is only two numbers – start and end time. In this case not much space is needed either because it's possible to hold just references to those objects without duplicating any of the containing data in those objects.

Using all the previous assertions there is one more useful assertion to make and it's about the playback progress representation: the required precision is only one-hundredths of a second. That precision gives a good-enough separation between words and ensures that no word gets skipped because of rounding errors.

That allows to create a *Map* data structure that can reference every millisecond in the audio file. It's the classic trade-off between time and space usage, although space usage remains here quite low because only references to DOM elements are stored.

That data structure should be generated only once when the transcription text is received and loaded to the DOM. A code sample of it is provided in Table 1. The structure should be updated each time user adds a new word. I doesn't need updating when words are remove because then that millisecond index will hold a null reference which we can interpret as no speech.

```
const words = new Map();
editor.querySelectorAll("span[start]").forEach(el => {
  const start = Math.round(parseFloat(el.getAttribute("start")) *
     100);
  const end = Math.round(parseFloat(el.getAttribute("end")) * 100);
  for (let i = start; i <= end; i++) {
    words.set(i, el);
  }
  el.addEventListener("click", handleClick);
});
```

Table 1: Word coloring algorithm - Adding a DOM reference to the map object for every millisecond of each word

The update process is also very time-efficient because only those millisecond indexes need to be updated for which words were added. Any existing references can be safely overwritten because of the assertion that only one word can be played at any one millisecond.

Finally, the time-efficiency of the lookup based on this data structure is on average constant time because it's represented as a *Map* data structure and web browsers use hash tables to do lookup from Maps. In JavaScript a *Map* is a keyed collection. As hash-map based lookup is on average a constant-time calculation and comparing the start and end times from a DOM element is also, then we can infer that this algorithm also on average completes in constant time.

3.4 Data Caching

The web application requires a relatively large amount of data to be fetched for common use cases compared to a standard websites. That is because the JavaScript libraries (listed in section 2.4.1 on page 24), large JSON transcription files, media files and less so other user data are quite sizeable here to demand extra attention.

3.4.1 Caching JavaScript and Requests

As described in section 2.4.1, the library Next.js was used to create a build process after which a unique set of JavaScript file bundles are created, some unique to particular pages of the application, some shared between some pages. The bundles are also assigned unique names. That ensures that web browsers can be allowed to cache those bundles and when a new version of the site is released, browsers will download the updated bundles because of the name changes.

Static assets like images and audio files are cached by browsers by default if these are embedded using a URL and not fetched through API requests. All of that ensures that on initial page load, the web browser only needs to load the bundles and assets required for rendering that particular page.

An extra custom API request caching was also implemented for this application because the Wavesurfer.js library requires the file to be fully downloaded before a waveform of the file can be drawn. Initially there seemed to be an option to provide a pre-generated waveform array object and embed the audio file using a direct URL, which would have allowed it to be cached by web browsers by default. But during testing severe problems with the Firefox browser were revealed and since Wavesurfer's documentation itself recommended it also only as a fallback option for older browsers, that method of embedding was neglected.

Luckily most modern browsers have now support for the CacheStorage API which was used to instruct the browser to hold the downloaded audio file in browser cache and check there first always each time that request had to be made.

3.4.2 Server-Rendering

Next.js also helps to server-render the pages of the application by default. That means the first content that the client browser receives is HTML, not just JavaScript which client can only render after having loaded and processed it. It ensures that users see a part of the application quicker which is an important user experience principle.

Since the website contains pages that require users to be logged in, an authorisation token is kept in the browser cookies. Since those cookies are available before server-rendering is done, even the pages which require authorisation and carry user-specific data can and are server-rendered.

Parts of the application that contain either the text editor or the audio player and audio waveform cannot be server-rendered though. That is because the libraries used to build these require access to the browser DOM and since server-rendering happens in a Node.js server, the rendering of those component is delayed to happen after the required JavaScript bundles have been loaded.

3.4.3 Caching User Data

The Apollo-React library and its related child libraries were used to simplify making requests to the GraphQL back-end API, as mentioned in section 2.4.2. Apollo-React makes writing queries in the GraphQL query language easier and provides ways to cache the results.

By default, all request results involving user data are cached in the application, starting from the point where user logs in. After mutation operations like uploading a new file or deleting a file and the associated transcription, the cache is manually updated. After logging out and logging in the cache is invalidated if the user is different.

After user uploads a file there is a long waiting period before the file is processed by the automatic transcription service. A periodic re-fetching was implemented to update the cache with the latest status of the transcription.

This type of application-level caching helps to share data when users move between pages or return to the page again. It reduces page load times significantly because requests to the back-end tend to be time-consuming and can even fail when internet connection is unstable.

