TipsyTongues

Report for the Praxisprojekt Real-Time Mobile NLP Analysis SS21 at the University of Duisburg-Essen

Jule Buschmann
Front-end
jule.buschmann@stud.uni-due.de

Tabea Graszynski
Design
tabea.graszynski@stud.uni-due.de

Marvin Westphal
Backend, Front-end
marvin.westphal@stud.uni-due.de

***Abstract*—TipsyTongues is a mobile application that tells the user their alcohol level by analyzing their speech via NLP. The user gets encouraged to read a given sentence out loud while recording themselves. After analyzing the audio the user sees their result and can try again. The design of the app is inspired by the general aesthetic of beer and is designed to fit the needs of the user in a dark environment. The Front-End for TipsyTongues was built with Xamarin. Our main task was to implement functionalities such as a permission service, an audio recorder, an audio player and a working Back-End connection, as well as exception handling. The Back-End server was developed using Python and the server framework Flask. We used Microsoft Azures Cognitive Services to perform a pronunciation assessment of a given audio file in comparison to a reference sentence. From that pronunciation assessment the users level of drunkenness is calculated.**

***Index Terms*—app design, NLP, InVision, VisualStudio, Xamarin CSharp, Audiorecording, Speech Recognition, Microsoft Azure, Cognitive Services, Python, Flask, Heroku, Speech Recognition, Pronunciation Assessment, calculate level of drunkenness,**

I. Introduction

This report describes our process of developing the mobile application TipsyTongues. First, we shortly discuss our motivational background I-A, and perform a brief market analysis I-B. Next, we go more in depth on Natural Language Processing I-C, and how it is important with respect to our application.
Furthermore, we describe our design process and how our ideas and visions were implemented. Subsequently, we detail our Front-End development, which includes an outline of our software decisions, an explanation of our main functionalities, and major issues we faced during the development process. Lastly, we describe the data flow from request to response in our Back-End server.

A. Motivation

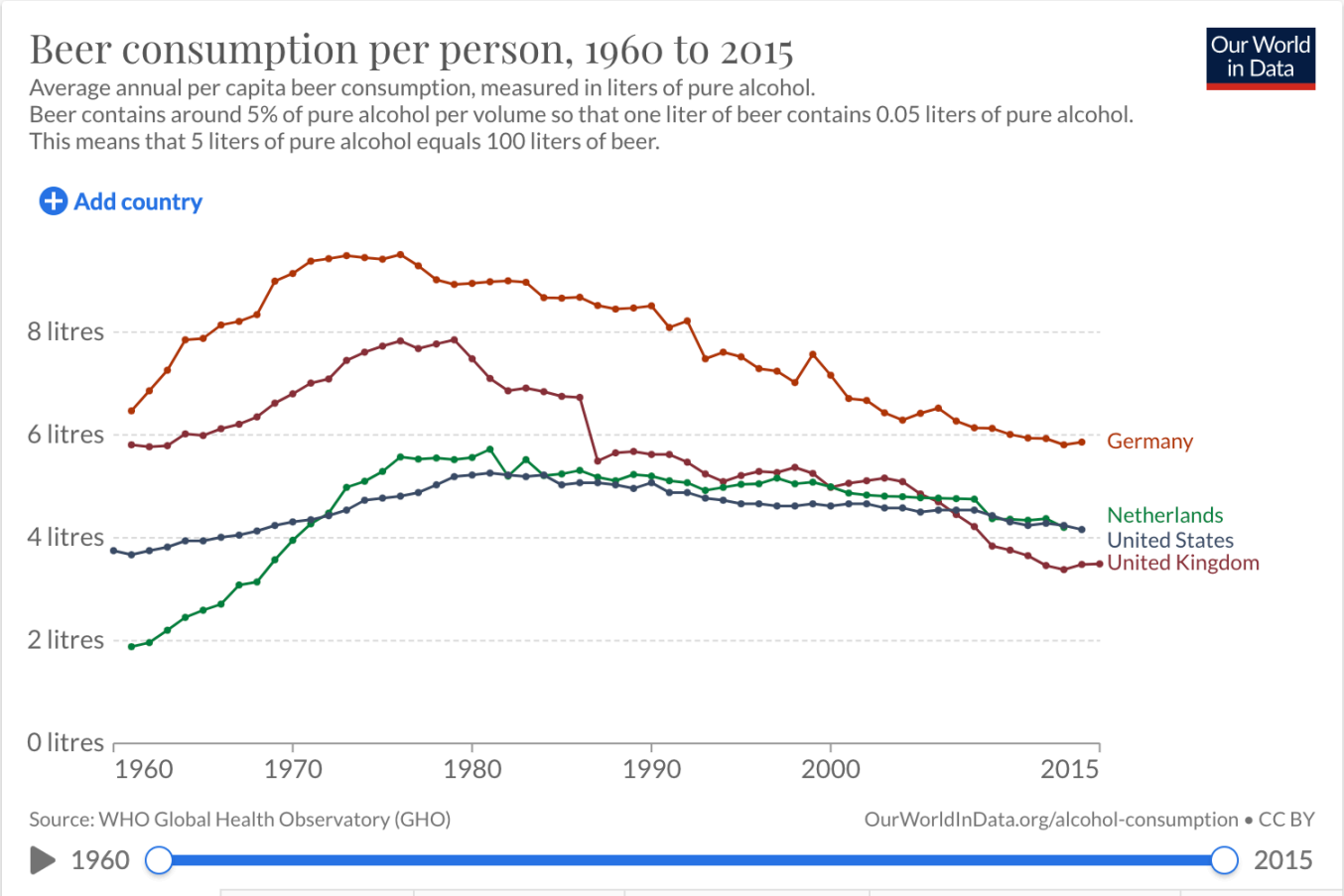
When going out for drinks with friends, the typical party-goer tends to remain unaware of their slurred speech pattern until a friend points out their tipsy tongue. Because of that, one may struggle to find the right moment to switch out their alcoholic beverage for water.
Our app replaces that friend and tells the user how drunk they are and when to get that water. The app is thought to be a party gadget as well, to test out together with friends, and compare drunkenness scales. The app is aiming at English speaking users within the legal drinking age.

