Master thesis on Sound and Music Computing

Universitat Pompeu Fabra

Online Audio Editor in Freesound

Roberto Pérez Sánchez

Supervisor: Frederic Font

August 2020

Table of contents

Table of figures

Dedication

I dedicate this master thesis to my parents, Cristina and Marcos, for always helping and supporting me whenever I had to take an important decision in my personal and academic life. Thank you for letting me study the topics I have always wanted and for giving me the opportunity to go and live this course in Barcelona, where I have met loads of new people that will always be part of me as a person. You have always encouraged me to take my own decisions, and I could not have been in the position I am now without your support. I love you!

Acknowledgements

I cannot express enough thanks to my supervisor of this thesis, Frederic Font, for the continuous support and advice he has given me during its development. He has always been honest with me and my work, telling me his opinions about my project and always giving me the appropriate advice for each of the problems I have encountered.

I would also want to thank my SMC Master classmates for the time spent together in Barcelona this year. It has been short, but I am sure we will meet again in the near future.

Finally, I would like to say thank you to all the teachers in charge of the Sound and Music Computing master for always trying to make classes really interesting and easy-to-follow and for all the knowledge that we have learned during these months, and to the whole institution of the Universidad Pompeu Fabra for giving me the opportunity of studying in one of the best universities of Spain.

Abstract

Freesound has a huge community dedicated to upload and share sounds online for other users to use those sounds in their productions. In some cases, the sounds will need to be transformed and users will have to go in their devices and edit that sound in order to get the exact version that fits their production the most. In this thesis, an audio editor is developed and added into the Freesoud environment in order to have those two steps together in the same workflow. This approach will add a huge value to the Freesound experience and will differentiate it from its competitors. For the development of this editor, different Javascipt and HTML frameworks have been used, like Wavesurfer (based on Web Audio API) or Bootstrap, but always trying to keep the interface and functionalities accessible and easy-to-use to a general audience. Finally, two evaluation methods have been developed, the first in order to get some general feedback about the use of the editor and the second one to be sure that all its different functionalities behave as expected. The first evaluation showed the need of a simpler interface and functionalities, which was developed and then tested using the second evaluation approach, showing that all different actions, filters and effects worked as expected.

1. Introduction

Looking at the numbers presented by Font in [12], the number of sound downloads in Freesound during 2019 was around 20.6 million. That is 1.6M more than the previous year. Having this huge number of downloads, which is continuously increasing every year, the website would benefit from a tool to let users take the most advantage of the sound they want to download by selecting only one of its parts or applying some effects and filters to them, simplifying their workflow by not having to use another platform like Audacity to edit the sample. In this thesis, an online embedded editor in the Freesound web will be developed and tested in order to see if this tool will add some extra value to the community experience while using the system.

First of all, I will present an overview of the state of the art regarding Freesound, the existing stand-alone editors in the market, different usages and frameworks of the Web Audio API and some evaluation techniques commonly used for testing these types of projects. Then, the methodology followed for the development of the editor will be explained, taking into account the planning (divided in sprints), the implementation of the code and the evaluation methods used (usability questionnaire and user cases). Moreover, the different versions of the editor during its development and evaluation will be presented with screenshots of the interface as well as an explanation of the problems that I have encountered during the whole process. Finally, some conclusions will be outlined to sum up how Freesound will benefit from the addition of this editor, as well as some possible lines of future work regarding its development.

2. State of the Art

2.1. Introduction

Since its creation in 2005, Freesound[1] has developed a huge user community and sound database. In 2012, as stated in [3], there was more than 2 million registered users in the platform and more than 140,000 uploaded sounds. Nevertheless, Font et al. also noticed that 89% of these users are considered as ‘regular’ users, which are the ones that have registered to the site but never contributed to it, maybe just registering in order to

[1] https://freesound.org/

download some samples, and less of 1% of the total users are the ones that upload sounds. Although this gives the impression that there are not many features that engage the user to participate actively in the system, either uploading sound or participating in forums, the reason for that low percentage of contributors could be explained because it requires the user to have some related skills and (possibly) gear. Adding these new features, for example an embedded editor in the web, might make the platform more attractive both for contributors and for consumers

Nowadays, Freesound counts with more than 8 million registered users and more than 400,000 sounds and effects available to be downloaded. This growth of users and sounds have led to the development of new features in the main website and subprojects, for example, Freesound Datasets (later renamed as Freesound Annotator). Fonseca and his colleagues explain in [6] the use of Freesound samples in order to create open audio datasets based on four principles: transparency, openness, dynamism and sustainability. Regarding this new features or subprojects related to the platform, a new "idea" appeared: an audio editor embedded in the website in order to edit the sample prior to its download. This new feature will give the user the opportunity to perform quick fixes of the sample, as well as applying different effects and filters to it without having to execute a different program just for editing the sound. From here, it derives the research question of this thesis: **How an easy-to-use, maintainable, and efficient audio editor embedded in the Freesound website could be developed?**

The goal of this thesis is to implement such audio editor. In order to develop and insert the editor into the Freesound website, the Web Audio API, a modern and powerful audio framework for the browser [4], will be used. The API developed by W3C describes a high-level JavaScript API for processing and synthesizing audio in web applications and its supported by the main web browsers (Google Chrome, Mozilla, Opera, Safari and Edge). But, historically, this API was not the first approach in order to work with audio in the web. As Boris Smus explains in his book 'Web Audio API' [2], the first way of playing sounds on the web was using the <bgsound> tag in HTML, but it was very limited as it only allowed to play background audio in the browser. More recently, the <audio> tag was the one used in order to perform these tasks until the creation of the Web Audio API in late 2011, which was created in order to give a better performance of those tasks, as well as giving them a bit more flexibility, bringing a fully featured audio framework to the browser.

The Web Audio API is based and built around the concept of an Audio Context, which is a directed graph of audio nodes that defines how the audio flows from its origin (often an audio file) to its destination (often your speakers). The different nodes between the source and the destination are known as modification or analysis nodes. They are used in order to add some filters or process some feature of the sound, for example. In *Figure 1* we can see the simplest connection of the audio context, from destination to the source.

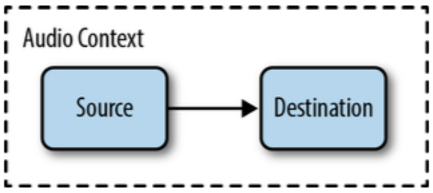


Figure 1. The simplest audio context [2]

Frequently, this API is wrapped into other frameworks to make it easier and intuitive to work with it. Some frameworks studied for the development of this project are the following:

- Wavesurfer.js[2]: Customizable audio waveform visualization JavaScript framework.
- Pizzicato.js[3]: Web audio JavaScript library that aims to simplify the way you create and manipulate sounds via the Web Audio API.
- Tone.js[4]: Framework for creating interactive music in the browser. It provides advanced scheduling capabilities, synths and effects, and intuitive music abstractions built on top of the Web Audio API. [9]

Furthermore, and as every other web-based API or framework, the Web Audio API also has its limitations. Because of this, many developers and researchers have developed several extensions for this API. For example, in 2013, Choi and Berger presented in [1] WAAX, an extension that was created 'to facilitate music and audio programming based on Web Audio API bypassing underlaying tasks and augmenting useful features. In their paper, they explain how this extension can speed up the development and the additional features developed on top of the Web Audio API, as well as discuss future improvements.