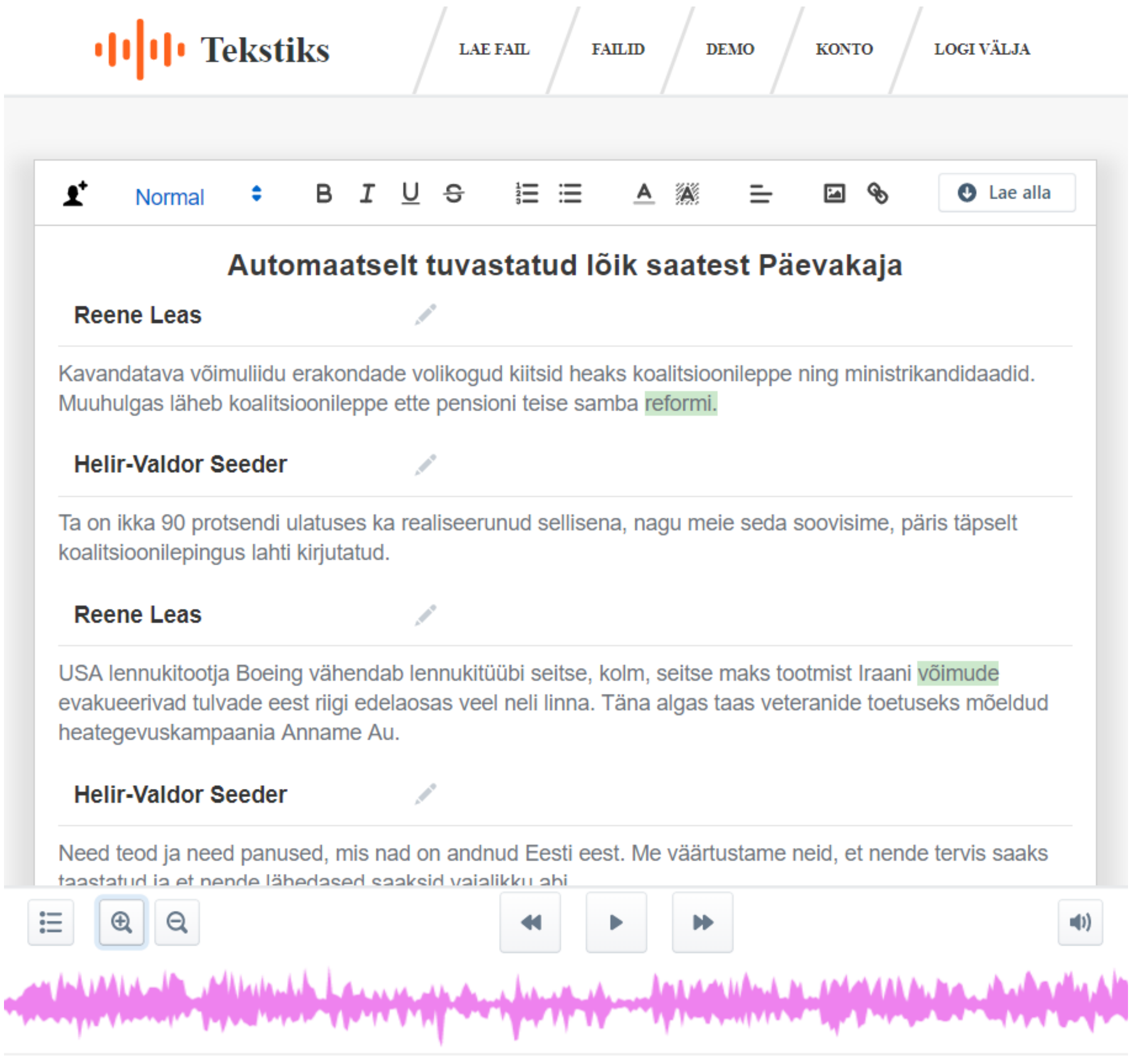
3.5 Implemented features

The full list of notable implemented user-interface features of the transcription authoring web application were as follow. Some of these are visible on the screenshot of the completed text editor Diagram 3:

1. Audio player.
    (a) Audio waveform rendering.
    (b) Playback speed controls.
    (c) Keyboard shortcuts to control audio playback.
2. Rich text editor for transcription editing.

    (a) Custom model entity *speaker* and properties *start* and *end*.
    (b) *Undo* and *redo* support.
    (c) Rich content paste-in support.
    (d) Renaming, adding and removing of speakers.
    (e) Audio playback control when clicks on words.
    (f) Word colouring during playback.
    (g) Automatic state saving to the back-end server.

3. DOCX file generation from the editor contents.
4. File upload and storage.
    (a) File list management.
5. User accounts.
    (a) User sign-up and log-in.
    (b) JSON Web Token based authentication and sessions.
6. JavaScript code and request caching.
7. Global user data caching.
    (a) Automatic user data updates with polling .



Automaatselt tuvastatud lõik saatest Päevakaja

Reene Leas

Kavandatava võimuliidu erakondade volikogud kiitsid heaks koalitsioonileppe ning ministrikandidaadid. Muuhulgas läheb koalitsioonileppe ette pensioni teise samba reformi.

Helir-Valdor Seeder

Ta on ikka 90 protsendi ulatuses ka realiseerunud sellisena, nagu meie seda soovisime, päris täpselt koalitsioonilepingus lahti kirjutatud.

Reene Leas

USA lennukitootja Boeing vähendab lennukitüübi seitse, kolm, seitse maks tootmist Iraani võimude evakueerivad tulvade eest riigi edelaosas veel neli linna. Täna algas taas veteranide toetuseks mõeldud heategevuskampaania Anname Au.

Helir-Valdor Seeder

Need teod ja need panused, mis nad on andnud Eesti eest. Me väärtustame neid, et nende tervis saaks taastatud ja et nende lähedased saaksid vajalikku abi

Figure 3: Screenshot of the completed text editor

Chapter 4

Empirical Study

The first part of the research analysed and experimented with available software libraries to find those that support the kind of architecture that was needed to reach one the research goals set in the thesis. That was followed by implementation of the application, solving data engineering and algorithmic problems along the way, described in the last chapter.

The resulting application includes a new feature set, not validated before on target users. The tool is targeted towards a wide user base whose interests the researcher can't possibly represent.

This meant that an empirical study was needed which would allow to validate whether the hypothesis holds that building this tool with latest web technologies and new features resulted in an intuitive application that is well-received by and useful for the target users. The study would also get input from the potential users about what directions to take in the future and to reveal the types of needs that they have.

4.1 Methodology

Since the application was not targeted toward specific professions or certain industries, finding a representative sample of the Estonian population to study is quite difficult. Luckily TalTech university had a popular publicly available automatic transcription service and email addresses of its users. Since that service was also not targeted toward any specific users then it was a good source for finding a representative population sample.

The application developed in the current thesis in such an early stage that the most useful study method can be argued to be the one that can effectively include a wide sample-size. In addition, qualitative and open-ended data could be more beneficial than quantitatively measuring specific aspects, which the study has to decide and select beforehand. The research did not also have a budget and time to individually interview tens of potential users to guarantee a representative sample.

For those reasons a questionnaire based methodology was selected. The target sample was 298 of the most frequent users of the service (300 initially but 2 turned out to be duplicates). That was decided to be big enough to represent different professions and use cases while limiting the participants to be experienced users in creating speech transcriptions. That limitation allows to draw clearer conclusions, as non-experienced users might desire functionalities and judge the system by random properties whereas experienced users know more what is important for them. It's not to say that those users can't be a source of valuable input as well but they should probably be studied separately.

4.1.1 Survey Data Collection

The participants were contacted through email sent by the researcher. The email introduced the new application as a new interface for the existing transcription service that they were the users of. It gave a quick one-paragraph introduction of the new features, introduced the researcher who created the application, said that the new interface will be from then on freely available in parallel with the other service, and asked for an honest participation while promising that the development of the tool will continue and their input will influence its future directions.

The questionnaire was constructed to have questions that evaluate the application overall and also its specific functionalities. Since the participants were users of an existing service, they were asked to say which evaluate the new tool in light of the new features. They were also asked to list their use cases to find out how the application was useful for them.

