1 Abstract

Lorem ipsum

2 Table of contents

Contents

3 List of Tables (grids)

List of Tables

4 List of Figures (graphics)

List of Figures

5 List of listings (code snipptes)

Lorem ipsum

6 Introduction

6.1 Initial Situation

According to the Federal Office for the Environment (DE: BAFU) one in seven people in Switzerland is affected by noise pollution [1]. The pollution comes primarily from road traffic, followed by railways and then air traffic. In addition to these noise sources, construction sites, nightclubs and public facilities also produce noise. Because of this, Switzerland has set up upper noise limits that must be respected. However, it is of course the case that these limits are not always adhered to. Affected people must then either accept this or take action against it. For the latter, they must gather evidence to prove their noise disturbance to the police and the courts. This evidence then comes from an audio recording which the affected person made them self.

6.2 Product Goals

To help the people affected by noise pollution, we want to create an application which processes a given sound file (.wav) and analyzes it. It should detect when a specified threshold has been exceeded and then summarizes the result in a PDF document. The document should then contain all necessary information for filing a complaint. As our application will have a wide range of end users, two of our main design goals are to make it as user-friendly as possible and to make it platform independent, so it can be used with any PC operating system and ideally mobile device.

6.3 Problems with Audio Files

A wave file (.wav) contains samples of the recorded audio, where each sample represents the amplitude at a given moment. Those amplitude values are relative to each other and not absolute. It is therefore impossible to determine the actual dB(A) (loudness relative to the human ear) someone would perceive without any further information [2–4]. Because of this, we require the user to also give information about the minimal and maximal dB(A) measured in the given audio file. To do this, we recommend using a smartphone app like DecibelX for IOS, which allows the user to record the audio and also conveniently inspect the minimal and maximal dB measured in that recording. With those two values, we can then map the relative values from the wave file to its db(A) values.

7 Specification

The following chapter describes the system specification. The specification is derived from the requirements in the project description as well as the discussions with the stakeholder.

Assumptions and constraints are described in the following sections and were validated by the defined product owner and the stakeholder.

7.1 System Delimitation

The system delimitation is split into the static system environment (7.1.1) and the dynamic process environment (7.1.2).

7.1.1 System Environment (statics)

The static system environment is split into the three contexts *System*, *System context*, and *Out of scope*. *System* includes the application as well as any software dependencies. The *System context* includes people or objects which have an influence on the application. *Out of scope* describes particularly what could have an influence on the application but has been deliberately excluded by the project team (see7.2.2).

System:

- Frontend (User interaction)
- LaTeX and pgfplots
- WAV file analyser

System context:

- Stakeholder (tutor)
- User
- Noise producers
- Lärmliga
- Recording Device

Out of scope:

- Integration in legal complaints

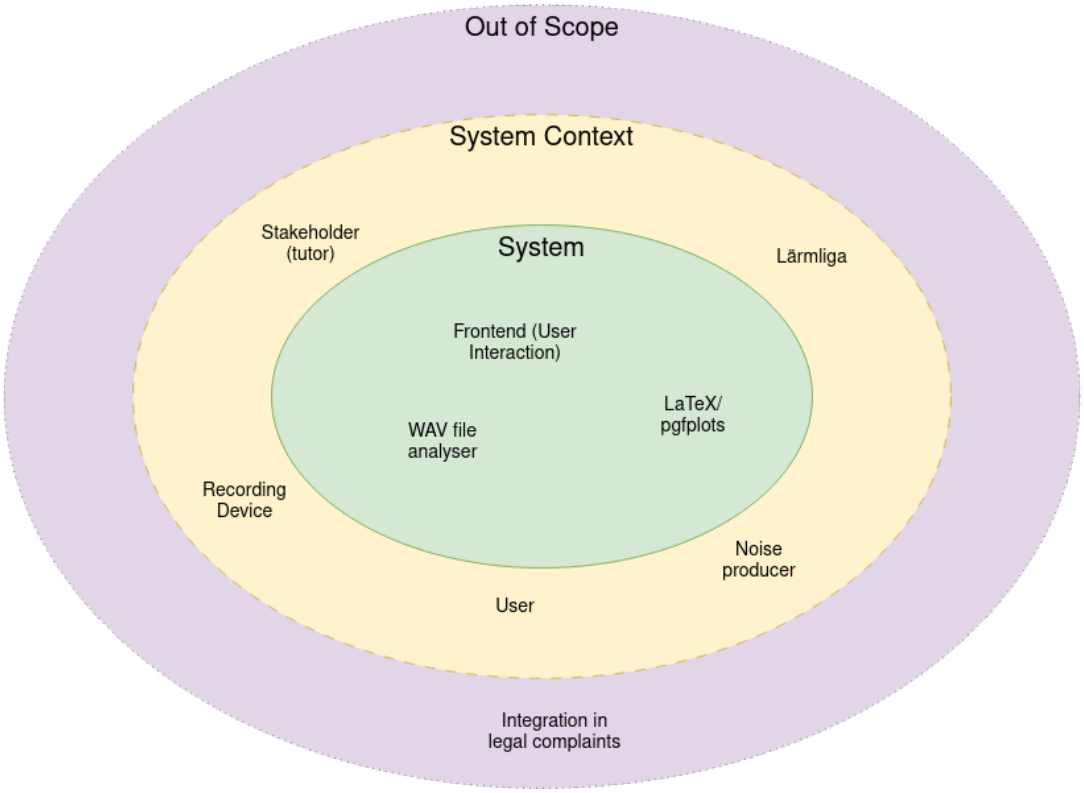

Figure 1: System Environment

7.1.2 Process Environment (dynamics)

As in the previous chapter (7.1.1), the process environment is also split into *System*, *System context*, and *Out of scope*.

System:

- Selecting (uploading) a WAV file
- WAV file analysis (validation, parsing, conversion to absolute db values, threshold filtering)
- Plotting analysis result (LaTeX / pgfplots) and PDF generation

System context:

- Recording noise
- Producing noise

Out of scope:

- Device calibration (see7.2.2)

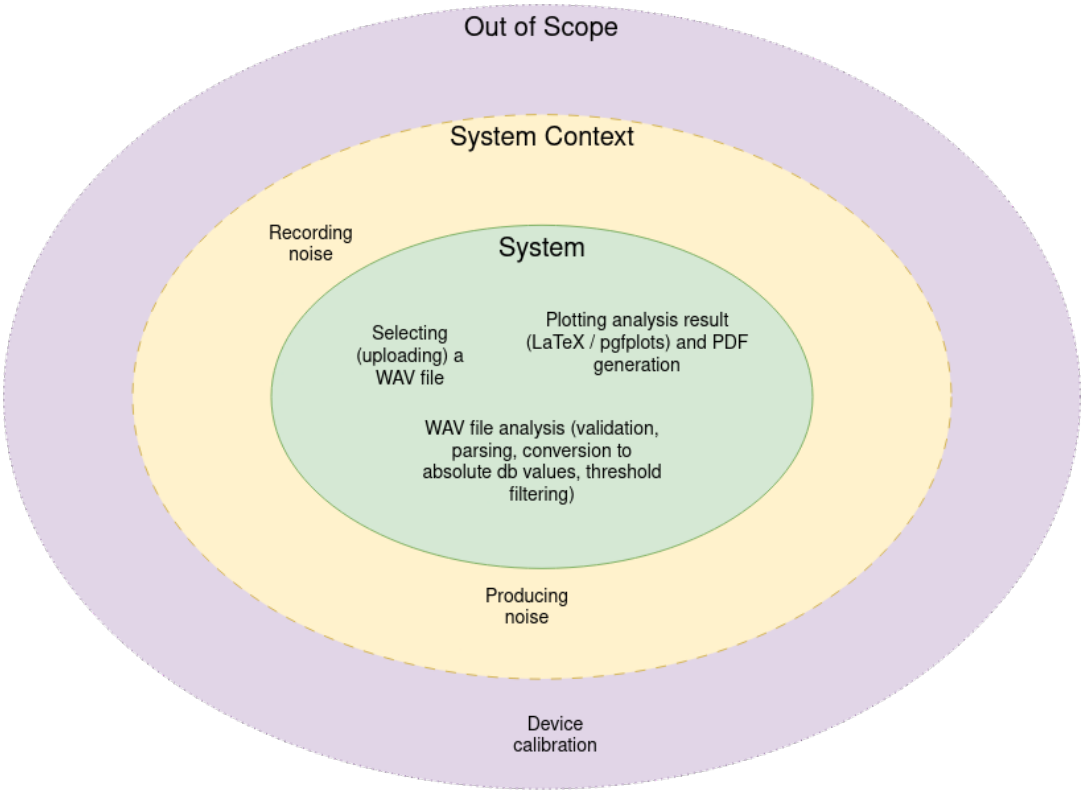

Figure 2: Process Environment

7.2 Requirements

In the following section the requirements are detailed. Also, the project boundaries and preconditions are described.