B. Existing Apps / Market Analysis

When taking a look at the market of drinking related applications it becomes apparent that there are already countless options for drinking games like ”Drunken Wizards” [1] or ”Picolo” [2] . But something that resembles TipsyTongues cannot be found on the Appstore yet, thus there is no application that analyses drunkenness through speech recognition. Something that is comparable regarding at least some factors is on the one hand an ”Drunk Voice”-Video Editor [3], which edits the users voice into sounding drunk and on the other hand an application named ”Am I Drunk?” [4], that analyses the user’s drunkenness based on reactivity tasks. Both applications have positive ratings but seem to be lacking a pleasant interface design. TipsyTongues fulfills this criteria as well as being simple and self explanatory, which means it can be used no matter the age or level of drunkenness.
Since the 1970s there is steady decline in total beer consumption, which suggests that more and more people lean towards a rather responsible use of alcohol. [5] Letting an application analyse one’s level of drunkenness could support that responsibility and help with knowing when to stop drinking.
In resume it can be said that TipsyTongues seems to have great potential to soon belong among the ranks of successful drinking game applications.

C. NLP

”Natural language processing (NLP) refers to the branch of computer science — and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.” [6]
One task of NLP is speech recognition. Amazons ”Alexa” for example uses speech recognition to support the user in its daily life. It can provide answers to almost any imaginable question, the answer of which can be found on the web [7]. Furthermore, it can be used for any task, which can be started by spoken command, like starting an entertainment system, shutting of smart home devices, etc. [8].
In our application we use NLP to calculate the level of drunkenness by a spoken sentence. For that we do a pronunciation assessment, in which we compare the users pronunciation to



[5]

Fig. 1. Beer consumption per person, 1960 to 2015.

that of a native speaker. We use Microsoft Azures Cognitive Services for that comparison. Based on the result of the assessment, we calculate the users level of drunkenness, which is then displayed in the app.

D. Overview

In Section II we first discuss the Design of the app, followed by the Front-End and Back-End development in the Sections III and IV. We conclude the report by our future vision in Section V

II. Design

The design part of app development is crucial to create an outlay and give an overview about functions and usability of the app. Furthermore, the design is critical for the first-impression the user is going to get when opening the app for the first time. It is necessary that the design gives information on how to navigate within the app and is a decisive factor for a good user experience. Creating a prototype is the first thing to do when developing an app, so that the Front-End and Back-End developers know, which functions need to be developed, how views are connected and which features need to be included. Optimization of the design follows through the entire development process to fit all the users needs.