The survey also had control questions to make sure the users had extensive previous experience with using automatic transcription services since the questionnaire was publicly available (although no steps were taken to publicly promote the application during the study period).

In total the survey had 9 questions relevant to this study and one for optionally submitting an email address. The answers could be chosen from multiple options and where possible there was an option to add a custom answer. Unless the options were on a scale their order was randomised for each participant to minimise the risk of answer order affecting the average survey results.

The questionnaire was delivered through a web link, embedded into pages of the live application, published at **tekstiks.ee**, where the users had access to the editor. That was chosen to ensure that users got at least some familiarity with the tool while making it easily accessible so as to increase the study participation rate.

4.2 Study Results

In total the study had 34 respondents. As discussed in the previous section, the target sample population was experienced transcribers who were frequent users of an online transcription service which returns the resulting transcription to users as a text file.

The participants who answered that they had transcribed speech only once or never before were treated as outliers and removed from the main data analysis. The proportions are illustrated on Figure 4. That removed 5 respondents and the rest of the analysis was done on the remaining 29 responses. Keeping the 5 responses would not have significantly changed the total results but would have considerably reduced the ability to interpret and to draw conclusions from the results.

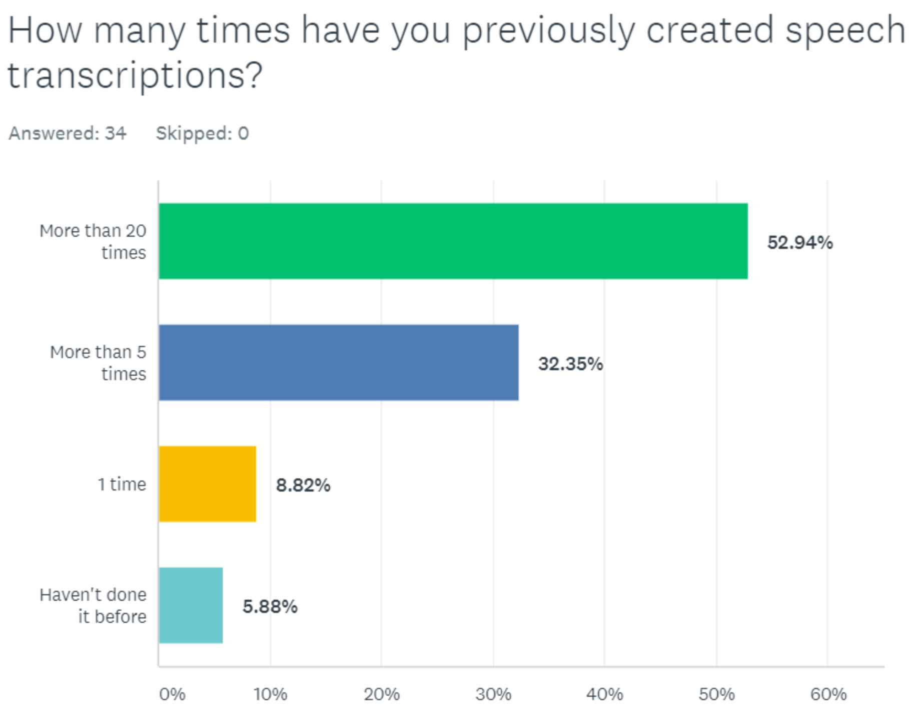


Figure 4: Study results - Previous respondent experience (translated from Estonian)

Among the remaining participants 82% transcribe speech for work or school purposes, as shown on Figure 5. That indicates that most turn to automatic transcription services because of a work or study related need. It suggests that its possible to find common use cases shared by groups of users.

The features of the application that the users marked as the most useful are illustrated on Figure 6. The top two features were marked by almost 70% of respondents and it suggests that the sample group highly appreciated the added features that just a textual transcription service cannot provide them. The most frequently noted feature was the integrated audio player.

Showing words on the sound waveform was one of two features which got the least amount of votes. That was expected because it was by default hidden from users. It needed to be turned on from the audio toolbar. It was hidden because to make the word spans noticeable would have meant also to zoom in the waveform considerably and it would have probably been visually unappealing for most users during playback. But that feature was still implemented and put into the study because it would be required for

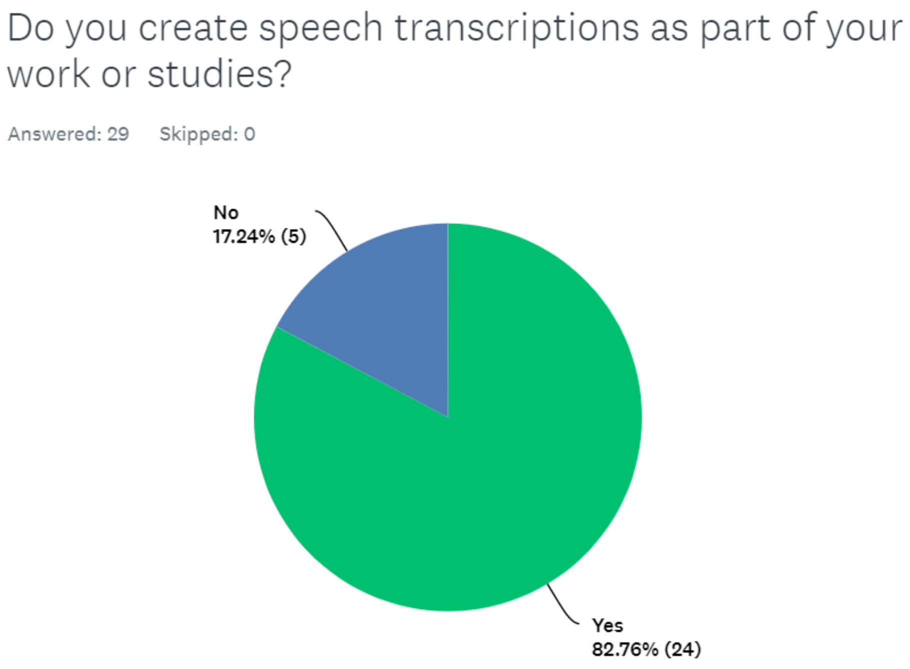


Figure 5: Study results - Used for work or school (translated from Estonian)

supporting subtitle creation and word or word-part annotation use cases. Those features would however require separate user interfaces and the current thesis only tried to find out whether those features would be useful for a significant group of users.