7.2.1 Functional requirements

The project team identified the following functional requirements:

ID	Requirement	Priority
R1	Allow users to upload a WAV audio file.	MUST
R2	Validate the uploaded file to ensure it is in WAV format, providing an error message if it is not.	MUST
R3	Analyse the uploaded WAV file and calculate the noise level in decibels (dB).	MUST
R4	Allow users to download the plotted noise data as an image or PDF.	MUST
R5	Allow users to input metadata for the audio file, such as location, date, and time.	MUST
R6	Plot the noise level data over time, with the x-axis representing time and the y-axis representing noise level (in dB).	MUST
R7	Generate a PDF report including the plotted noise data and user input metadata.	MUST
R8	Provide clear feedback and error messages.	SHOULD
R9	Intuitive and responsive UI for selecting files, configuring options, and viewing results.	SHOULD
R10	Allow the user to configure custom thresholds for noise levels.	COULD
R11	Allow the user to change the language of the application.	COULD

Table 1: Functional Requirements

7.2.2 Pre-Conditions and Boundaries

Research done by the project team has revealed that reading absolute decibel values from WAV files is more difficult than anticipated. The reason for this is that normal consumer microphones are not calibrated for scientifically accurate measurements [5]. Furthermore, it is trivial for a user to increase the volume of a WAV file using free, easy-to-use software like Audacity [6, 7]. Therefore, the project team has decided on a tentative working hypothesis together with the Stakeholder.

Pre-Conditions:

- The user must use a properly calibrated microphone or use a phone app which has built-in calibrations like Decibel X [8] [9].
- The user must have a way to export their audio recording as a WAV file.
- The user must have access to a web browser capable of running WebAssembly.
- The user must not edit the exported WAV file in any way except to decrease the length of the recording.

Boundaries:

- The application supports only the analysis of pre-calibrated, unedited WAV files.
- The application supports only single-channel WAV files.
- The application does not allow generating a legal complaint or directly integrating its output in a legal complaint.
- The user is presumed to be using properly calibrated equipment. There is no validation done in the application to ensure proper calibration.
- The user is presumed to be honest and to not have edited the WAV file. There is no validation done in the application to ensure the file has not been edited.

7.2.3 Legal requirements

Because we are analyzing audio, privacy is an important point to consider. For analyzing the audio, we will only look at the amplitude of individual samples and take the rms over 300ms. No further analytics will be made. The get the rms, we combine a number of individual samples into a single one, which will then be ploted in the resulting document. Because of the rms we take, it is impossible to reconstruct the original audio from the plot. Additionally, we don't use any library to analyse the audio so we have full control over what's happening.

Furthermore, the application will run solely on the client and the server are only providing the html, css and javascript documents, hence no data is sent back to the server. We also don't safe any information from the user like ip address or the web client he uses. But we will deploy our website with GitHub pages, which means that we don't have any control about the information GitHub saves about the user.

7.3 Usability

In order to best understand the targeted user base, the project team identified personas (7.3.1) and came up with a storyboard (7.3.2). Using this specification, the project team created prototypes for the user interface as well as the PDF report generated by the application (7.3.4).

7.3.1 Personas

Noise is a problem almost every demographic can be affected by. Nonetheless, the project team has attempted to capture the most common attributes of people likely to use the application.

Name	Daniel
Age	42
Sex	Male
Occupation	Business consultant
Marital Status	Married & has two young children
Lifestyle	Daniel usually has to get up early in the morning because his customers are spread across the entire German-speaking part of Switzerland, and they expect him to meet them in their offices.
Goals	Daniel wants to expand his client base in hopes of possibly quitting at his workplace and founding his own consulting firm.
Frustrations	Daniel needs a lot of sleep which has been difficult ever since his first child was born. Much worse however, is that his new upstairs neighbours are up late every night, playing loud music, stomping across the floor, or arguing loudly. Daniel has already asked them to be more considerate but to no avail.
Expectations	Daniel expects to find a way to prove to the police that his neighbours are consistently breaking the law and interfering with his and his families' lives. He expects this solution to be free, easy to use, and reliable.

Table 2: Persona 1 (Daniel)

Name	Julia
Age	25
Sex	Female
Occupation	Student
Marital Status	Single
Lifestyle	During the day Julia is mostly attending lectures at university, running errands, or going after one of her hobbies. In the evenings she studies, often long into the night.
Goals	Julia wants to land a scholarship at a prestigious university, meaning she has to be top of her class.
Frustrations	Julia's apartment has windows facing a busy street, and she regularly faces difficulties focussing on her studies due to the noise. She is convinced her landlord has neglected to install the noise isolation required by law.
Expectations	Julia is looking for a way to make her landlord install better noise isolation to the apartment complex so she and her neighbours can spend their evenings without being bothered by the noise of passing cars and trucks.

Table 3: Persona 2 (Julia)

7.3.2 Storyboard

Lorem ipsum

7.3.3 Process Model

The project team has created the following process model 3, which is a high level illustration of how the application should work.

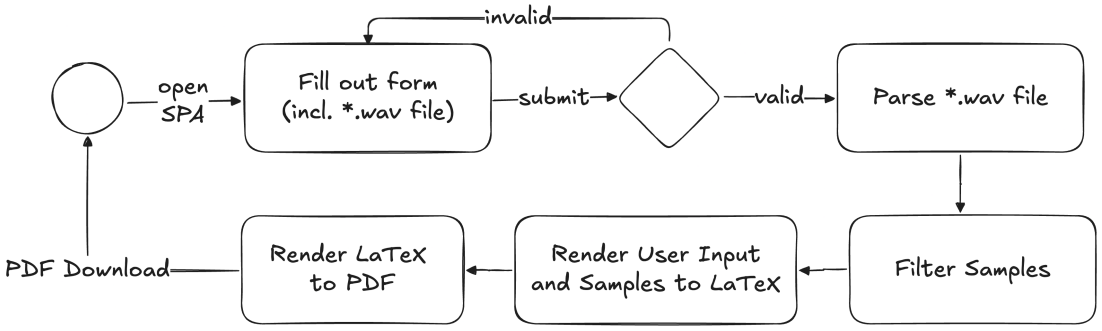

Figure 3: Process Model

7.3.4 UX-Prototyping

Based on our the previous specification, especially the functional requirements 7.2.1 and the process model 3, the project team identified two main UX components of the application and created their respective UX-Prototypes:

- UX Prototype – PDF Report 4
- UX Prototype – Website 5, 6

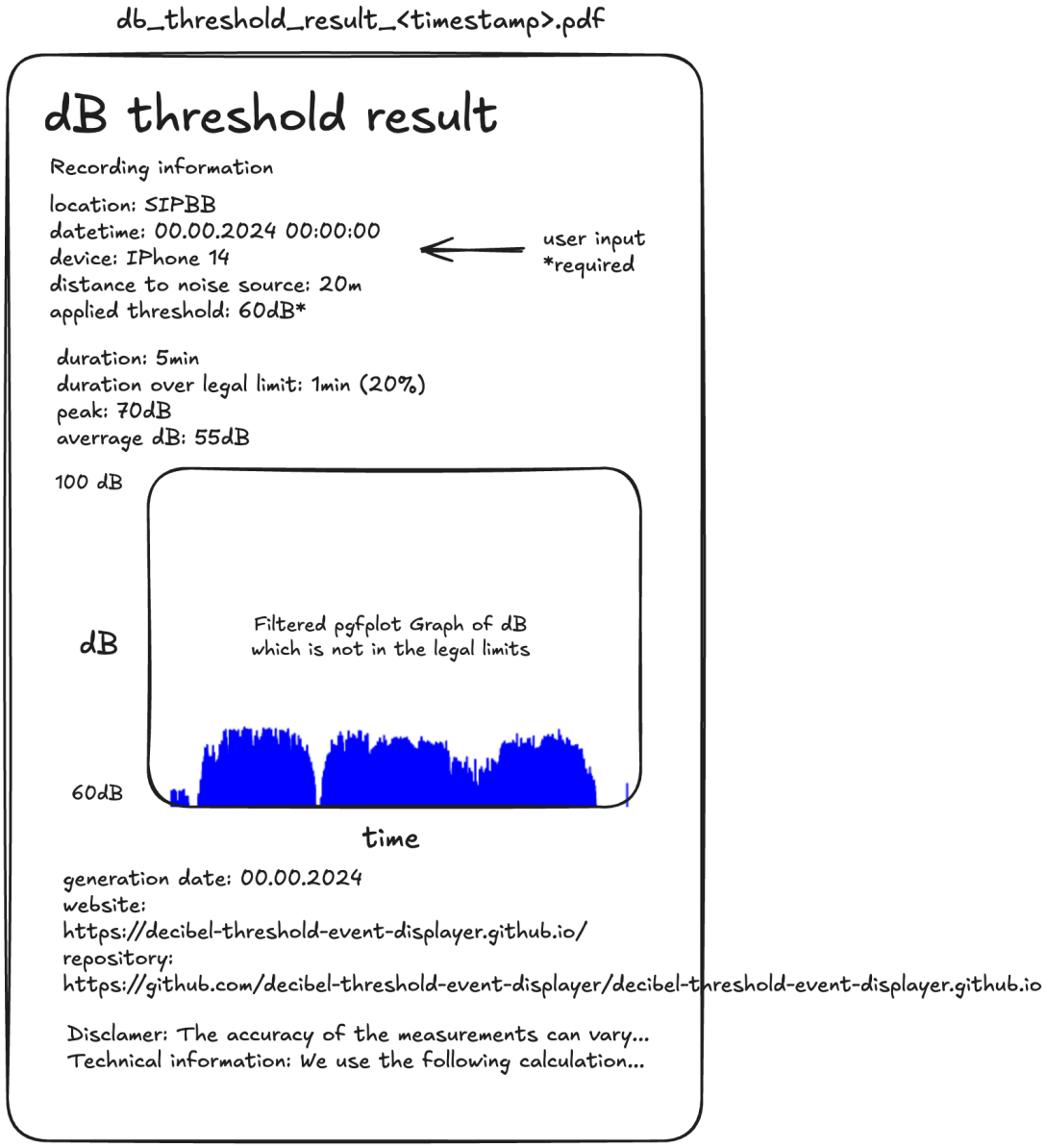

Figure 4: UX Prototype – PDF Report

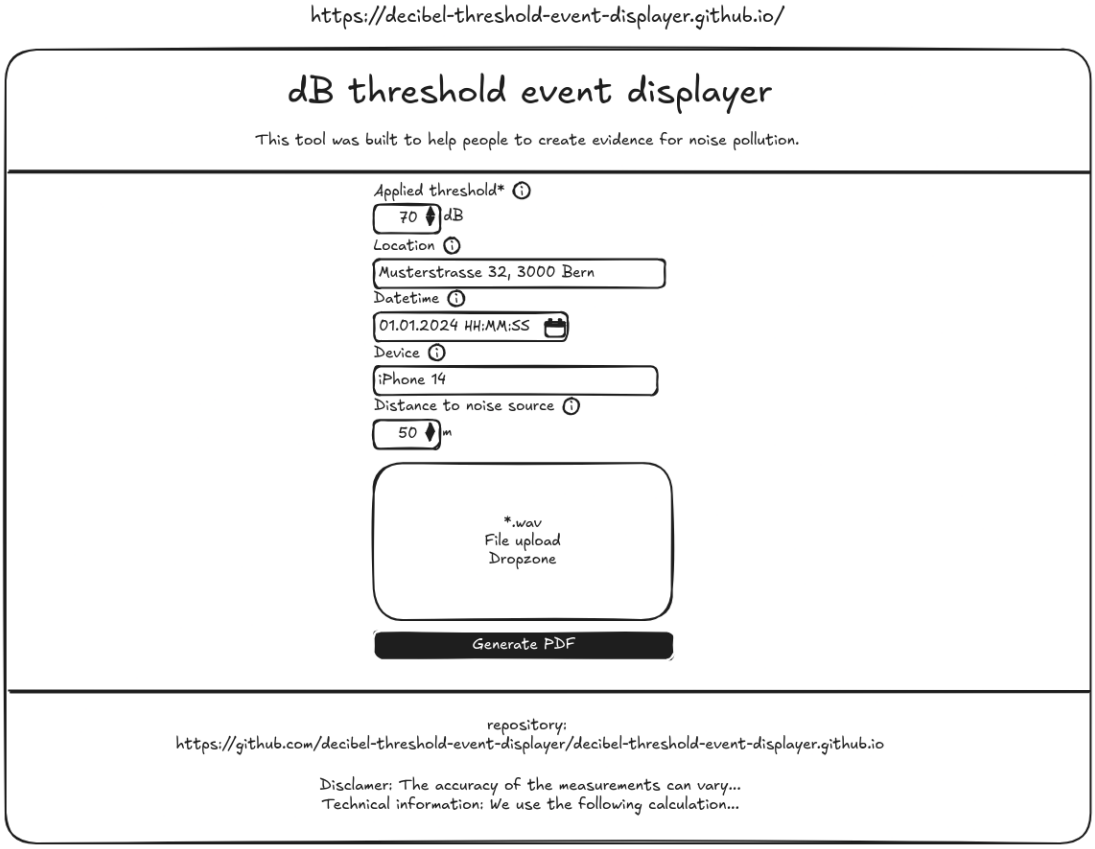

Figure 5: UX Prototype – Website

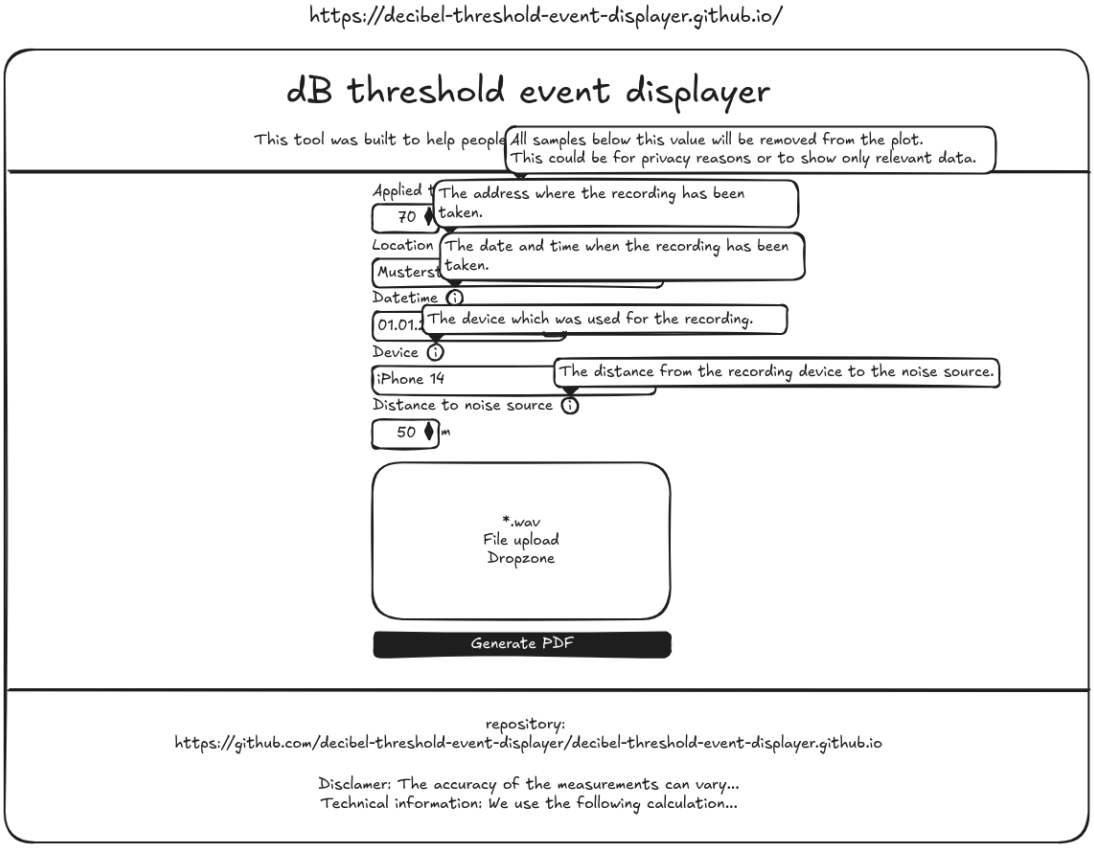

Figure 6: UX Prototype – Website with open tooltips

8 Evaluation

8.1 Technology stack

Since the project description does not mention specific technologies to be used except for the LaTeX package pgfplots, the project team decided to evaluate different technologies for the project by creating working prototypes. The following chapters describe the evaluation criteria and the results of the evaluation.

8.1.1 Criteria

The following criteria are used to decide which technologies should be used for the project:

- Know-how (10%)
- Complexity (20%)
- Usability (30%)
- Distribution (40%)

8.1.2 Calculation

The formula for calculating the weighted value of each criterion is:

$$v_w = \text{weight} \cdot \frac{\text{evaluation}}{5}$$

Where *evaluation* is the scoring between 1–5. The total score for a given technology is the sum of weighted values.

8.1.3 Technologies

The following technologies were evaluated:

- Kotlin with pdflatex as a dependency
- Kotlin bundled with pdflatex
- Website with SwiftLaTeX

8.1.4 Reference LaTeX pgfplots template

For building the prototypes we will use the following reference LaTeX template which includes a pgfplots object: [10]

```
\documentclass{article}

\usepackage{siunitx}
\usepackage{tikz} % To generate the plot from csv
\usepackage{pgfplots}

\pgfplotsset{compat=newest} % Allows to place the legend below plot
\usepgfplotslibrary{units} % Allows to enter the units nicely

\sisetup{
  round-mode          = places,
  round-precision     = 2,
}

\begin{document}

\begin{figure}[h!]
  \begin{center}
    \begin{tikzpicture}
        \begin{axis}[
            xmin=-3,   xmax=3,
            ymin=-3,   ymax=3,
            extra x ticks={-1,1},
            extra y ticks={-2,2},
            extra tick style={grid=major},
        ]
            \draw[red] \pgfextra{
              \pgfpathellipse{\pgfplotspointaxisxy{0}{0}}
                {\pgfplotspointaxisdirectionxy{1}{0}}
                {\pgfplotspointaxisdirectionxy{0}{2}}
              % see also the documentation of
              % 'axis direction cs' which
              % allows a simpler way to draw this ellipse
            };
            \draw[blue] \pgfextra{
              \pgfpathellipse{\pgfplotspointaxisxy{0}{0}}
                {\pgfplotspointaxisdirectionxy{1}{1}}
                {\pgfplotspointaxisdirectionxy{0}{2}}
            };
            \addplot [only marks,mark=*] coordinates { (0,0) };
        \end{axis}
        \end{tikzpicture}
    \caption{My first autogenerated plot.}
  \end{center}
\end{figure}

\end{document}
```

Listing 1: Reference LaTeX template with a pgfplots object

8.1.5 Kotlin minimal – Kotlin with pdflatex as a dependency

This prototype uses Kotlin to satisfy platform-independence and invokes the pdflatex binary which is included in the TexLive distribution. The project team chose to evaluate this technology because it satisfies the base requirements according to the project description and because the project team is familiar with Kotlin. Another advantage of this technology is that the distribution of the necessary LaTeX installation and packages is relegated to the user. This greatly reduces distribution complexity.