A. Design Software

First we created a storyboard of our app idea and the different views with Miro. [9] Miro is a website that provides an online whiteboard together with templates for prototyping. It is an easy tool to gather first ideas and record design visions. We used InVision Studio version 1.28.2 to create a responsive prototype of TipsyTongues. InVision was the best choice to develop a prototype for us, since it has many options to design animations and different changes between views. There are many features that are useful to create a more advanced design that goes beyond different views simply stringed together. On top of that, there is an easy way to share a project with teammates and an option for leaving comments within the prototype. Additionally, InVision is providing the code for diverse design items in several programming languages.
The illustrations and the logo, that are included in the final app were predominantly made with Adobe Illustrator 2020 version

24.0.1.341. Illustrator is used by a lot of professional user-interface designers and has many tools and options to make a good quality design. Furthermore, there are many online tutorials on how to work with Adobe Illustrator that can help when being inexperienced with the software.
For the loading screen we generated a loading animation. The final animation is made with Adobe After Effects 2021 version 18.4.1. As with Adobe Illustrator, After Effects is used by many professional video designers, which leads to a lot of tutorials on how to use the software. After working with Illustrator, After Effects was easier to learn since the layout of the two software resemble each other. Besides, they are compatible which helped when integrating Illustrator designs into After Effects. Before using After Effects we tried to work with Synfig Studio version 1.4.1, which is a open-source animation software. There are not many tutorials on how to use it and the set up can be confusing this is why we decided work with After Effects insetad.

We used Lottiefiles [10] to create animations in JSON file format to include them in the app. To generate the icon for the android and the iOS app we uploaded our logo to the website appicon.co [11].

B. Design Workflow

1) View structure: We started the design process by discussing ideas and creating a storyboard 2 to visualize our vision of how the distinct views are designed and connected.

Fig. 2. Storyboard of TipsyTongues with first ideas.

After that, we designed the views in InVision and built the responsive prototype. It was important to us to start with a disclaimer to remind the user of the dangers of alcohol misuse, as well as drinking and driving. Moreover, we wanted to clarify that TipsyTongues is not accurate enough to give a measure of roadworthiness and should not be used as such.
Following the disclaimer the user is lead to the recording page which includes a box with a sentence to read out loud, a shuffle button to get another sentence, and a recording button. Later in the developing process we added the feature of an easy mode and a hard mode, with the hard mode only generating tongue twisters. The recorded audio file can be listened to in the next view and the user gets to decide whether they want to repeat the recording, in case the audio might not be complete, or if the audio file is good to be analyzed.
While the audio is being processed the user sees a loading animation on the loading page. The result of the analyzing process can be seen in the next view. We included a visual scale and a text below to tell the user their result. We wanted to make it as easy to understand as possible, since our users are potentially intoxicated with alcohol. Beneath the result is a button that takes the user back to the recording page so they can try again.

2) Theme and Color: The app is kept in a dark design as it is presumably used at night time and it shall not blind the user. We decided to keep all of the design very simple, considering that the user might be inebriated while using the app. Furthermore, a minimalist design is recommended when using a dark UI design, for a noisy design can be overbearing for the user [12]. We used the color dark grey (hexcode:121212) as the background color throughout the entire app by reason of it being the recommended color for dark UI design [12] [13].
Because our application is build around alcohol usage we wanted to use a theme that pairs well with it. Early on we had the idea of a loading animation that displays a beer glass filling up, while the user is waiting for their result. That is why, we decided on a beer inspired theme throughout the entire app. With dark UI design it is best to use only a few colors and colors with low saturation so that the interface is not looking too busy [12].
We used yellow (hexcode:FFFF8D) as the primary color [14] to go with the beer scheme. Different matching shades of yellow from the same color palette [14] appear in the disclaimer and in the loading animation. For the secondary color [14] the pink color (hexcode: FF848C) was used. After naming the app TipsyTongues we decided to also include a tongue theme in the app and used the secondary color for the tongues in the logo and for smaller details in the app. For buttons and text fields we picked another grey tone (hexcode:262424) to stick with the dark design.
We chose the font ”Fjalla One” in white (hexcode:FFFFFF) to assure good readability [15].

3) Name and Logo: The applications name ”TipsyTongues” summarizes the purpose of the app. ”Tipsy” stands for the alcohol level measurement and the ”Tongues” represents the used speak recognition as an indicator of drunkenness. The alliteration in the name is supposed to be catchy.
We designed the Logo to match with the apps name, so the logo consists of two tongues that intertwine with each other, like a yin and yang symbol. In front of the tongues, the name ”TipsyTongues” is written, in the font ”Bangers” and in the primary yellow that is consistent in the app.
First we designed the logo in InVision in a different style, and later created an enhanced version in Adobe Illustrator to better fit to the rest of the app. 3

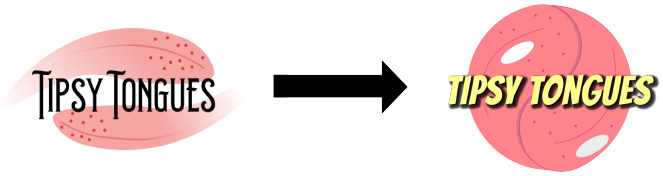


Fig. 3. First logo (left) and final logo (right) of TipsyTongues.