Although the text editor was the second feature to get the least amount of votes, the speaker editing feature was integrated into the editor as a custom entity and so speakers were seamlessly woven into the text contents. This suggests that the choice to invest into an extendable text editor at least for data representation was still worthwhile. The result also suggests that at this point most users preferred to do final text formatting in a fully-featured text editor, if they needed it at all.

As shown on Figure 7, all but one of the respondents marked that they were at least satisfied with the application. 24% responded that they were generally satisfied, 48% chose that they were very satisfied but there were a few aspects to improve, and 24% responded that the application was excellent and very useful. The single respondent who chose that the tool did not fulfil its purpose also later expressed in a comment that her main frustration was with the quality of the automatic transcription, saying that its faster for her to manually transcribe than to correct all the mistakes made by the system.

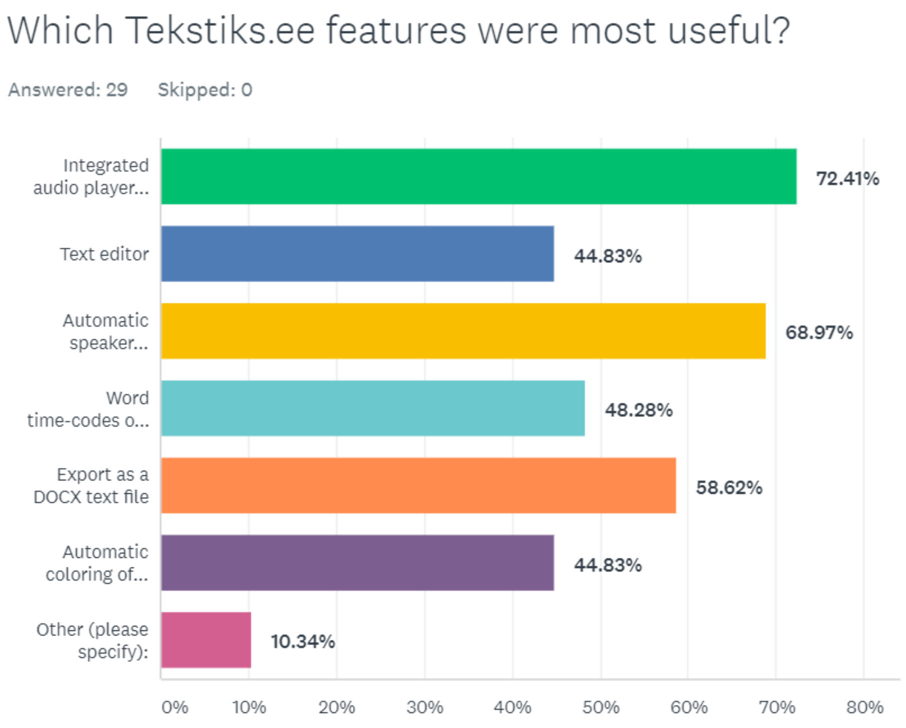


Figure 6: Study results - Most useful features (translated from Estonian)

Since the question was asked on a scale from 1 to 5, the answers can be transformed onto that scale. If 5 is chosen to represent the most positive answer and 1 to represent the most negative answer, then the sample $mean = 3.90$ and sample $median = 4.00$. By knowing the *mean* and *standard deviation* of the sample, a one-sample, two-sided hypothesis test can be performed to predict what would have been the *mean* score if all 298 people had participated in the study (target group *mean* score), at a certain significance level. That population *mean* score will be the null hypothesis.

The most commonly used significance level is 95%, which is good enough here. Table 2 lists for every interesting population score (or null hypothesis) a *p*-value and a confidence interval in between which the *p*-value has to fit for the null hypothesis to not get rejected. In statistics the *p*-value is the probability that result (of the study) occurred (naturally) under the null hypothesis.

As the table shows, all but the top and bottom rows have a *p*-value that is in between the confidence interval and thus there is no statistical reason to reject the possibility that when all 298 people of the target group had participated, they would have given a satisfaction

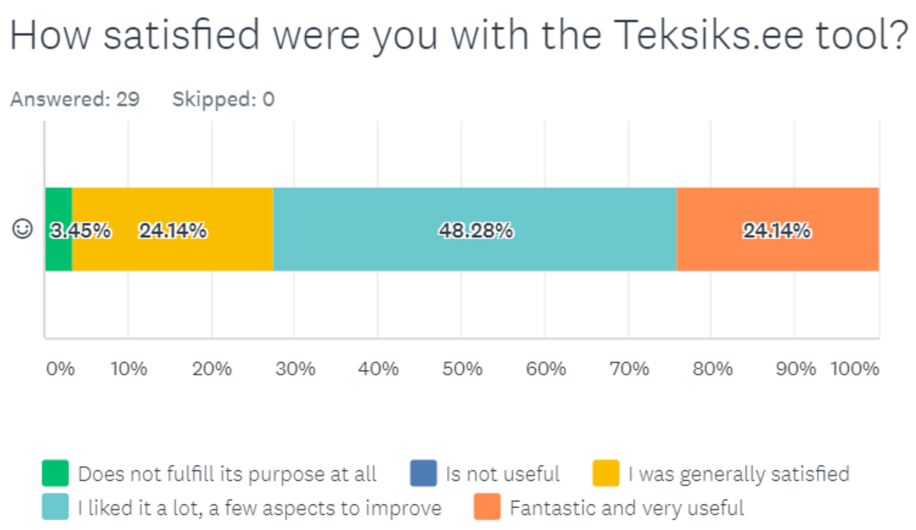


Figure 7: Study results - Satisfaction with the tool (translated from Estonian)

score with a *mean* in the range of 3.55 – 3.80. For the current study that is close enough to the obtained 3.90 and gives a reason to assume that the target group will be at least satisfied with the application.

The requested new features are shown on Figure 8. 44% of the respondents requested support for additional languages, 401% though inputting text using dictation would be useful, 37% wished for ability to share the transcription using a link. Features requested by 25% – 30% of respondent were ability to translate, create and export subtitles and add comments.

The least popular choices were editing word end and start time codes and editing in real time with other users. The editing of time codes could have been very difficult for participant to understand because as mentioned above, the time codes were hidden in the application.