An obvious disadvantage of this approach is that the user needs to install TeXLive (and the pgfplots package) on their system, which might be difficult for non-technical users.

8.1.5.1 Licensing

This prototype is dependent on both TeXLive and pgfplots. TeXLive uses a libre license that allows for the free redistribution with or without modifications [11] in accordance with the Free Software Foundation's free software definition [12]. The pgfplots package uses the GPLv3 license [13].

8.1.5.2 Evaluation

Criterion	**Evaluation (1-5)**	**Weighted value**
Know-how	4	8
Complexity	5	20
Usability	1	6
Distribution	5	40
Total	n.a.	**74**

Table 4: Evaluation of Kotlin with pdflatex as a dependency

8.1.6 Kotlin bundled – Kotlin bundled with pdflatex

Similarly to the prototype in Kotlin minimal 8.1.5, this prototype uses Kotlin and invokes the pdflatex binary. The major difference is that the pdflatex binary is provided alongside the software. The advantage of this approach is that the user does not have to install TexLive or any dependencies, greatly improving usability. However, this comes with the disadvantage that the complexity of the distribution is much higher. This approach also limits the advantages of using a platform-independent programming language because for every platform there has to be a release which packages the appropriate binary.

8.1.6.1 Licensing

The licensing is equivalent to the licenses of the prototype in Kotlin minimal 8.1.5.

8.1.6.2 Evaluation

Criterion	Evaluation (1-5)	Weighted value
Know-how	3	6
Complexity	3	12
Usability	5	30
Distribution	1	8
Total	n.a.	**56**

Table 5: Evaluation of Kotlin bundled with pdflatex

8.1.7 Web with SwiftLaTeX

During the projects initial discussions, which had brainstorm like character, we wondered if we could render latex files directly in the browser, which would have the following benefits:

- Providing the functionality as a web application would be probably the easiest and most accessible way for an end user to use the tool.
- Introducing the additional constraint of running the application only on the client side, would also avoid any infrastructure service and maintenance cost.

After a quick search we found the following open source JavaScript/WebWebassembly project: SwiftLaTeX: WYSIWYG LaTeX Editor for Browsers According to their website, the JavaScript/WebWebassembly library has the following characteristics [14]:

- 100% Browser – PdfTeX and XeTeX written in 100% WebAssembly and run in browsers.
- Compatibility – Produce exact same output you would get from TexLive or MikTeX.
- Library Support – Simply include a script tag and use PdfTeX or XeTeX in your own webpage.
- WYSIWYG – Support WYSIWYG editing on LaTeX documents using XeTeX engine.
- Speed – Run merely 2X slower than native binaries.
- Open Source – Completely Open Source. You can find the code on GitHub.

If the statements on the SwiftLaTeX website hold true, it would neatly full-fill our requirements for building a client side only web application.