4) Design choices for different views: We integrated the Logo in the Splash Screen and designed it in a vibrant way so the user gets a strong first impression.4

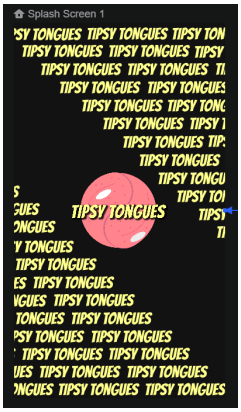


Fig. 4. Splash Screen with logo.

The disclaimer is based on colors of beer and foam. It symbolizes the waves of beer getting poured in. When designing the disclaimer we added many changes over the entire design process 5, to deliver the correct message and to make it interesting to look at, so the user will take the time to read the text.

Fig. 5. First disclaimer (left) and final disclaimer (right).

For the recording page we wanted to keep it simple to not distract from the important features. The recording button shows a tongue wrapped around a microphone, which is consistent with the tongue scheme. On the listening button on the review page headphones in a tongue inspired design are displayed 6. The microphone and the headphones are designed in Adobe Illustrator and then integrated in InVision.

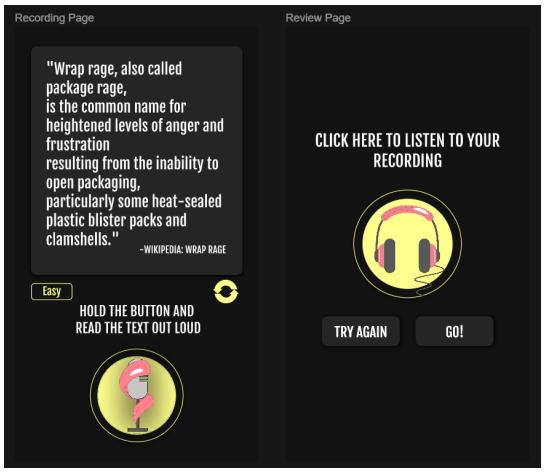


Fig. 6. Recording page (left) and Review page (right).

The beer animation for the loading page was first generated in InVision and later recreated in After Effects.

Results are shown in a scale in the same design as the beer glass in the loading animation. We also included a caption below, showing the results in a written form. We added a home button with the caption ”I just want to go home”, to play with the word home-button and the experience that many people make when going out, where at some point in the night they just want to go home.

C. Design Issues

The first issues that we had were with understanding all of the countless features and possibilities of InVision Studio. Watching all of the provided tutorials by InVision [16] and looking at their design examples helped with getting to know all the important basics to start effective prototyping.

Further, we had troubles with creating an enjoyable color palette, therefore we began to observe other designs in our surroundings and read about color theory [17]. Additionally, we read articles about dark UI designII-B2.

In the beginning we started to make illustrations with InVision, however realized that it is not the ideal software to use for complicated designs. For this reason, we started using Illustrator. The loading animation was the most complicated issue. We first made the animation in InVision, but when it came to programming it, we did not know how to extract it from InVision. After realising that we need the animation in the JSON-format, we used Adobe After Effects and had to recreate the animation. We used the designs from InVision, which looking back, was more complicated for the animation process than creating new shapes in After Effects. Because of this we were not able use some of the features that we could have used otherwise. We fixed this problem by trying random approaches in After Effects.

To create a JSON-file, extra plug-ins had to be installed and

after rendering the animation the quality of resolution got inferior to the shown animation in the After Effects window. We could not solve this problem, even though we changed the settings to the highest possible resolution.

D. Summary

To sum up, we came up with the idea of TipsyTongues, made a storyboard, created a prototype and integrated illustrations and an animation for the design part. We updated our design throughout the entire process to keep up with new requirements and improve imperfect ideas. We learned how to use InVision Studios, Adobe Illustrator and Adobe After Effects.

III. Front-End Development

The primary goal of the Front-End development is to build an app interface, that merges with the server functionalities from Back-End while following along the visual requirements from the Design, thus creating a pleasant looking, frictionless user experience.
In this chapter an explanation of our software decisions, followed by an elucidation of our work and some major problems we faced during the process can be found.