The respondents also had the option to freely vote for other features. The following improvements or features were requested:

1. Showing time codes between text at certain intervals.

2. An option to hide the detected speakers.

Target group *mean* score	95% confidence interval	*p*-value
3.50	0.05 – 0.74	0.03
3.55	0.00 – 0.69	0.05
3.60	-0.05 – 0.64	0.30
3.65	-0.10 – 0.59	0.25
3.70	-0.15 – 0.54	0.20
3.75	-0.30 – 0.49	0.15
3.80	-0.25 – 0.44	0.10
3.85	-0.30 – 0.39	0.05

Table 2: Predicted mean satisfaction score for the whole target group

3. Controlling playback using keyboard and an external foot-pedal.

4. Recognition and separation of non-speech or foreign speech and in case of foreign speech, sending it to another service to process.

5. Possibility to search through multiple transcriptions and audio peculiarities. With the ability to play audio in the search results.

6. Ability to download the application and install locally to transcribe more private files.

7. Improve the quality of the transcription results.

It should be mentioned that keyboard shortcuts, which one respondent requested, were in fact supported in the application, but were not easy to find out about.

Most participants responded that it was very likely that they would recommend the tool to their friends or colleagues, as shown on Figure 9. Only one participant chose that it was not at all likely that they would recommend the tool. The respondent was the same person that added a comment that the quality of the speech recognition was below an acceptable level for her speech recordings and that this tool would be interesting once that was to

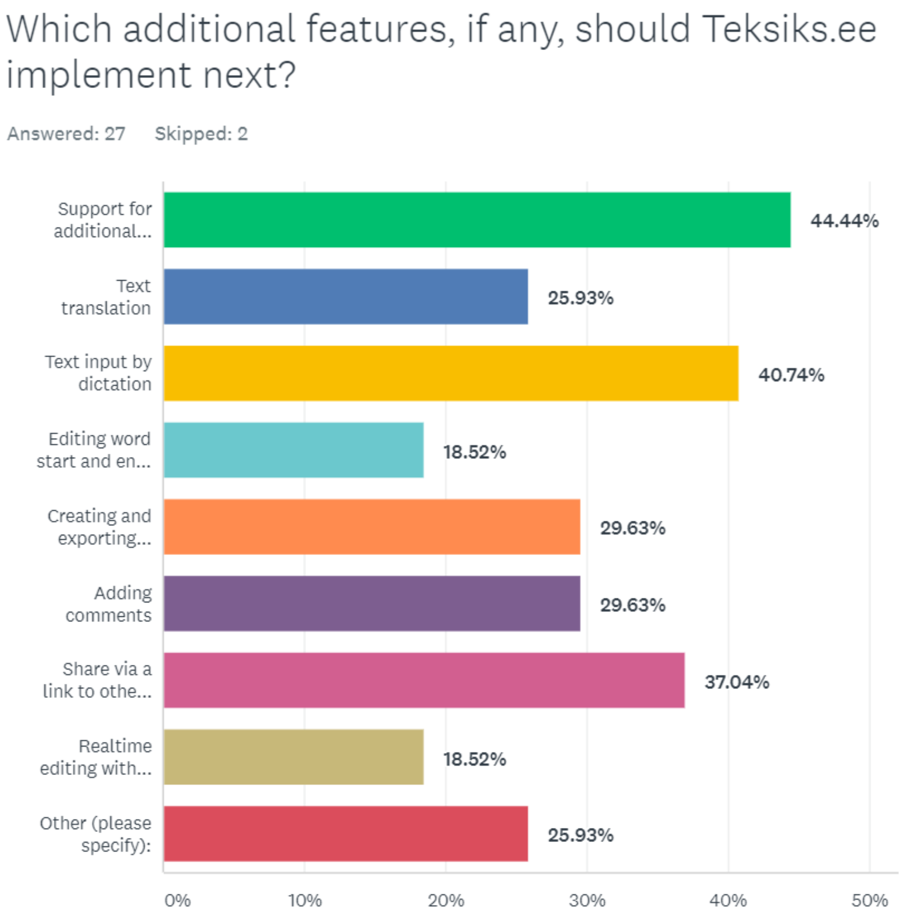


Figure 8: Study results - Requested additional features (translated from Estonian)

improved.

14% chose the average likelihood of recommending and the same amount of respondents chose the above average likelihood. This means that almost all users found the tools useful enough to recommend it to others.

The study also included a question which tried to find out the common use cases that potential users will use the tool for. As illustrated on Figure 10, an overwhelming majority of respondents would use the tool for creating transcriptions of interviews. Another popular option, chosen by almost half of the respondents, was creating meeting transcriptions.

Other chosen options were much less popular. The least chosen option was labelling data for scientific purposes. That suggests that the sample group was not involved in such activities or could not imagine how that tool could one day help them with that activity.

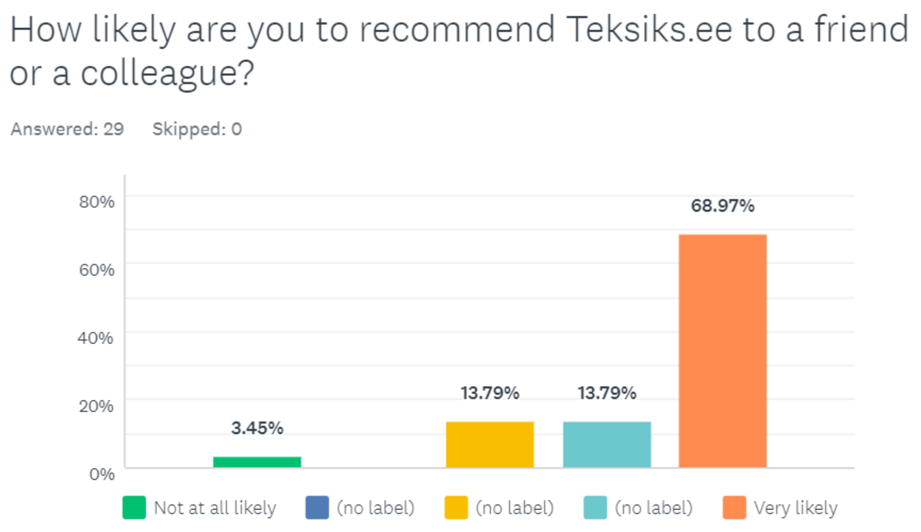


Figure 9: Study results - Likeliness to recommend (translated from Estonian)

Users also mentioned other use cases which weren't among the pre-defined options:

1. Finding new topics in radio shows and then listening to these.

2. Writing down song lyrics.

3. Adding subtitles to private videos so that these could be indexed and made searchable.

Most respondents found that the user interface of the application was very user-friendly 45% or extremely user-friendly (17%, as illustrated on Figure 11. 31% of the respondents found it to be quite user-friendly. While only 2 respondents found it to not be quite user-friendly. No-one chose the most negative of the options. This suggests that overall the user interface was successful but there is quite a lot of room for improvement.