8.1.7.1 Verifying basic functionality

To make a first feasibility test we can insert our reference LaTeX pgfplots template 1 into the SwiftLaTeX Demo tool. Even though it took more than one and a half minute to process, we were very surprised that this just worked, see figure 7:

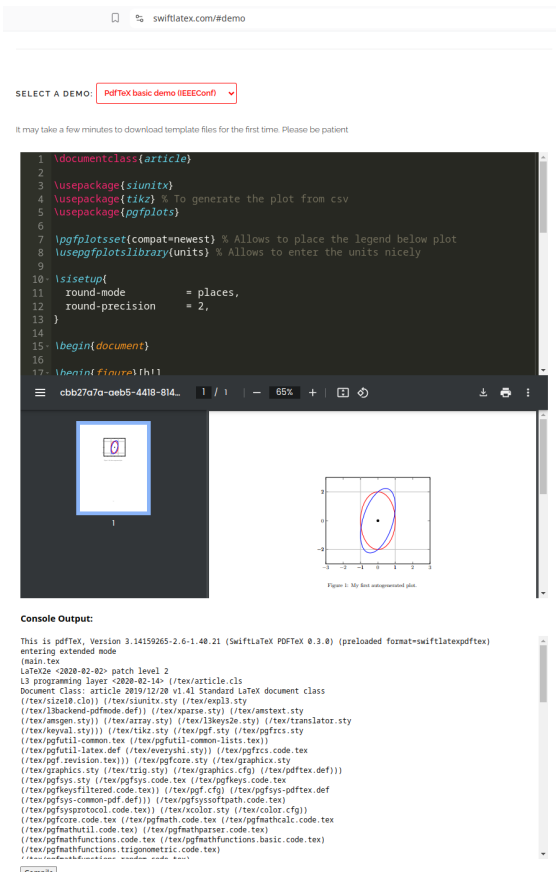

Figure 7: Screenshot of the SwiftLaTeX Demo tool rendering the reference LaTeX pgfplots template

8.1.7.2 Investigating dependencies are loaded

After investigating what actually happens on the page with the browsers developer tools network tab, we saw that SwiftLaTeX will load all dependencies asynchronously via https://texlive2.SwiftLaTeX.com/. This process and how self-host those files, is documented in the README.md of the SwiftLaTeX GitHub library [15]. As we will be able to specify which LaTeX dependencies our application will use, it would be interesting to download and serve only the files we actually need. We also saw that the dependencies are loaded sequentially, which we would like to optimize to reduce the pdf generation time.

8.1.7.3 Local development setup

To make the same example working on a local machine, we can download the latest release from the GitHub repository: Releases 20/02/2022. The release contains the website visible at SwiftLaTeX.com including the compiled WebAssembly files. As expected, opening the index.html file from the unzipped release folder in a browser will look like the website SwiftLaTeX.com. Validating the basic functionality with our reference LaTeX pgfplots template 1 did not work, as we get the following error message in the JavaScript console:

```
1 PdfTeXEngine.js:89 Uncaught (in promise) SecurityError: Failed to construct '
      Worker': Script at 'file:///home/lukas/Downloads/20-02-2022/SwiftLaTeXpdftex.
      js' cannot be accessed from origin 'null'.
2 ...
```

Listing 2: SwiftLaTeX local development setup: JavaScript error message

According to the following stackoverflow question, Google Chrome does not load web workers when running scripts from local files. The web worker is required to run WebAssembly, which does all the actual LaTeX to PDF transformation in this prototype. The easiest solution for this issue is to run a webserver, luckily this is trivial with Python. We just have to run the following command in the directory of the unpacked SwiftLaTeX release folder:

```
python -m http.server
```

Listing 3: Start a webserver on http://localhost:8000/ with Python

Validating the basic functionality with our reference LaTeX pgfplots template 1 on http://localhost:8000/ works now as expected.

8.1.7.4 Distribution option 1: Python webserver

This would be the easiest solution, as it is analog to the local development setup described in the previous paragraph. But installing Python, downloading the git repository or downloading a release and running the Python web server in the correct directory, requires at least a lot of technical affinity.

8.1.7.5 Distribution option 2: PyInstaller

To enhance the end users experience we could build an executable from a Python file with PyInstaller which supports builds for Windows, macOS and Linux. Even though this solution worked in a quick proof of concept on a linux system, the user would need to find the public GitLab repository and download a release, which contains the latest executables. Further the user might have security concerns, as downloading an executable from an unknown or untrusted source is really dangerous.

8.1.7.6 Distribution option 3: GitHub Pages

After thinking about it, we figured out that we could deploy the tool as a static HTML page to GitHub pages, this would resolve the need of downloading any source code or executable program. In our opinion this would result in the best user experience, as the user only needs a web browser to access and use the tool. As this solution was worth exploring, we deployed the prototype with GitHub pages: https://vonal3.github.io/

8.1.7.7 Licensing

- SwiftLaTeX: SwiftLaTeX License, GNU AFFERO GENERAL PUBLIC LICENSE
- LaTeX: The LaTeX project public license
- TeXLive: TeX Live licensing, copying, and redistribution, COPYING CONDITIONS FOR TeX Live
- pgfplots: https://ctan.org/pkg/pgfplots, GNU General Public License, version 3 or newer

8.1.7.8 Evaluation

Criterion	Evaluation (1-5)	Weighted value
Know-how	3	6
Complexity	3	12
Usability	4	24
Distribution	5	40
Total	n.a.	**82**

Table 6: Evaluation of SwiftLaTeX JavaScript/WebAssembly library

8.1.8 Chosen Solution

Using the evaluations given to the selected technologies, the following demonstrates the combined results:

Technology	Total score
Kotlin minimal	74
Kotlin bundled	56
Web SwiftLaTeX	82

Table 7: Technology stack evaluation

In accordance with the table above, the project team decided to use **Web SwiftLaTeX** with the distribution option 3 **GitHub Pages**.

8.2 License

After evaluating the different possible technologies, the following third party software will be used:

- SwiftLaTeX: SwiftLaTeX License, AGPL-3.0
- pgfplots: https://ctan.org/pkg/pgfplotspgfplots, GPL-3.0

To be conform with the used software, we will also publish our code under the GPL-3.0 licence.

9 Implementation

9.1 Architecture

9.1.1 Data Structures and Overview

For conveniently accessing the data contained in the wave file, we created the WaveFileWrapper class. The wrapper is used by the home components to access the relevant data needed to create an instance of the FrameCollection class. A FrameCollection object contains all Frame objects of the audio file and a single instance of the Frame class represent a set of samples covering a time span of 300ms.

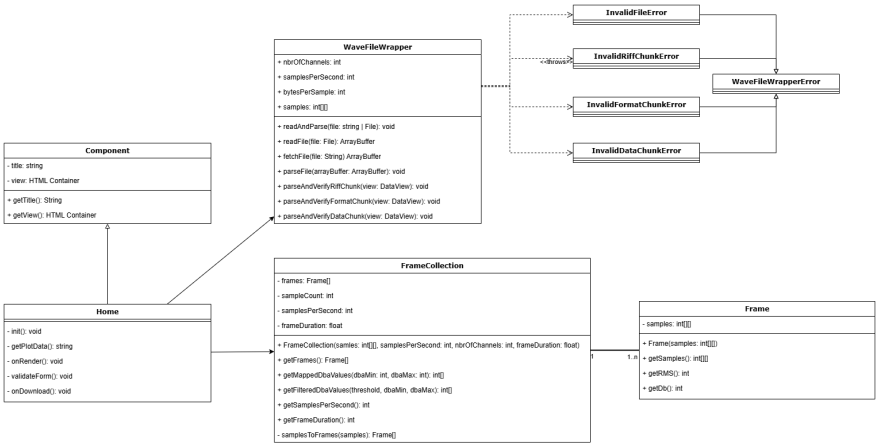

Figure 8: Class diagram

9.1.2 JavaScript Frontend (SPA)

The frontend hosted on Github Pages is implemented as a single page application (SPA) following the architecture provided by the BFH in the module BTI1301 Web Programming. In this architecture, the web server provides the index.html file as and uses JavaScript to manage the web contents (DOM updates) without changing the HTML file itself. Currently, the application only has one route; however, the project team chose to implement a small framework allowing for future expansion without rewriting the entire application. Since the application uses SwiftLaTeX, which is run locally using WebAssembly, the SPA provides an interface for the PDF engine to its components.