A. Software Decisions

We decided to use Xamarin, which is an open-source framework for building cross-platform native apps without coding in two separate languages and user interface paradigms. We did not want to pass on the advantages of a native user interface especially because of the microphone and storage access we would need. The language C# is object oriented and static typed, which provides benefits like error detection and quick debugging. The integrated development environment for Xamarin is VisualStudio. [18]

B. Work

1) Permission Service: In order to get access to the microphone and storage usage, the user gets a permission request right after the application is launched. This permission service checks which permissions might already be given and asks for those that are not given at this point. If the permissions are not given the user is shown an error page, because these permissions are required for the functioning of the application. [19]

2) Audio Recorder: As one of the main functionalities of the Front-End, a well functioning audio recorder is of great importance. We decided to use Plugin.AudioRecorder [20]. First we had to configure the permissions for Android and IOS separately. In order to set different types of behaviors for our audio recorder, which are displayed below, the class has to be set on top. The methods we use are OnButtonPressed, which lets the user record as long as the button is pressed, OnRecordingTimeout, which sets the time span on 15 seconds in order to prevent an overload and OnButtonReleased, which stops the recording, gets the audio filePath needed for further processing and navigates to the next page. [21]

```
async void OnButtonPressed (Object sender, EventArgs e)
{
    recordingTimer = new Timer(14999);
    recordingTimer.Elapsed += new ElapsedEventHandler(OnRecordingTimeOut);
    recordingTimer.Enabled = true;
    await audioRecorderService.StartRecording();
}

private async void OnRecordingTimeOut(object sender, ElapsedEventArgs e)
{
    await Task.Run(() => Device.BeginInvokeOnMainThread(() => OnButtonReleased(sender, e)));
}

private async void OnButtonReleased (object sender, EventArgs e)
{
    await audioRecorderService.StopRecording();

    String audioFilePath = audioRecorderService.GetAudioFilePath();
    AudioStreamDetails audioStreamDetails = audioRecorderService.AudioStreamDetails;

    await Navigation.PushAsync(new ThirdPage(audioFilePath, sentence, audioStreamDetails));
    Navigation.RemovePage(this);
}
```

Fig. 7. Methods for the Audio Recorder.

3) Audio Player: Implementing an option to re-listen to the recorded audio has several functions. On the one hand, if the user is not content with their recording, they can go back to the recording page without having to go through the whole process of making the application analyze a failed recording. On the other hand, if an audio happens to be funny, the user has the option to re-listen and enjoy it. It works in a relatively simple way. The audio player gets the audio file from the previous page handed over and plays it as the button is pressed. [20] In order to stop the audio player from playing, as this page is left through either ”Try Again” or ”Go”, there is a pause command within the eventhandlers.

4) Back-End Connection:

a) Posting: The content-payload, that is posted to Back-End, contains the sentence that has been read out loud and the data from the audio file that is read into byte array in the previous step. [22] The payload is therefore defined as MultipartFormDataContent. The HTTP Client lets us post the request and receive the results. [23]

b) Receiving and displaying Results: The level of drunkenness is parsed to JSON-format and forwarded to the following page, where the results are displayed. There the result is translated into different image/text combinations.

```
if (response.IsSuccessStatusCode)
{
    var responseBody = await response.Content.ReadAsStringAsync();
    var jsonObject = JObject.Parse(responseBody);
    var levelOfDrunkenness = jsonObject.Value<int>("levelOfDrunkenness");

    await Navigation.PushAsync(new FourthPage(levelOfDrunkenness));
    Navigation.RemovePage(loadingPage);
    Navigation.RemovePage(this);
}
else
{
    await Navigation.PushAsync(new ErrorPage("Service currently unavailable"));
    Navigation.RemovePage(loadingPage);
    Navigation.RemovePage(this);
}
```

Fig. 8. Receiving Results from Back-End.

5) Exception Handling: Errors, from not accepted permissions over to failed recordings, or errors with loading the audio file, are caught throughout the whole application. All of those

exceptions access a universal Errorpage, that changes the error message on basis of what type of error occurs. [24]

C. Issues and Solutions

1) Time Span for the Audio Recorder: We tried several options of implementing a time-out span of 15 seconds, of which none worked at first. Using a timer within the OnButtonPressed method, in conjunction with another OnRecordingTimeOut method, was the first well functioning solution.

2) Creating an Adaptive Layout: Creating a dynamic layout that works cross-platform as well as within different screen sizes was another major challenge for us, since a basic Grid-Layout itself did not seem to function correctly. Some elements would extend beyond the frame or interfere with other elements. Implementing a service that works with height and width percentages of the display was a well functioning solution. [25]

D. Summary

To sum up, the Front-End development for TipsyTongues included some essential functionalities like the audio recorder, the audio player, a permission service and a working Back-End connection as well as exception handling. Apart from that, there are many other relevant implementations, like the launchscreen, the shuffle method for randomizing the displayed sentences or the loading animation.