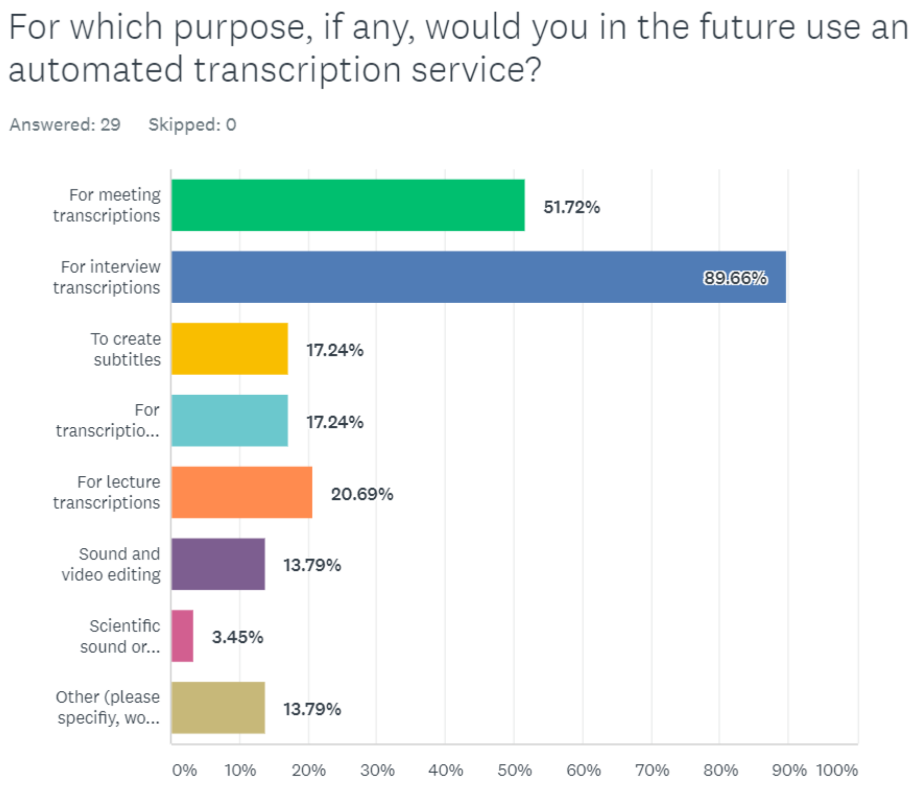


Figure 10: Study results - Use cases for an automated transcription service (translated from Estonian)

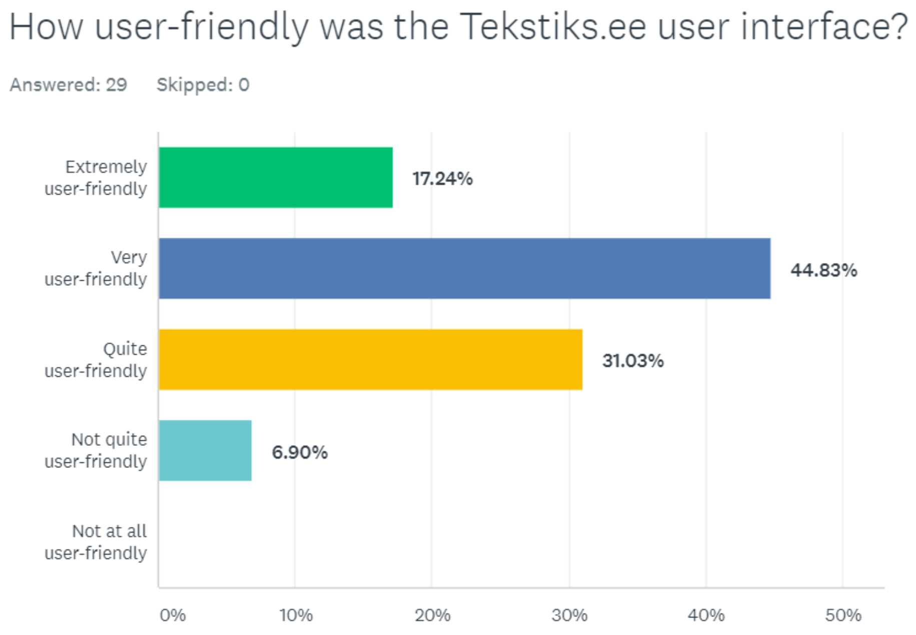


Figure 11: Study results - User interface user-friendliness (translated from Estonian)

Chapter 5

Discussion

Overall, primary objectives of the present thesis were met. As described in the problem statement in section 1.2, the thesis set out to accomplish three research goals:

1. To build a web application which offers benefits to both professional and amateur transcribers.

2. To validate empirically whether building such a tool with modern web technologies results in a usable and intuitive user interface that is appreciated by users.

3. To establish a software architecture which performs well under the conditions where text is being edited and played back and playing words highlighted at the same time. And that enables even more advanced features.

The building of the application was successful and all planned features were implemented. Due to file and request caching (detailed in section 3.4) the resulting application was able to load fast when users used the same functionalities and assets more than once. The pages were also server-rendered (detailed in section 3.4.2) to decrease the time to at least a partially visible page compared to a standard single-page application.

Effort was also put into the text editor and audio player integration, which users ended up voting as their favourite feature, and into the challenge of how to represent speaker data, time codes and confidence levels of each word that were made available from the automatic transcription service response.

The established architecture was built on currently very popular technologies that are being used to build very large applications. The most critical dependency was the text editing library which could have resulted in technical limitations. But the chosen library Quill.js is modular and easily extendable, as demonstrated in this thesis. It is also actively being developed and a popular open-source project on GitHub.com. Its minimalist data representation format proved to hold data consistently and allowed changes done to text during editing be transported compactly. It also meant that arbitrary rich content could be displayed in it, the toolbar could be custom-built, and since the data is represented compactly, it can be used even for real-time editing applications.

This allows to conclude that the architecture scales well and could be used to add arbitrary new language processing features.

The study to validate the application was also successful. The participation rate was large enough to provide a broad range of answers and opinions. According to the survey platform's records, all participants who started the questionnaire also finished it and in general it seemed that the questions were clear to understand and were answered properly.