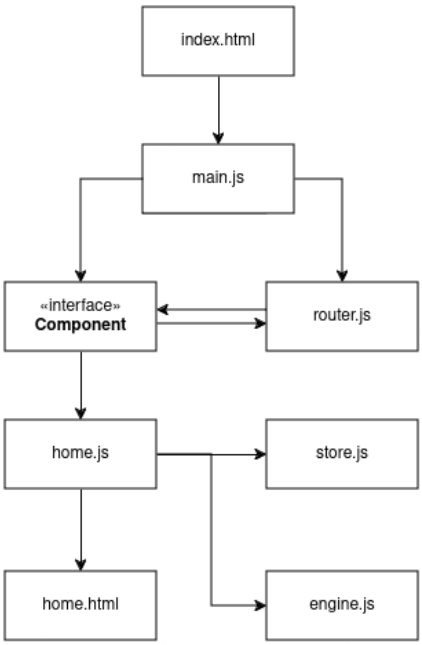

Figure 9: SPA Architecture

In order to ensure a consistent and agreeable look, the project team has decided to use Bootstrap, which is licensed under MIT.

We also decided to define default values for the threshold that the audio should get checked against, for which we have orientated ourselves on the immission limits from the BAFU.

9.1.3 High-Level architecture

!TODO: we should put this either at the beginning or end of this chapter to show what the entire architecture looks like high-level!

9.2 Processes

The following diagram shows which steps are involved when creating a noise report from a wave file.

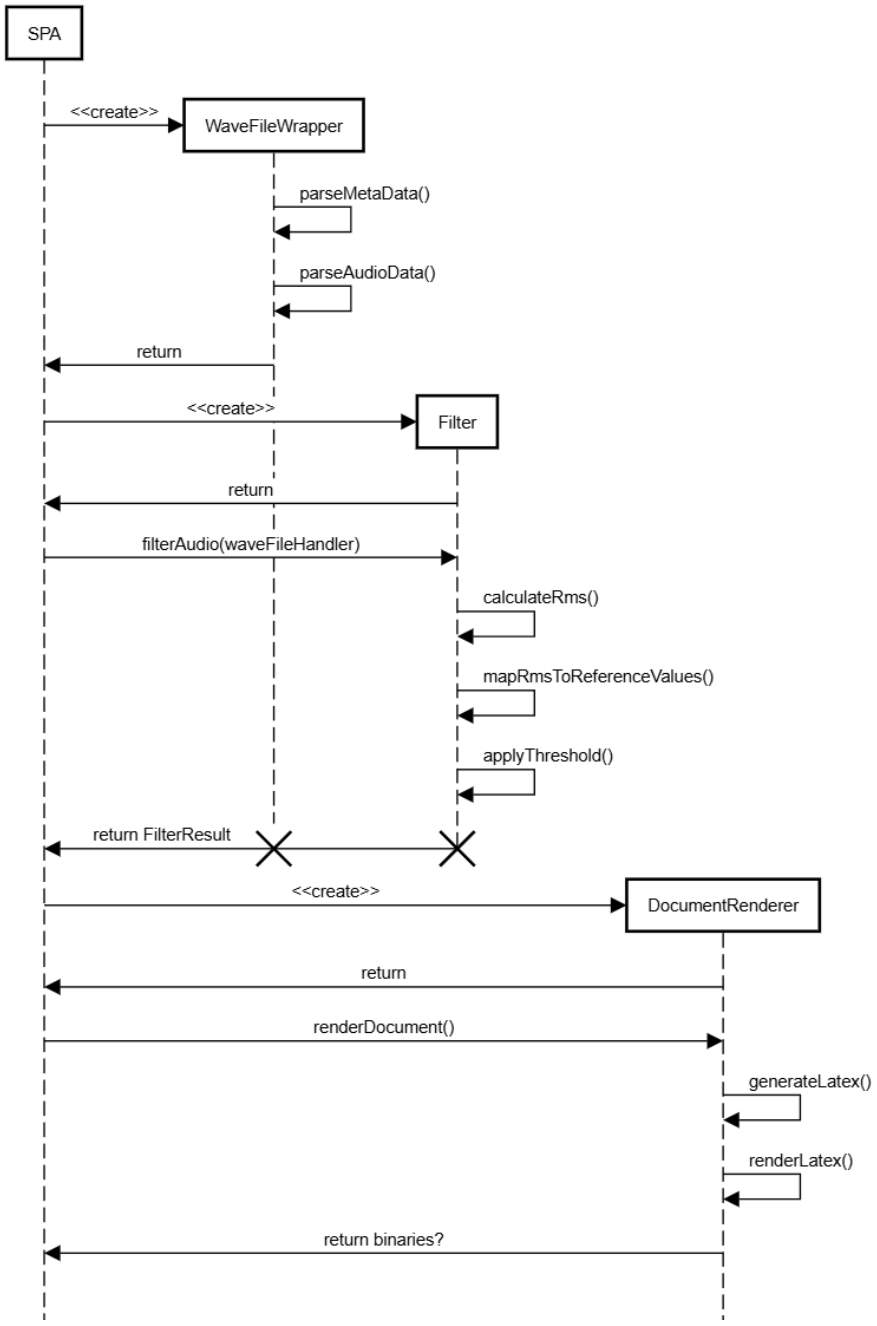

Figure 10: Sequence diagram of steps involved when creating a noise report

9.2.1 Parsing the Wave File

The WaveFileWrapper class will be responsible for reading and parsing the wave file. Reading a file in a web context always works asynchronous, which is why there is a helper function ‘readAndParse‘ which has to be called with async await. The format itself is straight forward [16], the only important thing is that the whole file is Little-Endian. It starts with a 12 byte header which we use to verify that the given file is actually a wave file. After the header comes a 24 byte chunk which contains information about the data format. We are interested in the following fields:

- AudioFormat (2 bytes): we only support PCM and IEEE Float format
- NbrChannels (2 bytes): is used to parse individual samples
- Frequency (4 bytes): number of samples per second, this is needed for filtering and summarising
- BitsPerSample (2 bytes): number of bits for a single sample for a single channel, is used to parse individual samples

After the format chunk can come several optional metadata chunks, which we don't need. Therefore we skip those chunks and go straight to the data chunk. This chunk starts with a 4 byte long DataBlocID, which should always have the value ‘data‘. The next 4 byte represent the number of samples in the data. After that comes the actual audio data. To parse a single sample, we need to parse a frame. A frame contains the data of all channels and has therefore a size of

$$\frac{NbrChannels * BitsPerSample}{8}$$

bytes.

9.2.2 Calculating Decibel Values

If an audio file consists of several channels, we use the average of the channels as the value of the sample. When we want to analyze where a certain decibel threshold has been exceeded, we must have accurate values. People only perceive something as loud if it is loud over a longer period of time. Short peaks are perceived less. Therefore, when determining the effective loudness of audio, we do not use the loudness of individual samples, but rather the average of a number of samples over a certain period of time. Such group of samples over a certain period of time is implemented as the Frame class. We will, as usual when working with audio, use the root-mean-square (RMS), as we are interested in the absolute amplitude values. We implemented the calculation of the root-mean-square as the method ‘getRMS()‘ on the Frame class. The calculated average then depends heavily on how long the time period is. In practice, 300ms is usually used [17]. This is also the default value, which can be adjusted by the user via the settings. The decibel value is calculated with the following formula [18]:

$$20 * \log_{10}(sample)$$

We implemented again a method ‘getDb()‘ on the Frame class, which computes this value.

9.2.2.1 Mapping RMS to absolute dB(A) values

As mentioned in the introduction, the audio file only contains relative values. We therefore need to map the RMS values to the number range of the reference values, where the smallest sample is mapped to the minimal dB(A) and the largest sample is mapped to the maximal dB(A) measured. This calculation does not work in the context of a single frame and therefor is added as the method ‘getMappedDbaValues(dbaMin, dbaMax)‘ on the FrameCollection class.

9.2.2.2 Filter dB(A) values

One of the requirements is to plot the dB(A) values which are higher than a certain threshold. For this reason we implemented the method ‘getFilteredDbaValues(threshold, dbaMin, dbaMax)‘ which returns the final output, which we can plot to a PDF file.