IV. Back-End Development

In this Chapter we will present our server architecture. In Chapter IV-A we will describe the API-Specification of the Back-End server. Chapter IV-B is about the serverside preprocessing we need to do, before we have all the components to calculate the level for drunkenness. The calculation is described in Chapter IV-C.

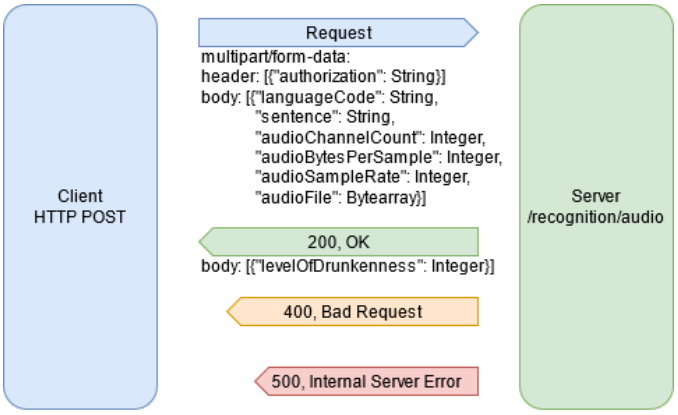


Fig. 9. API-Specification of the Back-End server.

A. API-Specification

We use the OpenAPI-Specification to define the RESTful webservice our backend server provides. We developed a POST-Endpoint, which takes a multipart/form-data request, to get an audio file with a JSON-body in a single request [26]. The integer attributes ”audioChannelCount”, ”audioBytesPerSample” and ”audioSampleRate” are sent in the JSON-body to rebuild a wave-file from the audio file bytearray. The string attributes ”languageCode” and ”sentence” are necessary to do the calculation of the level of drunkenness. The string attribute ”authorization” is sent as a headerparameter, to be compared to a key defined by the server. All attributes are required.
If any of the attributes are missing or the wrong datatype, we respond with the HTTP-Statuscode 400 (Bad Request). Then we check whether the authorization either equals the Front-End testing key, which returns a mocked response, or is the correct authorization key, which leads to the recognition. If the authorization equals neither of the keys, we return an HTTP-Statuscode 400, as well.
For the development of the RESTful webservice we used the Python framework ”Flask” [27]. The implemented Sever was deployed to the cloud application platform Heroku [28]. It can be reached at the address ”https://tipsy-tongues.herokuapp.com/recognition/audio”.

B. Preprocessing

After extracting the elements from the request, we want to rebuild the audio file from the given bytearray and temporarily persist it to the server. Firstly, we define a randomized filename, with which we can access the file. For that, we use the random-package to attach a randomized number to the filename [29]. That is done to prevent an error, when two contemporaneous requests would access the same file, if there was no randomization and they had the same filename.
Then we forward the randomized filename, the bytearray, ”audioChannelCount”, ”audioBytePerSample” and ”audioSampleRate” to the ”wave_file_converter.py”, where we use the wave package to build a wavefile and persist it [30]. After that the audio file is accessible via the filename.
After persisting the file, we forward the filename, the language code and the sentence to the ”pronunciation_recognizer.py” to do the pronunciation assessment using Microsoft Azure Cognitive Services speech SDK. Before the pronunciation assessment, we have to define the configuration the assessment is based on. First of all, we need an active account for Microsoft Azure Cognitive Services. We chose to use the free access for students, which grants us 5 hours of use a month for a year. In our pronunciation recognizer we then use the given subscription key and service region, to define the speech configuration [31]. We build the language configuration from the language code [31]. By getting the language code through the endpoint, the server could calculate the level of drunkenness for different languages. For now, the app only handles american english. We then build the audio configuration from the filename [31].
The speech recognizer is then built from the speech configuration, the language configuration and the audio configuration. After that, we build the pronunciation assessment configuration and apply it to the speech recognizer. In the pronunciation assessment configuration we define the parameters ”ReferenceText”, ”GradingSystem” and ”Granularity” [31]. In

”ReferenceText” we set the sentence, the pronunciation is evaluated against , so the sentence the user read in [31]. The ”GradingSystem” describes the point system for the score calibration [31]. It is displayed in a floating point score, which is either from 0 to 5 (”FivePoint”) or from 0 to 100 (”HundredMark”) [31]. For a more detailed response, we chose ”HundredMark”. For ”Granularity” we chose the option ”Phoneme”, which recognizes the score on the full text, word and phoneme level [31].
After setting all necessary configurations, we do a single-utterance recognition through the speech-recognizer. The speech recognizer assesses the pronunciation based on a 15 second long audio file. The time is restricted by the application (Front-End).