[2] https://wavesurfer-js.org/

[3] https://alemangui.github.io/pizzicato/

[4] https://tonejs.github.io/

WAAX provides the API with a live 2D and 3D visualizer and an interactive code editor, but its main advantage is the wrapping of the original API nodes into ‘units’, which are abstractions of the audio graph with additional features for a specific purpose.

Taking all of this into account, the development of an easy-to-use, maintainable and efficient web audio editor should be very feasible. On top of that, it would add a huge amount of value to the Freesound webpage, because, to the best of my knowledge, similar websites like FindSounds[5] or FreeMusicArchive[6] do not feature an audio editor like the one its being envisioned in this thesis.

2.2. Existing editors

But prior to this development, some further research has to be done in order to familiarize myself with the usual functionalities used in different stand-alone audio editors and digital audio workstations (DAWs). Some of the most relevant features that an easy-to-use audio editor should give to the user are, according to [7], (1) simple and uncluttered workflow and interface, (2) use of effects in order to alter the sample and (3) support for a wide array of audio formats (WAV, MP3, etc.). Taking this features and some more complex ones into account, [7] has created a list of the best audio editors available in the market. Some of the most relevant ones for the purpose of this thesis and the ones which can resemble more the idea of the web audio editor for Freesound are the following:

- Audacity. It is the most capable free audio editor. It provides the user with a full set of editing and mastering tools, as well as a pretty complete list of audio effects, sound generator plugins and export formats.
- Ocenaudio. It is a free single-track editor for making destructive edits to audio files. Destructive edit means that the changes are made in the same sample file whenever the user hits the ‘Save’ button. This destructive approach is the one also used in Audacity, but Ocenaudio gives a clearer interface and includes the basic operations that Audacity is capable of performing.

[5] https://www.findsounds.com/

[6] https://freemusicarchive.org/search

- TwistedWave[7]. This editor offers two options: a stand-alone desktop application or an online web-based audio editor with the same functionalities. It can be used to create recordings directly on the web or edit files you already own by just uploading them to the web. It also includes some effects on top of the usual editing features, like amplify audio, create fades and change pitch or speed, as well as applying different external VST effects. Within the free tier of TwistedWave, you can edit mono files up to 5 minutes long and create an account in order to keep your files there.

Apart from TwistedWave, there exist more simple online audio editors with very similar features between them. According to [8], TwistedWave is one of the top 5 online edition tools available on the Internet, but there are more, for example, Beautiful Audio Editor[8], Sodaphonic[9], Bear Audio Tool[10] or Hya-Wave[11]. Out of this five, TwistedWave is the only one supporting all available web browsers, since the other four only works on Chrome and/or Mozilla. Despite this fact, the one that appeals to me the most due to its simplicity and performance is Hya-Wave. It is important to be aware that, to the best of my knowledge, neither TwsitedWave nor Hya-Wave are open-source projects, that means, their code and functionalities are not available of the web.

However, there exists another very capable online editor called AudioMass[12]. In this case, the whole project is open source, with its code available in Github[13]. This editor uses the previously mentioned Web Audio API framework Wavesurfer.js, which gives a lot of facilities in order to draw the sample waveform as well as methods which encapsulate the basic Web Audio API operations. AudioMass offers the basic playback controls for the sample and a catalogue of effects like fades, compressor, delay, distorsion or EQ among others. With some personalization and because it is open source, this editor can be embedded into other websites.

[7] https://twistedwave.com/online

[8] https://beautifulaudioeditor.appspot.com
[9] https://sodaphonic.com
[10] https://www.bearaudiotool.com/sp/
[11] https://wav.hya.io/#/fx
[12] https://audiomass.co
[13] https://github.com/pkalogiros/audiomass

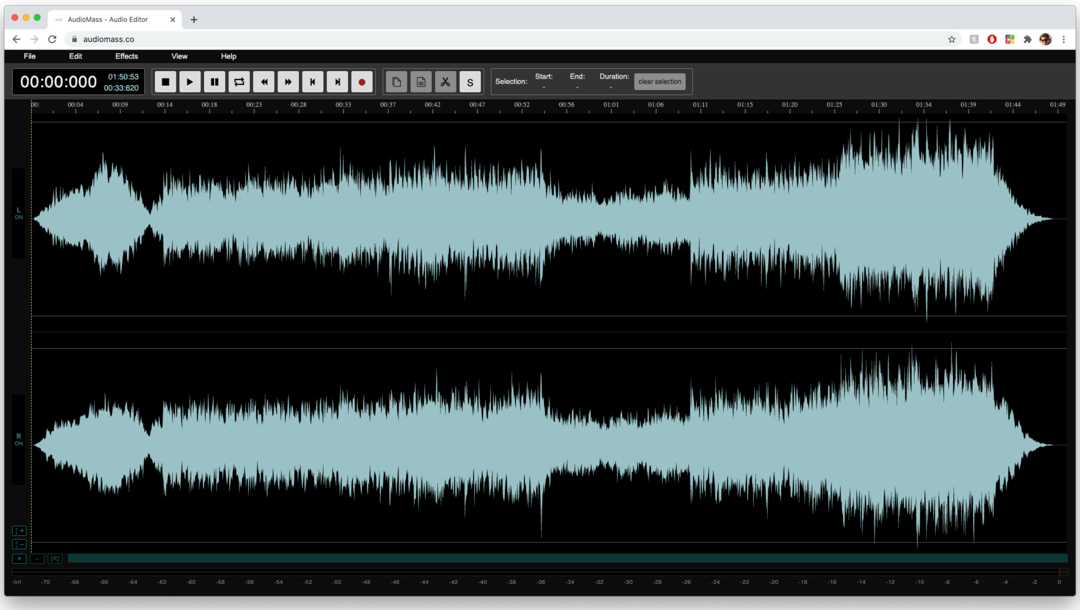


Figure 2. AudioMass editor

Hya-Wave provides a really easy-to-use and clear interface (Figure 2) where the user can see all the features available in the editor. It provides the basic operations to cut, paste, crop or export the audio, but it also includes the possibility to record audio directly from the browser and to add up to 18 different effects to the sample, like amplification, fade in and out, low and high-pass filters, etc.