9.2.3 Performance improvements

Generating the PDF file from a LaTeX document in the browser takes quite some time, as it loads all dependencies on the fly. To speed up this process, we serve all required LaTeX files ourselves and prefetch them in parallel. This means we have to download all LaTeX files from a public source, we are using the same source as SwiftLaTeX: texlive2.swiftlatex.com

We collected all required LaTeX dependencies by modifying the SwiftLaTeX JavaScript, in such a way that during a sample all requests pdf generation, are stored in an array, which is then printed to the console. We put the console output in the file "app/dependencies.json" and created a python script under "app/download_dependencies.py", which downloads all LaTeX files to the directory "app/dependencies/". We then copied the contents of "app/dependencies.json" to "app/dependencies.local.json" and made the paths relative. Now we load this file in our main JavaScript file "app/app.js" and use "Promise.all" together with "fetch" to load all dependencies into the cache of the browser.

Currently, this process involves some manual steps and could be automated in a later step.

9.3 Testing

Because we don't use any Frontend Framework and instead build our own SPA, we don't have a testing framework to use. We therefore decides to build our own rudimentary testing framework. It consists out of 4 assert functions, a test runner and a html page for displaying the results. There is no automatic mechanism running the tests after every commit, the tests have to be run manually.

As we are developing an application for analyzing a wave file, nearly every tests first step is to create a WaveFileWrapper object for further use. Because the reading and parsing of the file works asynchronous, the test suit has also to be asynchronous. We decided to use the async/wait approach instead of using promises because promises are more difficult to debug, which is not suitable for a test framework.

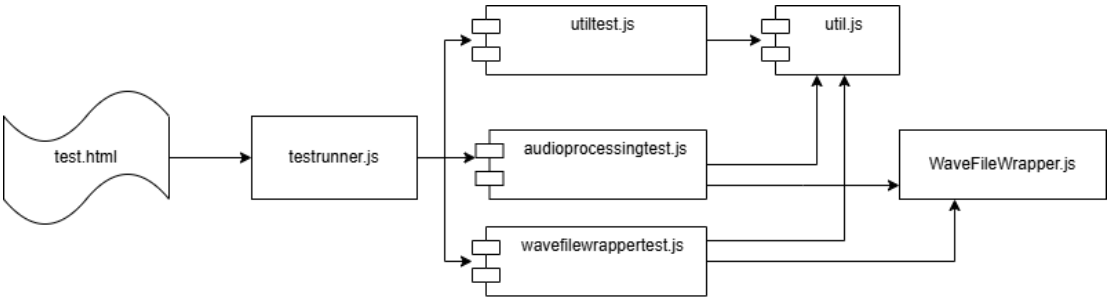

Figure 11: Overview test framework

9.3.1 Assert functions

For implementing tests there are 5 assert functions contained in the util module which can be used:

- assertThrows: verifies that a certain exception is thrown
- assertNotThrows: verifies that no exception is thrown
- assertEquals: verifies that something is equal to a given value
- assertNotEquals: verifies that something is not equal to a given value
- assertGreaterThen: verifies that a number is greater the other

If the assertion of a function fails, it throws an exception with some information on why the test failed. The assertEquals and assertNotEquals function work with any kind of object. They

first stringify both objects to a json format and then compare the json. It is certainly not the most efficient method, but it works for all cases. The assertThrows and assertNotThrows both take in a function which will be called in the assert function itself and verified that it throws or not throws an exception. The assertGreaterThen can also take in two arguments from any type, as javascript allows such comparisons. With those functions we were able to implement all our tests. To make sure that the assert functions work, we implemented also tests for them. Those test obviously don't use the assert functions and rather have some hard coded tests created for them. Those tests can be found in the utiltest module.

9.3.2 Creating tests

To create tests, one has to create a javascript file containing its tests. All functions that are being exported by the given javascript file are considered to be tests and will be executed by the test runner. If a function throws an exception, it is assumed that the test failed. If no exception is thrown, the test passes. We tried to keep the manual work for adding tests minimal, but some manual work is still necessary. To add a new test module, the module has to be imported into the test runner file. After that, a new call to the function runTestGroup can be added, which will then add the test. That's it.

```
import * as waveFileWrapperTest from './wavefilewrappertest.js';
import * as utilTest from './utiltest.js';
```

Figure 12: Importing a module with tests

```
// run all test groups
function runTests() {
    runTestGroup(utilTest, "Util");
    runTestGroup(waveFileWrapperTest, "Wave File Wrapper");
}
```

Figure 13: Adding a method call to run a test module

9.3.3 Test runner

The test runner is responsible for running the tests. It runs every test module in sequence and summarises its results. For each module a table in the result page is created which contains the test names, result and in the case of a failed test a message why it failed. The tests are run when the dedicated button in the frontend is pressed. The test runner always runs every test of every test module. There is currently no function to run individual modules or tests. The test runner also has a function to just test the WaveFileWrapper. It creates such a WaveFileWrapper object from the file specified on the result page and displays the values of its properies.

9.3.4 HTML page

The fronted page of the test framework is used to start the tests and view the results. It is additionally used to create a WaveFileWrapper object and view the properties of it. The test page is accessible under /js/test/test.html. To run the test page locally, see chapter 10.2.1.

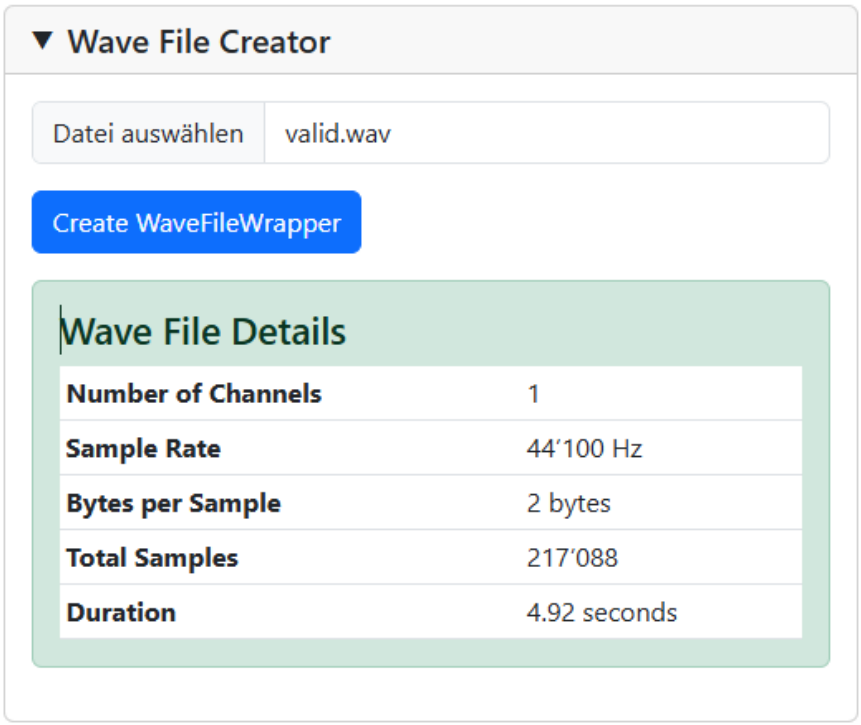

Figure 14: Create a WaveFileWrapper object and view its properties

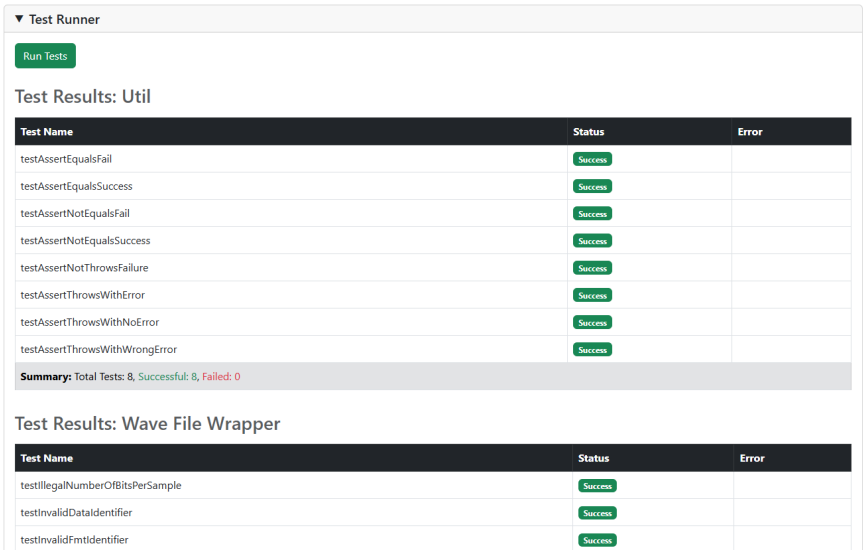

Figure 15: Test results

10 Deployment/Integration

We decided to use GitHub Pages via an organization for distribution because it provides an easy way to access the application for end users and simplifies the maintenance for developers. The GitHub organization is located under: https://github.com/decibel-threshold-event-displayer The application is deployed as a GitHub Page under: https://decibel-threshold-event-displayer.github.io/

10.1 Licensing

Lorem ipsum

10.2 Installation Manuel & Script

We are using a Makefile to automate the development setup, build and deploy the application to GitHub Pages.