It must be mentioned however that the questionnaire was not validated beforehand and has not ever been used before so it is possible that some of the questions or options were misunderstood.

It is also possible that from the original sample of 298 transcription service users not a completely random set of participants responded. Since the application and a link to the study was publicly available, then theoretically anybody could have joined it. That was not a problem for the study however because there were control questions to determine whether participants had prior experience creating transcriptions which was the main criteria for participant selection. The study also did not require a prior introduction. The qualitative nature of the study was also welcoming of users who were maybe not on the initial sample but still had interest and experience with creating transcriptions. The final sample also did not need to be completely random subset of the initial sample, same kinds of conclusions can still be drawn.

The results of the completed study suggest that the main user interface features of the application were appreciated by users and 93% of the respondents were at least satisfied with the user interface. And all but one or 97% of the participants were at least satisfied with the tool.

The *mean* satisfaction score of the participants was a high 3.9, as reported in section 4.2. A statistical hypothesis testing gave reason to believe that even if the whole target group of frequent automatic transcription service users (who have experience in creating speech transcriptions) had participated, the score would have been very similar and especially the interpretation of that score. A result above 3.5 is already more toward *very satisfied* than just *satisfied*.

That score could to an extent suffer from a participant self-selection bias because roughly 10% of the initial sample participated and an argument could be made that the ones who decided to participate were more likely to respond to this question in some direction. Current research has however no reason to suspect that. The criteria for success for the thesis was any score above 3.0 and there was only one participant who voted lower than 3.0. That low vote was also not even because of the added value of the application but because she was unhappy about the transcription quality of the underlying system.

The user interface got mostly good feedback but can certainly be improved. Satisfaction with the application was also suggested by a very high *likeliness-to-recommend* score. Targeting the aforementioned use cases should help in designing a more optimised interface. And implementing some of the requested features should also raise the satisfaction scores as long as those don't distract from the main use cases.

5.1 Future Work

One problematic aspect of the application turned out to be the audio player. Once it had a fully loaded audio file, it performed well. The only exception was speeding up or down the audio which caused the sound to distort. That was reported as a bug but it's not clear how quickly and whether it will get fixed. A bigger problem was that allowing users to play audio without having the file fully loaded ended up resulting in major problems on the Firefox browser. The playback progress was reported incorrectly and that lead to the colouring of words in the editor being incorrect.

That time-consuming audio loading requirement was not noticed or at least reported by users, maybe because while the transcription was being generated, the application redirected the user already onto the result page where the audio file was silently loaded into browser's cache.

However, it remains a technical debt that should be resolved in the future, either by finding a fix to the browser inconsistencies or by finding another way to play back the audio.

The empirical study gave good feedback on which features users found most useful. And which ones should be investigated for possible improvements. The study also revealed the most popular ways the application is expected to be used. That is valuable information to concentrate further efforts and possibly create separate interfaces for different use cases which were mentioned in the previous section.

Since most respondents answered that they create transcription as part of work or school activities and the most popular use cases for the tool or automatically generated speech transcription in general were creating interview and meeting transcriptions, then these two are the use cases that the application will target in the near future. Journalists for example create a lot of interviews and including them to conduct detailed user interviews could be a useful next step to optimise the application further. A perspective group of users that record many meetings and require transcriptions turned out to be court employees. Further investigation will be done to explore ways to enable the application to be used in a way that it can be easily locally hosted or installed to support higher privacy needs.

Another possible distinct use case to target further in the future would be subtitle creation. Although, since only 17.24% of respondents requested that feature, an additional investigation should be done to find whether a big enough target group exists and if yes the they should also be interviewed to discover and map their specific needs.

The performed study was mainly qualitative and had its limitations. A quantitative study could be performed to validate further whether there are objective measurable benefits for using an integrated transcription authoring environment. That could be beneficial for example before making big financial investments. For the current thesis that was not and will not be important for the near-term future. Current research is more concerned about finding new ways to support the more popular use cases and about exploring new features which could offer either objective or subjective benefits to target users.

Chapter 6

Conclusion

The primary goal of the thesis was to build a transcription authoring system and validate its usefulness empirically. That goal was met. The empirical study got 34 responses out of which 29 were qualified. The participants had all prior experience creating speech transcriptions. The application received on a *5-point* scale a *mean* satisfaction score of 3.9, where 3 roughly meant *satisfied* and 4 *very satisfied*. Statistical hypothesis testing was done to generalise the result to the whole target group and showed that a similar satisfaction score in the range of 3.6 – 3.8 could be expected from experienced Estonian speech transcribers who already use automatic transcription services.

The results of the study also suggest that building an integrated transcription authoring system can be beneficial to a wide range of users. Participants most liked the ability to listen to audio and have it be in sync with the text. That was possible because clicking on a word would seek the audio to the right location and listening to audio would highlight the currently playing word. Participants also liked that information about speaker changes was displayed and that potential speech recognition errors were highlighted.

The algorithm to enable that kind of word colouring while the text is being edited runs on average in constant time and thus is very effective for that use case since the texts being edited can be long and the audio playback events should be reacted to at least 10 times per second.

The application was built with a flexible, modular and extendable architecture in mind that can in the future support advance word processing, text annotation and subtitle creation workflows. The data in the text editor is represented in a minimalist format that is effective to store and synchronise over the network.

The results of the survey revealed that most users use an automated Estonian speech transcription service for interview and meeting transcriptions and most do it as part of work or school tasks. The professionals who most transcribe interviews and meetings will be targeted and interviewed in the future to find ways to streamline the application further for their needs. The need for subtitle creation support will be validated in the future and if there exists a large enough target group then it will also be implemented.

The study also gave suggestions for improvements and new features, many of which can be implemented in the near future.

Acknowledgements

I would like to thank my supervisor Tanel Alumäe for his encouragement, guidance, knowledge sharing and motivation. And I would like to express my deepest gratitude to my friends and family for unconditional and unfailing support.

Bibliography

[1] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Michael Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, PP, 10 2016.

[2] Tanel Alumäe, Ottokar Tilk, and Asadullah. Advanced rich transcription system for estonian speech. volume abs/1901.03601, 1 2019.

[3] Taabish Gulzar, Anand Singh, Dinesh Kumar, and Najma Farooq. A systematic analysis of automatic speech recognition: An overview. *International Journal of Current Engineering and Technology*, 4:1664–1675, 06 2014.