C. Drunkenness calculation

The pronunciation assessment returns the scores ”AccuracyScore”, ”FluencyScore”, ”CompletenessScore” and ”PronunciationScore” [31]. The accuracy score describes how accurate the pronunciation of the speech is, compared to the pronunciation of a native speaker [31]. The fluency score compares the fluency of the speech to that of a native speaker, by matching the use of silent breaks between words [31]. The completeness score is a calculation of the ratio of pronounced words to the reference text input [31]. The pronunciation score is an aggregated, weighted score of the previously mentioned scores [31]. The scores are interdependent, e. g. if the the accuracy score is low, the completeness score is most likely low as well.
We construct a data transfer object (DTO) from the four scores, returned by the speech SDK. The DTO only provides a constructor and getters for every score, so once constructed, the scores can not be changed anymore. The DTO is then forwarded to the drunkenness calculator. The drunkenness calculator contains a static map, which has thresholds to be compared to the scores from the DTO as keys and the level of drunkenness to be calculated as values. For the calculation we iterate over the keys and check whether all scores from the DTO are above the threshold. If that is true, we return the value to the key as the calculated level of drunkenness. Because the scores are interdependent, we compare all four scores to one threshold, e. g. if all scores are above 95, we return 0 as the calculated level of drunkenness, which means the user had an accurate pronunciation result, therefore we expect the user is not drunk.
After the calculation, the persisted filename is deleted from the server. Lastly the calculated level of drunkenness is then returned as an integer in the response. It can range between 0 (not drunk) and 4 (very drunk).

D. Summary

In the last chapter, we described our Back-End architecture. We showed the data flow from request to response and what kind of preprocessing and calculation has to be done to return the level of drunkenness.

V. Future Vision

Our future vision for TipsyTongues includes some Front-End related changesV-A, a new system to calculate the level of drunkennessV-B and an update of our server and developing infrastructure**??**.

A. Update Front-End

In Front-End development we focused on implementing the core features, for the app to run, first. In the early iterations of brainstorming, we thought of some additional features, that could not be added in the given development time. As mentioned before, we set the time for the user to read in the sentence to 15 seconds. For a userfriendly UX-Design we wanted to add an animation to the recording page, which depicts the expiration of the time left, while recording. 10

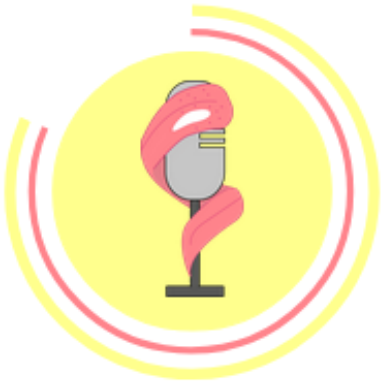


Fig. 10. Animation for pressing the recording button.

Our Back-End architecture allows us to do the calculation based on the languages, Azures Cognitive Services provides for the pronunciation assessment [32]. To do that, we need to change the language code and according to that the reference sentence in the request, that is sent to the server. With that we could add the option for different languages, when starting the app.
For Marketability, we would like to add a sharing-feature to the result page, so that users can share their results in their favorite social network.

B. Update classification

For now, the App is fooled quite easily, if the user just pretends it is drunk and reads the sentence gibberishly on purpose. That is, because the calculation is solely based on the pronunciation. Furthermore using Microsoft Azures Cognitive Services is either quite expensive [33] or can only be used for 5 hours a month. Therefore, with Microsoft as the main classifier, the app is not marketable at all and will not be published to the app store.

To change that, we would like to develop our own classifier. We would like to develop and train a convolutional neural network, which calculates the level of drunkenness by an audio file [34]. For the trainingset we need to gather a dataset containing of id, audio file, reference sentence and the alcohol level of the person that read in the audio file.

C. Update infratructure

At the moment we are using a free version of Heroku as our Back-End server, which shuts itself down, if it is not used for some time. After that the server starts automatically, if a new request is sent. The problem is, that the request takes about 2/3 more time to be answered, if the server needs to start up, than if it is already running. Therefore we would like to deploy the Back-End implementation to a server, which is up all the time.
Furthermore, if this becomes a longterm project it would be useful to have a CI/CD pipeline, which runs automated tests with every new commit to verify the functionality of the implementation. In addition to that, we would like to add a monitoring system, which alarms us, if the server can not be reached.