10.2.1 Development setup

A local webserver is needed work on the application as some Browsers will not execute web workers on locally served files [19]. By default, the project uses the builtin webserver of Python 3. Python 3 can be installed with the systems package manager or by downloading it from the official website: https://www.python.org/downloads/ Afterward the following command can be used to run the local webserver on http://0.0.0.0:8000/:

```
$ make dev
```

Listing 4: Makefile: Start local webserver

10.2.2 Distribution

The project and the repository is managed via the GitLab of BFH, but we want to use GitHub pages to deploy and distribute our application, the project team decided that we mirror the GitLab repository to GitHub:

- GitLab Repository
- GitHub Repository

10.2.2.1 Mirror setup:

1. Log into gitlab.ti.bfh.ch
2. Navigate to the repository: GitLab Repository
3. Go to the repositories settings page: Settings > Repository
4. Open the Tab “Mirroring repositories“
5. Click the button “Add new“
6. Fill the form as follows:
    - Git repository URL: ssh://git@github.com/decibel-threshold-event-displayer/decibel-threshold-event-displayer.github.io.git
    - Mirror direction: Pull
    - Authentication method: SSH public key
    - Username: github
    - Mirror user: autofilled
    - Overwrite diverged branches: Select
    - Mirror branches > Mirror specific branches: main
7. Click the button “Mirror repository“
8. The first try will fail, as we have to add the public key generated by GitLab to the GitHub repository
9. On the newly created entry, click the clipboard button “Copy SSH public key“ (the public key is now in your clipboard)

10. Keep the current GitLab tab open
11. Log into github.io in a new tab
12. Navigate to the repository: GitHub Repository
13. Go to the repositories deploy keys page: Settings > Deploy Keys
14. Click the button “Add deploy key“
15. Fill the form as follows:
    - Title: github
    - Key: Copy the SSH public key from the clipboard
    - Allow write access:
16. Click the button “Add Key“
17. Go back to the gitlab tab
18. Click the reload button “Update now“
19. Wait until the process is finished

10.2.3 Build

As we have only plain JavaScript files, we don’t really have a build step.

10.2.4 Deploy

As described above, we created a Mirror of this repository on GitHub. Based on the following documentation, we configured GitHub to automatically deploy to GitHub Pages: Creating a GitHub Pages site

Further we had to add a configuration file under “.github/workflows/static.yml“ and change value of “path: ./“ to “path: ./app“. The initial file was generated by the following setup page: Repository > Settings > GitHub Pages > Static HTML > Configure

10.3 User Manuel (hopfully not)

Lorem ipsum

11 Conclusion

Lorem ipsum

11.1 Review

Lorem ipsum

11.2 Bottom Line

Lorem ipsum

11.3 Future Work

Lorem ipsum

12 Glossary

- dB(A): a decibel value with a scale relativ to the human ear.
- RMS: root mean square

13 Index

Lorem ipsum

14 Bibliography

References

[1] Federal Office for the Environment FOEN. Noise and vibrations: In brief. https://www.bafu.admin.ch/bafu/en/home/topics/noise/in-brief.html. Accessed: 2024-11-4.

[2] how to know the real-world db level of a file? https://community.adobe.com/t5/audition-discussions/how-to-know-the-real-world-db-level-of-a-file/m-p/13939424. Accessed: 2024-11-4.

[3] How can i calculate audio db level? https://stackoverflow.com/questions/2445756/how-can-i-calculate-audio-db-level. Accessed: 2024-11-4.

[4] Extracting sound pressure from wav file. https://dsp.stackexchange.com/questions/35250/extracting-sound-pressure-from-wav-file. Accessed: 2024-11-4.

[5] user545125. Answer to "how can i calculate audio db level?". https://stackoverflow.com/a/4464385. Accessed: 2024-11-02.

[6] Audacity Team. Audacity. https://www.audacityteam.org. Accessed: 2024-11-02.

[7] Audacity Team. Audacity amplify. https://manual.audacityteam.org/man/amplify.html. Accessed: 2024-11-02.

[8] Decibel X (iOS). https://apps.apple.com/de/app/decibel-x-db-sound-level-meter/id448155923?l=en-GB. Accessed: 2024-11-02.

[9] Decibel X (Android). https://play.google.com/store/apps/details?id=com.skypaw.decibel&hl=en&pli=1. Accessed: 2024-11-02.

[10] Pgfplots gallery. https://pgfplots.sourceforge.net/gallery.html. Accessed: 2024-10-22.

[11] TeX Users Group. TeX Live license. https://tug.org/texlive/LICENSE.TL. Accessed: 2024-10-22.

[12] Free Software Foundation (FSF). FSF Free Software definition. https://www.gnu.org/philosophy/free-sw.html. Accessed: 2024-10-22.

[13] Christian Feuersänger. pgfplots. https://ctan.org/pkg/pgfplots. Accessed: 2024-10-22.

[14] Latex.wasm: Latex engines in browsers. https://www.swiftlatex.com/. Accessed: 2024-10-22.

[15] Swiftlatex github repository. https://github.com/SwiftLaTeX/SwiftLaTeX/. Accessed: 2024-10-22.

[16] Wav file format. https://en.wikipedia.org/wiki/WAV. Accessed: 2024-11-5.

[17] Timespan for calculating audio rms. https://majormixing.com/what-is-rms-in-audio-world/. Accessed: 2024-11-5.

[18] Decibel wikipedia. https://en.wikipedia.org/wiki/Decibel. Accessed: 2024-11-5.

[19] Chrome can't load web worker. https://stackoverflow.com/questions/21408510/chrome-cant-load-web-worker. Accessed: 2024-10-28.

15 Appendix

15.1 Project description

The **goal (what)** of this project is to deliver a FLOSS-licensed, platform-independent piece of software (computer program), called the *Decibel Threshold Event Displayer*, that

1. takes as inputs a WAV-file and a list of sound level thresholds in decibels (e.g., legal day and nighttime noise maxima above which your health deteriorates);
2. filters out all data points in the file that correspond to sound events below the lowest of the above thresholds; and
3. displays the remaining data points (as a blue vertical comb plot) on a horizontal time axis (with the dates and times corresponding to the data points) as well as the thresholds (as horizontal red lines) in decibel, and statistically summarises the data set with the help of the LaTeX-package pgfplots.

The **purpose (why)** of this project is to empower poor folks who suffer from insomnia due to ambient noise (https://laermliga.ch/) by arming them with the (peaceful) means of proving their noise hell (a smart phone app such as https://apps.apple.com/ch/app/dezibel-x-pro-l%C3%A4rm-messger%C3%A4t/id1257651611 together with your software) to the police and the courts of law.

The code should be minimal, modular, and self-explaining.

The project report should be concise (maximally informative, minimally long). It must contain this project description as a quotation.

15.2 Declaration of Authorship

The project team, namely Dominic Gernert, Lukas von Allmen, and Darius Degel, hereby declare that the report submitted is our own unaided work. All direct or indirect sources used are acknowledged as references.