[4] Christoph Draxler, Jonathan Harrington, and Florian Schiel. Towards the next generation of speech tools and corpora. *Computer Speech and Language*, pages 175–178, 2017.

[5] Tobias Hodgson and Enrico Coiera. Risks and benefits of speech recognition for clinical documentation: a systematic review. *Journal of the American Medical Informatics Association*, 23(e1):169–179, 2016.

[6] Dragoș Ciobanu. Automatic speech recognition in the professional translation process. *Translation Spaces*, 5(1):124–144, 2016. ISSN 2211-3711.

[7] Christian Bokhove and Christopher Downey. Automated generation of 'good enough' transcripts as a first step to transcription of audio-recorded data. *Methodological Innovations*, 11(2):2059799118790743, 2018.

[8] S. Enarvi, P. Smit, S. Virpioja, and M. Kurimo. Automatic speech recognition with very large conversational finnish and estonian vocabularies. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(11):2085–2097, 11 2017. ISSN 2329-9290.

[9] Estonian Tanel Alumäe. Full-duplex speech-to-text system for estonian. In *Human Language Technologies – The Baltic Perspective*, 2014.

[10] Askars Salimbajevs and Indra Ikauniece. System for speech transcription and post-editing in microsoft word. In *Proc. Interspeech 2017*, pages 825–826, 2017.

[11] Hijung Valentina Shin, Wilmot Li, and Frédo Durand. Dynamic authoring of audio with linked scripts. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 509–516, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4189-9.

[12] Fernando Batista, Pedro Curto, Isabel Trancoso, Alberto Abad, Jaime Ferreira, Eugénio Ribeiro, Helena Moniz, David Martins de Matos, and Ricardo Ribeiro. Spa: Web-based platform for easy access to speech processing modules. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA), 5 2016. ISBN 978-2-9517408-9-1.

[13] Jie Yang and Yue Zhang. Yedda: A lightweight collaborative text span annotation tool. *CoRR*, abs/1711.03759, 2017.

[14] Stephen Mayhew and Dan Roth. Talen: Tool for annotation of low-resource entities. In *Proceedings of ACL 2018, System Demonstrations*, pages 80–86. Association for Computational Linguistics, 2018.

[15] Ronald Böck, Ingo Siegert, Matthias Haase, Julia Lange, and Andreas Wendemuth. ikannotate – a tool for labelling, transcription, and annotation of emotionally coloured speech. In Sidney D'Mello, Arthur Graesser, Björn Schuller, and Jean-Claude Martin, editors, *Affective Computing and Intelligent Interaction*, pages 25–34, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[16] Stephen Mayhew and Dan Roth. Talen: Tool for annotation of low-resource entities. In *Proceedings of ACL 2018, System Demonstrations*, pages 80–86. Association for Computational Linguistics, 2018.

[17] Fan Yan, Peter A. Heeman, Kristy Hollingshead, and Susan E. Strayer. Dialogueview: annotating dialogues in multiple views with abstraction. *Natural Language Engineering*, 14(1):3–32, 2008.

[18] Mark Finlayson and Tomaž Erjavec. Overview of annotation creation: Processes & tools. page 7, 02 2016.

[19] Stackoverflow.com. Annual developer survey results 2019, 2019. URL `https://insights.stackoverflow.com/survey/2019/`.

[20] Thomas Kisler, Uwe Reichel, and Florian Schiel. Multilingual processing of speech via web services. *Computer Speech & Language*, 45:326–347, 2017. ISSN 0885-2308.

[21] Juan Daniel Valor Miró, Joan Albert Silvestre-Cerdà, Jorge Civera, Carlos Turró, and Alfons Juan. Efficiency and usability study of innovative computer-aided transcription strategies for video lecture repositories. *Speech Communication*, 74:65–75, 2015. ISSN 0167-6393.

[22] Matthias Sperber, Graham Neubig, Jan Niehues, Satoshi Nakamura, and Alex Waibel. Transcribing against time. *Speech Communication*, 93:20–30, 2017. ISSN 0167-6393.

[23] H. Strobelt, D. Oelke, B. C. Kwon, T. Schreck, and H. Pfister. Guidelines for effective usage of text highlighting techniques. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):489–498, Jan 2016. ISSN 1077-2626.

[24] Jesper Näsman, Beáta Megyesi, and Anne Palmér. Swegram – a web-based tool for automatic annotation and analysis of swedish texts. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 132–141. Association for Computational Linguistics, 2017.

[25] Steve Cassidy and Thomas Schmidt. *Tools for Multimodal Annotation*, pages 209–227. Springer Netherlands, Dordrecht, 2017. ISBN 978-94-024-0881-2.

[26] Raphael Winkelmann and Georg Raess. Introducing a web application for labeling, visualizing speech and correcting derived speech signals. 05 2014.

[27] Hijung Valentina Shin, Floraine Berthouzoz, Wilmot Li, and Frédo Durand. Visual transcripts: Lecture notes from blackboard-style lecture videos. *ACM Trans. Graph.*, 34(6):240:1–240:10, October 2015. ISSN 0730-0301.

[28] Aditya Vashistha, Pooja Sethi, and Richard Anderson. Respeak: A voice-based, crowd-powered speech transcription system. pages 1855–1866, 05 2017.

[29] Zeyu Jin, Gautham J. Mysore, Stephen Diverdi, Jingwan Lu, and Adam Finkelstein. Voco: Text-based insertion and replacement in audio narration. *ACM Trans. Graph.*, 36(4):96:1–96:13, jul 2017. ISSN 0730-0301.

[30] Anh Truong, Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. Quickcut: An interactive tool for editing narrated video. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 497–507, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4189-9.

[31] Jonas Beskow and Kåre Sjölander. Wavesurfer - an open source speech tool. volume 4, 01 2000.

[32] W3C. Html editing apis, 02 2014. URL `https://dvcs.w3.org/hg/editing/raw-file/tip/editing.html`.

[33] Medium.com. Why contenteditable is terrible, 05 2014. URL `https://medium.engineering/why-contenteditable-is-terrible-122d8a40e480`.

[34] Jason Chen. Quill.js - comparison with other rich text editors, 05 2019. URL `https://quilljs.com/guides/comparison-with-other-rich-text-editors/`.

[35] Jason Chen. Delta format documentation, 05 2019. URL `https://github.com/quilljs/delta/`.

[36] Jason Chen. Parchment - documentation, 05 2019. URL https://github.com/quilljs/parchment/.