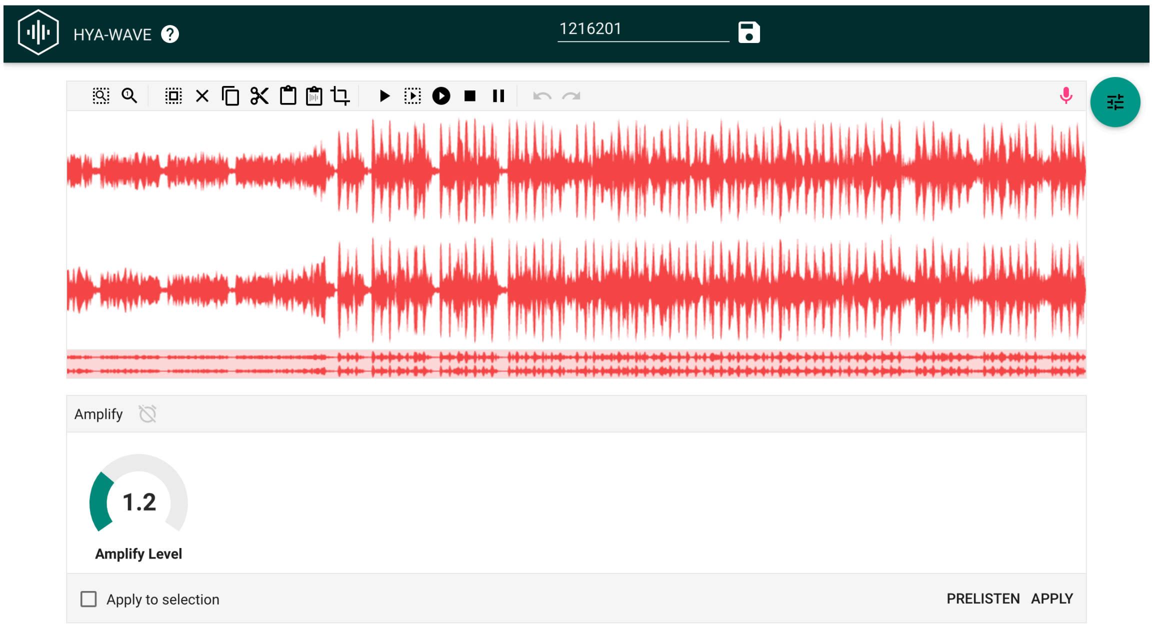


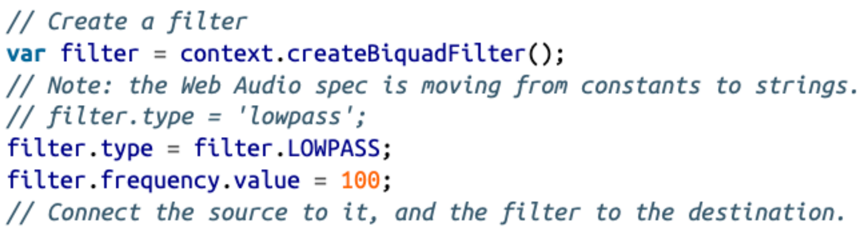
Figure 3. Hya-Wave layout

In my opinion, out of the online audio editors presented previously, Hya-Wave is the one that has the most features similar to the ones that I want to develop in my web editor embedded in the Freesound website. My idea is to handle the user the basic operations for formatting the audio and also offer the possibility to add different effects to the sample. I personally prefer Hya-Wave over AudioMass due to its simplicity and interface, but being open source, the code of the AudioMass editor will be taken into account in the development of the Freesound one in order to get some inspiration and solve any problem that I could encountered.

2.3. Web Audio API

Once the style and features of the editor is clearer, and the technologies to be used too, it is time to take a deeper look into the Web Audio API and see which algorithms are the ones that I am going to use and how this algorithms are usually used in this context. Let's start taking a look to how the Web Audio API handles the filters and effects that are going to be applied to the samples.

As explained in [2] (Chapter 6: *Advanced Topics*), when using the Web Audio API, in order to add some filter to the sample, the BiquadFilterNodes are used. It is very commonly used in order to build equalizers and manipulate sounds. For example, a simple low-pass filter to eliminate low frequency noise from a sample will be implemented like explained in Figure 3.



```
// Create a filter
var filter = context.createBiquadFilter();
// Note: the Web Audio spec is moving from constants to strings.
// filter.type = 'lowpass';
filter.type = filter.LOWPASS;
filter.frequency.value = 100;
// Connect the source to it, and the filter to the destination.
```

Figure 4. Low-pass filter using Web Audio API

The Web Audio API offers 8 different filters that will be implemented in the Freesound audio editor:

- Low-pass filter: Makes sounds more muffled
- High-pass filter: Makes sounds more tinny
- Band-pass filter: Cuts off lows and highs (e.g. telephone filter)
- Low-shelf filter: Affects the amount of bass in a sound
- High-self filter: Affects the amount of treble in a sound
- Peaking filter: Affects the amount of midrange in a sound
- Notch filter: Removes unwanted sounds in a narrow frequency range
- All-pass filter: Creates phaser effects

Other great features that could possibly be included in this editor could be time stretching and pitch shifting. Even though the Web Audio API is capable of performing many operations over audio, it has some issues related to these operations. As Dias et al. stated in [9], there is no way to retrieve the full frequency description using the Web Audio API. This situation forces developers to rely of the native Javascript implementation of the First Fourier Transform (FFT) in order to perform high quality time stretching, which leads to an increased overhead in computational costs because Javascript is an interpreted language. Nevertheless, in the same paper, the authors give a whole bunch of alternative libraries and algorithms in order to perform these two operations. After taking a look at all of them, and taking into account the different advantages and disadvantages of each of them, the library that I think would be the easiest to include in my project and the one that offers the cleanest solution for time stretching and pitch shifting is SoundTouch.js[14]. Making use of its PitchShifter object and its *tempo* and *pitch* attributes, these two operations should be easy to implement and add to my editor.

[14] https://github.com/cutterbl/SoundTouchJS

2.4. Evaluation

Finally, one last thing that I have to have in mind when developing this system is the way that the editor and its functionalities are going to be tested and evaluated by final users in order to get some impression and thoughts. Taking a look at the Audio Commons deliverable reports [10, 11], it seems that the evaluation methods for technological projects consist on different steps:

- Participatory methods. Prior to the development, it is a good practise to gather data from potential users regarding the features or functionalities that those users expect the system to have. These opinions should give the developer a good overview of which features to add to the system, as well as maybe some approach that he or she was not able to come across.
- Hands-on methods. After the development of the system, another set of users are asked to use the system and write down their opinions on the functionalities of it: is it easy to use, is it useful, does it have the functionalities that this type of system needs to have, etc.
- Informal methods. The developers can also retrieve useful information about the system when talking to potential stakeholders during demonstrations or performances

In conclusion, and taking all of this information about the Freesound environment and the technologies involving the Web Audio API that are going to be used, as well as the common features and operations that other editors include and the evaluation methods that are going to be developed for the testing of the editor, all the necessary information in order to start the development of the editor has been retrieved and studied.

3. Methodology

3.1. Planning

The planning of this project has been done following an agile methodology, developing it without previous deep documentation or planification. Some features might remember to the methodology followed by Scrum, like the division of the development in sprints, but no product backlog or sprints reviews have been made.

The development of this project up to the time this document is being written has been divided in 7 Sprints. The duration of each of them was not defined in advance. The content of each sprint is the following:

1. State of the Art study: technologies and alternatives
2. Stand-alone editor development
3. Integration with Freesound environment
4. Initial evaluation and documentation
5. Development of second version taking into account first evaluation
6. User cases evaluation
7. Final fixes and documentation

3.1.1. Sprint 1. State of the Art study: technologies and alternatives

All the different Javascript frameworks and the different audio editors studied prior to the development of this thesis are explained in *Chapter 1: State of the Art*.

3.1.2. Sprint 2. Stand-alone editor development

This sprint was dedicated to the creation of an HTML, CSS and Javascript stand-alone application[15] in order to develop the different actions of the editor without having to configure and download the whole Freesound project.

[15] https://robertops18.github.io/audio-editor-js/

3.1.3. Sprint 3. Integration with Freesound environment

Once the editor was working as expected with all the different functionalities available, the code was integrated with the Freesound code. I configured a local image of Freesound in my computer, forking the Freesound repository available in Github[16] in my account in order to integrate both projects into one.

3.1.4. Sprint 4. Initial evaluation and documentation

When a stable version of the editor was finished, I created an initial usability questionnaire[17] in order to get feedback from potential stakeholders and Freesound users. This questionnaire will give me an initial view of the general opinion about the editor and what to improve, fix or change when developing its final version. Deeper information and details about this first questionnaire will be explained in *Chapter 2.3.1: Usability questionnaire*.