VI. Where to find the App

https://github.com/marvinwest/TipsyTongues.git

References

[1] G. A. UG. Drunken wizards – trinkspiel. [Online]. Available: https://apps.apple.com/de/app/drunken-wizards-trinkspiel/id1372842624

[2] Marmelapp. Picolo · partyspiel. [Online]. Available: https://apps.apple.com/de/app/picolo-partyspiel/id1001473964

[3] B. O. M. G. . C. KG. Drunk voice - video editor. [Online]. Available: https://apps.apple.com/us/app/drunk-voice-video-editor/id1505999700

[4] M. Spilger. Am i drunk ? - game. [Online]. Available: https://apps.apple.com/kh/app/am-i-drunk-game/id1350220421

[5] W. G. H. Observatory. Beer consumption per person, 1960 to 2015. [Online]. Available: https://ourworldindata.org/grapher/beer-consumption-per-person?tab=chart&country=USA~DEU~GBR~NLD

[6] I. Authors. Natural language processing (nlp). [Online]. Available: https://www.ibm.com/cloud/learn/natural-language-processing

[7] A. Authors. Alexa information. [Online]. Available: https://www.amazon.com/-/de/alexa-information/b?ie=UTF8&node=21588420011&ref_=ftpk_infm

[8] ——. Alexa features? [Online]. Available: https://www.amazon.com/b?ie=UTF8&node=21576558011

[9] M. Authors. Miro homepage. [Online]. Available: https://miro.com/

[10] L. Authors. Lottifiles homepage. [Online]. Available: https://lottiefiles.com/

[11] appicon authors. App icon generator. [Online]. Available: https://appicon.co/

[12] M. Philips. In the spotlight: the principles of dark ui design. [Online]. Available: https://www.toptal.com/designers/ui/dark-ui-design

[13] A. Kulkarni. Dark mode ui: the definitive guide. [Online]. Available: https://uxdesign.cc/dark-mode-ui-design-the-definitive-guide-part-1-color-53dcfaea5129

[14] M. Authors. The color system. [Online]. Available: https://material.io/design/color/the-color-system.html

[15] Google. Google fonts. [Online]. Available: https://fonts.google.com/

[16] I. Authors. Master of invisionstudio. [Online]. Available: https://www.invisionapp.com/studio/learn

[17] K. Decker. The fundamentals of understanding color theory. [Online]. Available: https://en.99designs.de/blog/tips/the-7-step-guide-to-understanding-color-theory/

[18] Microsoft. Xamarin. [Online]. Available: https://visualstudio.microsoft.com/de/xamarin/

[19] M. Authors. Xamarin.essentials: Permissions. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/essentials/permissions?tabs=android

[20] G. Versluis. Record and play audio in your xamarin.forms app. [Online]. Available: https://www.youtube.com/watch?v=2V8J7v3FP78&t=413s

[21] L. Reyes. Working with audio recorder using xamarin forms. [Online]. Available: https://askxammy.com/working-with-audio-recorder-using-xamarin-forms/

[22] ?? How do i read an audio file into an array in c. [Online]. Available: https://stackoverflow.com/questions/47662865/how-do-i-read-an-audio-file-into-an-array-in-c-sharp

[23] M. Authors. Http client klasse. [Online]. Available: https://docs.microsoft.com/de-de/dotnet/api/system.net.http.httpclient?view=net-5.0

[24] ——. try-catch (csharp reference). [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch

[25] ——. Creating a service. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/services/creating-a-service/

[26] S. Authors. Documentation: Multipart requests. [Online]. Available: https://swagger.io/docs/specification/describing-request-body/multipart-requests/

[27] P. Authors. Quickstart - http methods. [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/quickstart/#http-methods

[28] H. Authors. Quickstart - http methods. [Online]. Available: https://devcenter.heroku.com/articles/git

[29] P. Authors. Documantation: random — generate pseudo-random numbers. [Online]. Available: https://docs.python.org/3/library/random.html

[30] ——. Documentation: wave — read and write wav files. [Online]. Available: https://docs.python.org/3/library/wave.html

[31] M. Authors. Pronunciation assessment. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-pronunciation-assessment?pivots=programming-language-python

[32] ——. Language and voice support for the speech service. [Online]. Available: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support

[33] ——. Azure cognitive services pricing. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/

[34] J. Miller, J. Donahue, and B. Schmitz. Speech emotion and drunkenness detection using a convolutional neural network. [Online]. Available: http://www2.ece.rochester.edu/~zduan/teaching/ece477/projects/2018/JoshuaMiller_JilianDonahue_BenjaminSchmitz_ReportFinal.pdf