3.1.5. Sprint 5: Development of second version

Taking into account the answers for the first evaluation questionnaire, I got a general impression about the look and feel and overall functionalities of the editor. Taking this general feedback and some of the specific proposals that were gathered, the second version of the editor was developed.

3.1.6. Sprint 6: User cases evaluation

After the development of a second and stable version, the second and final evaluation was held. I designed 14 different user cases in order to test the individual functionalities of the editor, taking into account the expected and actual workflow of each of them. Using this approach, I was able to detect which of the functionalities work perfectly as expected and which of them had some type of bug yet. Deeper information and details about this user cases evaluation will be explained in *Chapter 2.3.2: User cases evaluation*.

[16] https://github.com/robertops18/freesound

[17] https://docs.google.com/forms/d/e/1FAIpQLScTs7AMX6wXjqvGbWhz_O4GIxe5qfSIwFxq5FotHVMq7P8UrQ/viewform?usp=sf_link

3.1.7. Sprint 7: Final fixes and documentation

Finally, when performing the user cases evaluation, some issues regarding the fade in and out functionalities appeared. After these problems where well implemented and fixed, and the user case for this functionality was approved, the final version of the editor was closed, and this document written taking into account all the workflow that was being held for the development of the project.

3.2. Implementation

From the beginning, the structure of the project has been the following:

- HTML file containing the visual elements of the editor
- CSS file for customization of the elements
- Javascript files containing the logic of the editor
    - Main JS file with all the methods available
    - Plugin files in order to use the different external libraries selected

Both the code of the stand-alone editor and the Freesound integrated editor are available in my Github account as public repositories[18] [19]. The code is also available in the files attached together with this document.

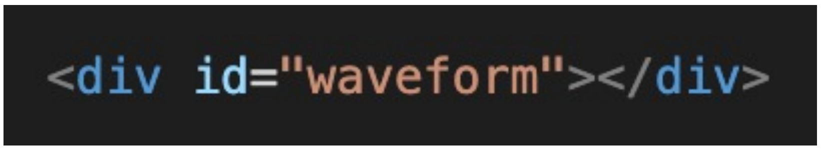
3.2.1. Wavesurfer object initialization

In Chapter 1, different Javascript frameworks oriented to audio treatment were studied, but the one which is finally used is Wavesurfer. Mostly all the functionalities of the editor are based on the different methods available in its API, apart from some of them where other plugins (PitchShifter, SmoothFade or Bitcrush) or just plain Web Audio API code are used.

The first thing that was created was the waveform visualization. For this, a <div> tag is created and named after a unique ID. Then, in the main JS file, a Wavesurfer instance is created referencing the ID of the HTML element. When this is done, the waveform is

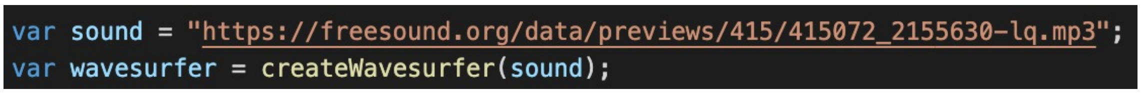
[18] https://github.com/robertops18/freesound
[19] https://github.com/robertops18/audio-editor-js

shown on screen, and the Wavesurfer object is ready to be used in order to execute the different methods available in its API.



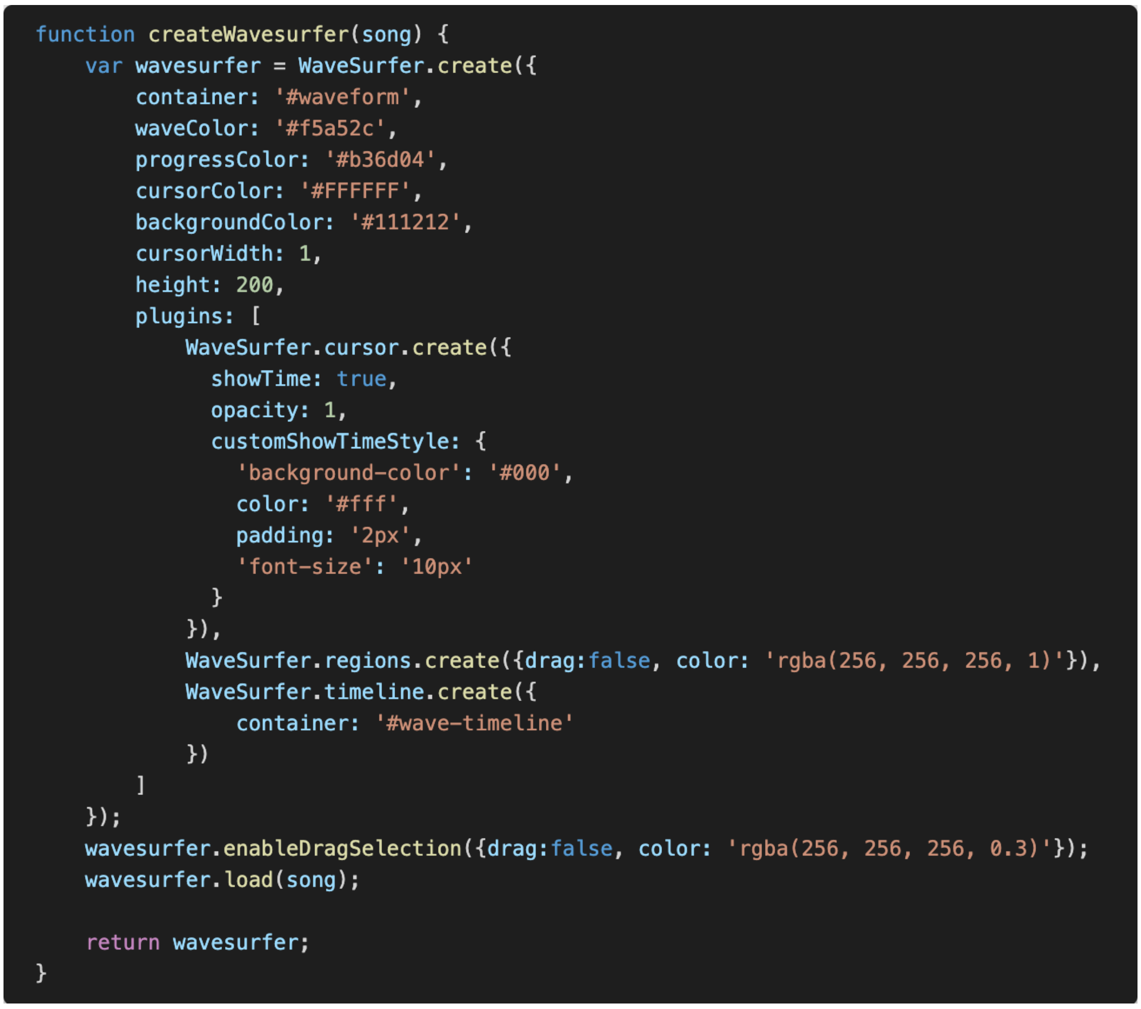
```
<div id="waveform"></div>
```

Figure 5. HTML div tag for waveform



```
var sound = "https://freesound.org/data/previews/415/415072_2155630-lq.mp3";
var wavesurfer = createWavesurfer(sound);
```

Figure 6. Creation of the Wavesurfer object



```
function createWavesurfer(song) {
    var wavesurfer = WaveSurfer.create({
        container: '#waveform',
        waveColor: '#f5a52c',
        progressColor: '#b36d04',
        cursorColor: '#FFFFFF',
        backgroundColor: '#111212',
        cursorWidth: 1,
        height: 200,
        plugins: [
            WaveSurfer.cursor.create({
              showTime: true,
              opacity: 1,
              customShowTimeStyle: {
                'background-color': '#000',
                color: '#fff',
                padding: '2px',
                'font-size': '10px'
              }
            }),
            WaveSurfer.regions.create({drag:false, color: 'rgba(256, 256, 256, 1)'}),
            WaveSurfer.timeline.create({
                container: '#wave-timeline'
            })
        ]
    });
    wavesurfer.enableDragSelection({drag:false, color: 'rgba(256, 256, 256, 0.3)'});
    wavesurfer.load(song);

    return wavesurfer;
}
```

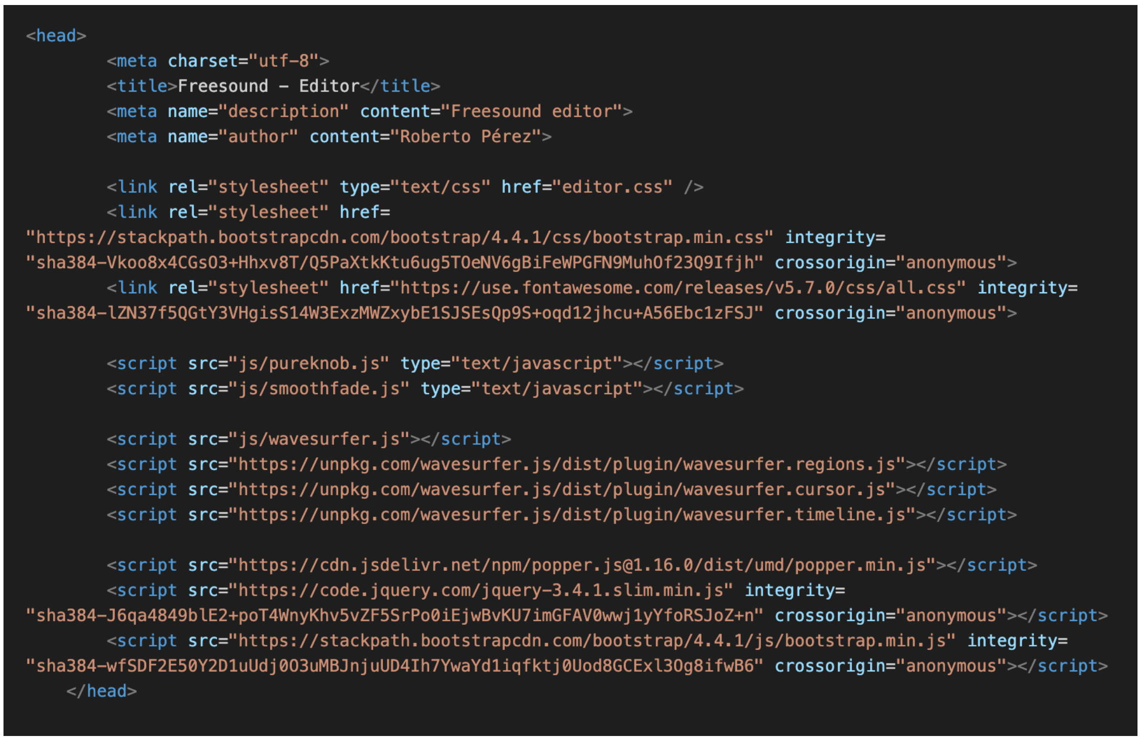
Figure 7. createWavesurfer method

At the time of creating the Wavesurfer object, the different plugins that are going to be used in the editor are also added to it. In this case, the three plugins used are: cursor (to move it throughout the waveform), regions (to select region in waveform for deleting or clearing) and timeline (to see the duration and actual time of the sample). Moreover, all the configuration parameters of the object are also configured, and the sample loaded into the object using the 'load' method.

Once the Wavesurfer object is created, it can be used in order to implement the different actions, filters and effects of the editor.

3.2.2. User interface

The visual elements of the editor have been developed using HTML (HyperText Markup Language), which is 'a standard markup language for documents designed to be displayed in a web browser' [14]. It is usually combined with technologies such as CSS (Cascading Style Sheets) and scripting languages as Javascript. On top of that, the Bootstrap framework [15] has also been used in order to build a responsive and cleaner interface. Bootstrap is the world's most popular HTML framework for this matter. It can be used very easily, just adding a pair of lines with the link to the necessary scripts in the head section of the HTML file. Adding those lines will give users the opportunity to use all of the different Bootstrap elements and functionalities in their projects.



```html
<head>
        <meta charset="utf-8">
        <title>Freesound - Editor</title>
        <meta name="description" content="Freesound editor">
        <meta name="author" content="Roberto Pérez">

        <link rel="stylesheet" type="text/css" href="editor.css" />
        <link rel="stylesheet" href=
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity=
"sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.0/css/all.css" integrity=
"sha384-lZN37f5QGtY3VHgisS14W3ExzMWZxybE1SJSEsQp9S+oqd12jhcu+A56Ebc1zFSJ" crossorigin="anonymous">

        <script src="js/pureknob.js" type="text/javascript"></script>
        <script src="js/smoothfade.js" type="text/javascript"></script>

        <script src="js/wavesurfer.js"></script>
        <script src="https://unpkg.com/wavesurfer.js/dist/plugin/wavesurfer.regions.js"></script>
        <script src="https://unpkg.com/wavesurfer.js/dist/plugin/wavesurfer.cursor.js"></script>
        <script src="https://unpkg.com/wavesurfer.js/dist/plugin/wavesurfer.timeline.js"></script>

        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity=
"sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity=
"sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
    </head>
```

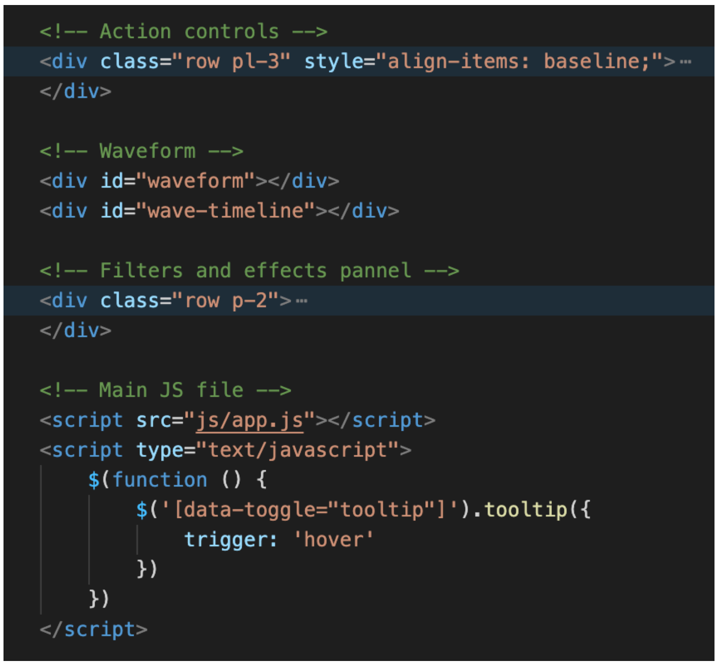
Figure 8. HTML head section

The head section in a HTML file is used for three main purpose: store metadata about the project, link the styles files (CSS) and declare the different JS files used in its functionality. Having said that, more scripts can be declared in the body of the file, where in this case is declared the main JS file, just for convenience with the object treatment. In this case, as we can see in Figure 7, apart from the Bootstrap CSS and JS files, more files are declared:

- Editor.css: main styles file for the editor
- Bootstrap CSS file
- Fontawesome: icon library
- Two JS files used for specific functionalities: knobs and fades
- Wavesurfer and its plugins
- JQuery and Bootstrap JS files

All these files configured in the HTML head will be used in the body part, which is the one where the different UI elements are declared. In the case of my editor, the body element is divided in four parts (as it can be seen in Figure 8):

1. Row of actions
2. Waveform visualization and timeline (explained in 2.2.1)
3. EQ and effects panels
4. Main JS file declaration and tooltips script

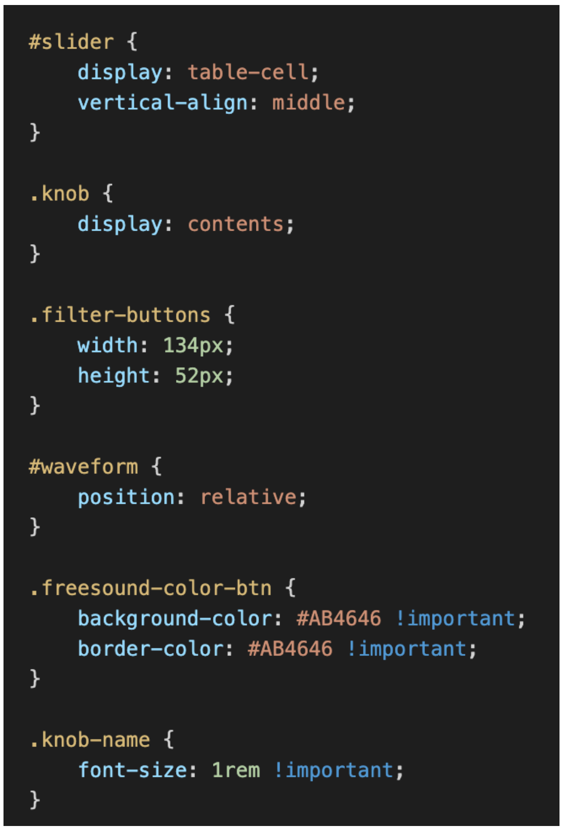


```
<!-- Action controls -->
<div class="row pl-3" style="align-items: baseline;">⋯
</div>

<!-- Waveform -->
<div id="waveform"></div>
<div id="wave-timeline"></div>

<!-- Filters and effects pannel -->
<div class="row p-2">⋯
</div>

<!-- Main JS file -->
<script src="js/app.js"></script>
<script type="text/javascript">
    $(function () {
        $('[data-toggle="tooltip"]').tooltip({
            trigger: 'hover'
        })
    })
</script>
```

Figure 9. HTML body section

3.2.3. Styles

Most of the styles in this first version of the editor are applied thanks to Bootstrap. However, there are some size and display settings which must be applied manually to some specific UI elements. For this matter, extra CSS are used. Cascading Style Sheets is 'a style sheet language used for describing the presentation of a document written in a markup language like HTML' [16]. The few settings done to the editor view can be seen in the editor.css file and in Figure 9. The most remarkable ones are the color change of some buttons in order to match the Freesound look and feel and the resize of the knob's font.



```css
#slider {
    display: table-cell;
    vertical-align: middle;
}

.knob {
    display: contents;
}

.filter-buttons {
    width: 134px;
    height: 52px;
}

#waveform {
    position: relative;
}

.freesound-color-btn {
    background-color: #AB4646 !important;
    border-color: #AB4646 !important;
}

.knob-name {
    font-size: 1rem !important;
}
```

Figure 10. editor.css file

3.2.4. Functionalities

All of the functionalities of the editor are stored in a JS file (app.js). This file includes all necessary object initializations and methods declarations for the editor to work as expected. It also links the actions done in the UI (like clicking a button) with the method that has to be triggered when doing that action, via HTML query selectors and listeners.

3.2.4.1. Actions

The different available actions are separated in four panels:

- Playback controls. This includes play/pause, stop (returns the cursor to the start of the sample), go backwards and advance forward (both set to 2 seconds)
- Undo and redo workflow, as well as clearing and deleting regions of the sample
- Zoom panel: zoom in, zoom out and zoom to selected region
- Crop and get the selected region of the sample, get the original one, reverse the sample, download it and the help button, which opens a modal with details and shortcuts about the editor.

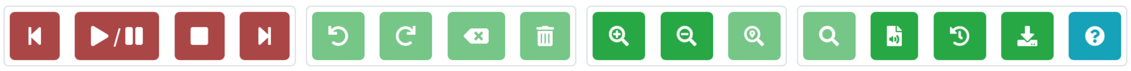


Figure 11. Editor actions

3.2.4.2. Waveform visualization and timeline

The user can select regions on the waveform in order to perform some of the actions. Also, the cursor moves along with the sound and a visual timeline is displayed at the bottom.

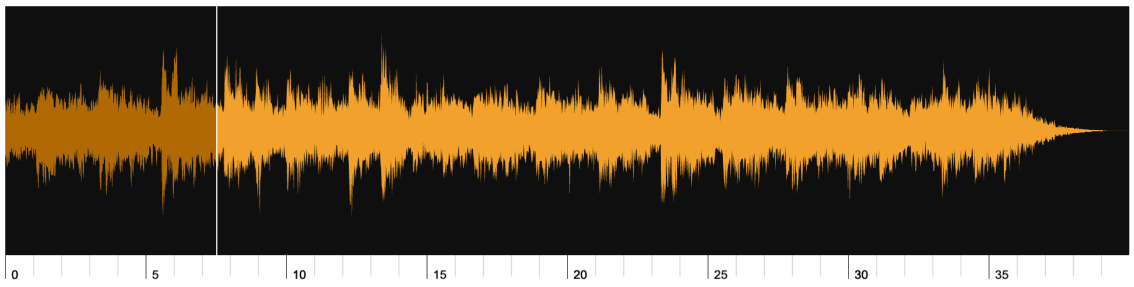


Figure 12. Waveform and timeline

3.2.4.3. EQ panel

It contains the knobs for applying the three filters available: lowpass, bandpass and highpass. For the lowpass and highpass filter, only the frequency cutoff can be set, but for the bandpass filter, the user can also change the resonance factor Q.

3.2.4.4. Effects

The effects for this editor are all set using knobs. The available effects are:

- Gain control. The user can change the gain value of the sample from -20 to 20 dB.
- Fade in and out. When using these effects, the gain or the sample will increase from 0 to 1 at the start of the sample for the fade in and decrease from 1 to 0 in the case of the fade out. The time for this change is set using the knobs.
- Playback rate. The user can change the playback rate of the sample, also changing its pitch.

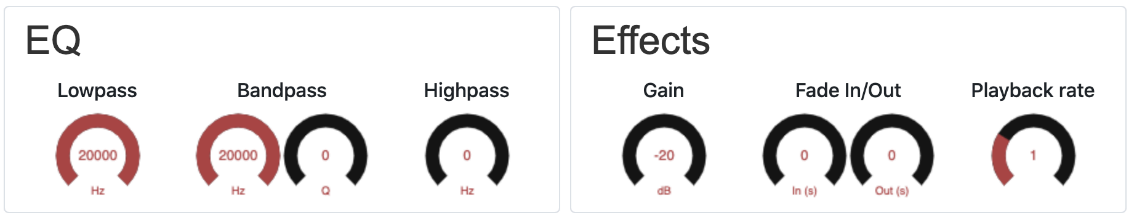


Figure 13. EQ and effects panels

3.3. Evaluation

3.3.1. Usability questionnaire

After a first stable version of the editor was available, I created a simple questionnaire [19] in order to store some feedback and opinions about the style, performance and usability of this first version. With that information, I will be able to fix the different bugs that may appear in the functionalities of the editor, as well as taking into account the usability and UI suggestions made by potential users of the editor.

3.3.1.1. Structure

The first questionnaire was divided in three parts:

- General usability
  - Do you know where to find the different functionalities of the editor?
  - Is there any help needed for the use of the editor?
  - Do you find the editor simple and easy-to-use?
  - Do the different editor actions work as expected?
  - Do the different editor filters work as expected?

    - Do the different editor effects work as expected?
- Interface
    - The font and size of the text is
    - The icons and images used are
    - The used colors are
    - Is the editor UI design clear and attractive?
    - Do you think the different functionalities are well structured?
- General impressions
    - Do you think the editor is useful in the context of Freesound?
    - Do you think the editor will add value to the user experience in the web?
    - Would you use the Freesound integrated editor for quick fixes in the audio instead of your preferred desktop editor?
    - Do you think the available functionalities are enough in order to edit the sample?

3.3.1.2. Results

In the case of the general usability of the editor, users agree that the behavior of the knobs was not the appropriate one. In the first version of the editor, the user would have to select the value in the knob and then apply the filter or effect pressing a button. The preferred workflow was to apply it was changing the value of the knob, in order to experiment easily with them. This approach was developed in the second version of the editor, as well as the addition of tooltips for the actions, which was another feature requested by a user.

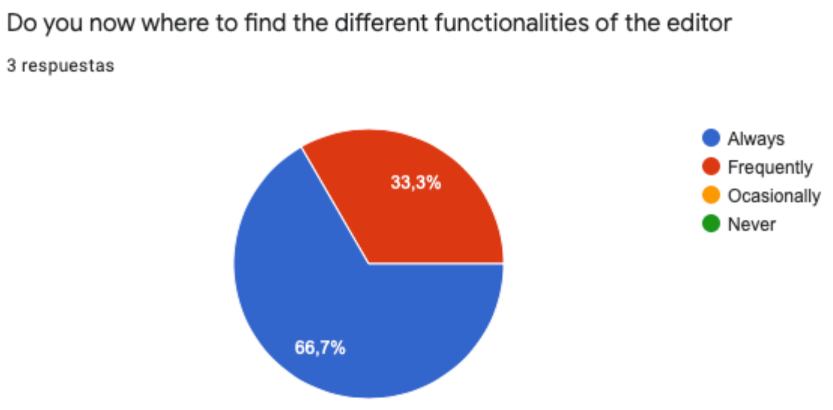


Figure 14. General usability: Question 1

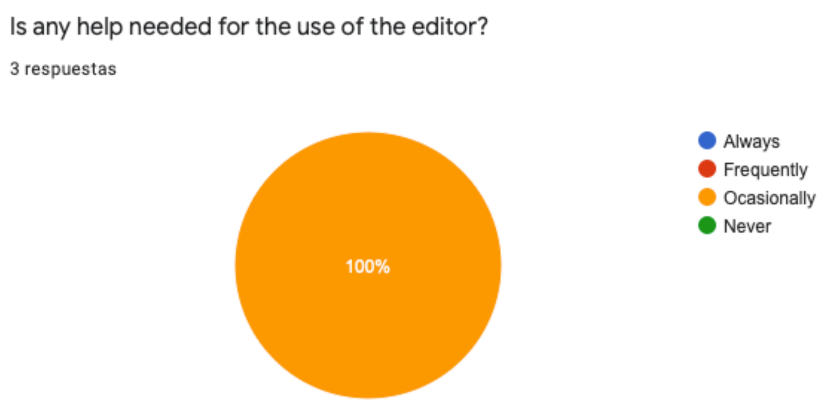


Figure 15. General usability: Question 2

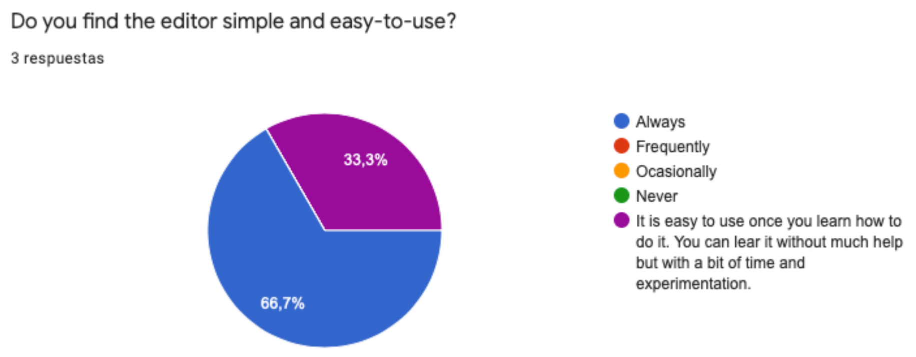


Figure 16. General usability: Question 3

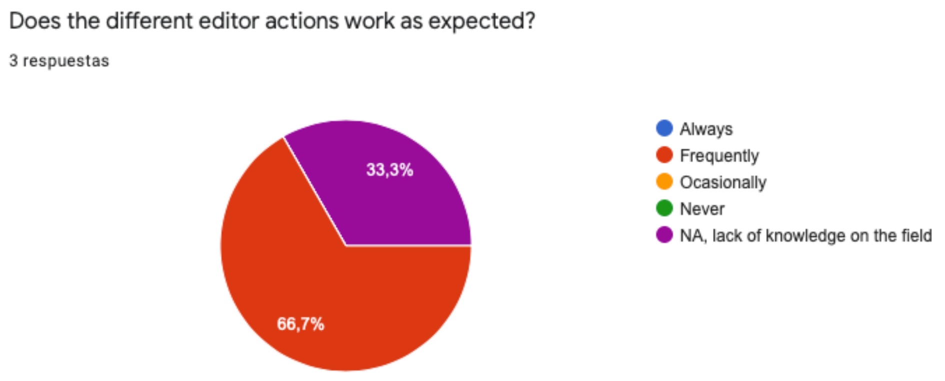


Figure 17. General usability: Question 4

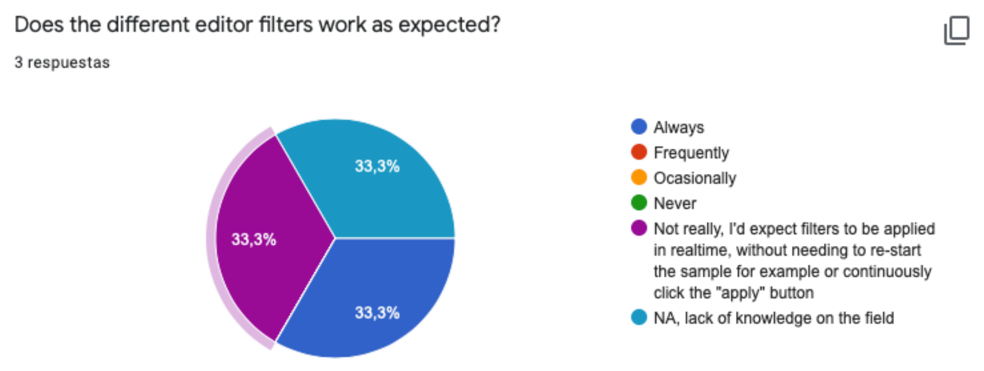


Figure 18. General usability: Question 5

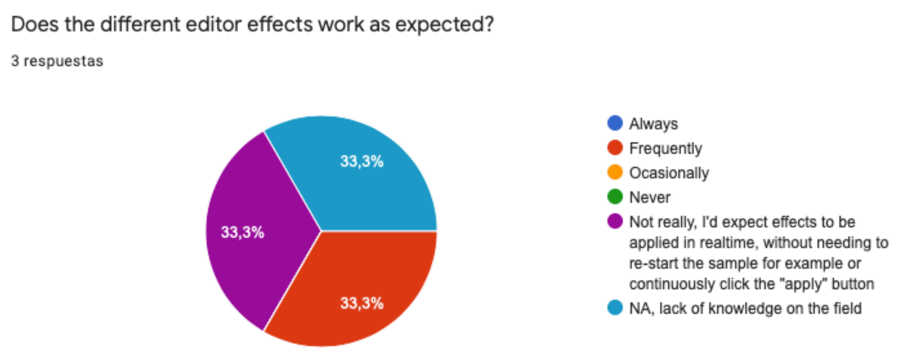


Figure 19. General usability: Question 6

Regarding the interface, the general opinion was that it took a lot of space in the web. The knobs for filters and effects, and the action button were too big, and the user had to scroll through in order to apply them. In the second version, the user interface was completely changed in order to get all the editor functionalities just in one screen: the actions buttons where replaced by icons at the top of the waveform, some filters were deleted, and all knob's size decreased.

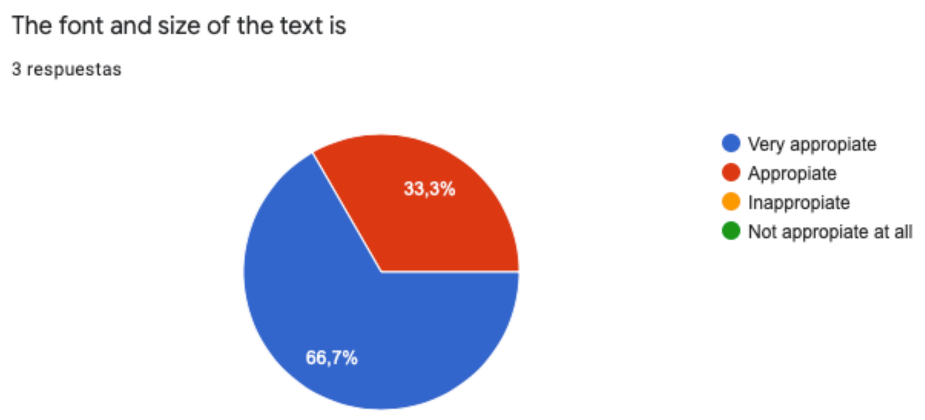


Figure 20. Interface: Question 1

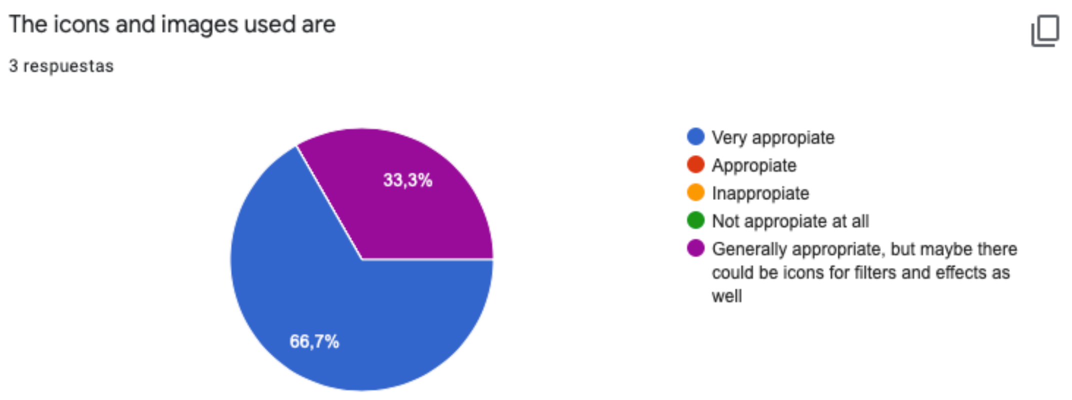


Figure 21. Interface: Question 2

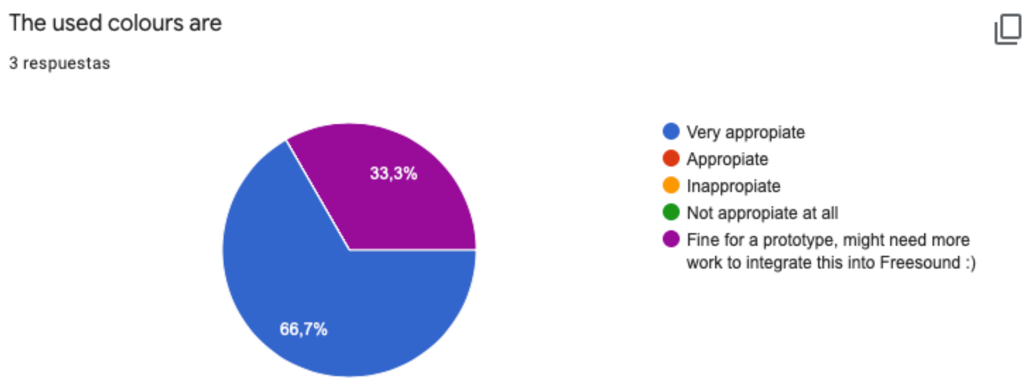


Figure 22. Interface: Question 3

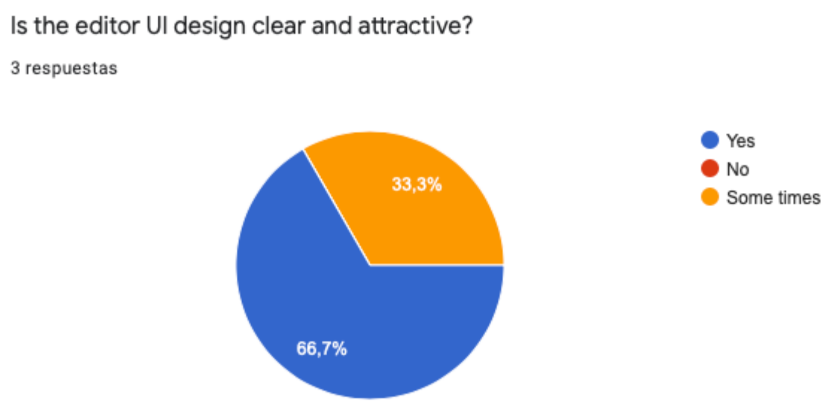


Figure 23. Interface: Question 4

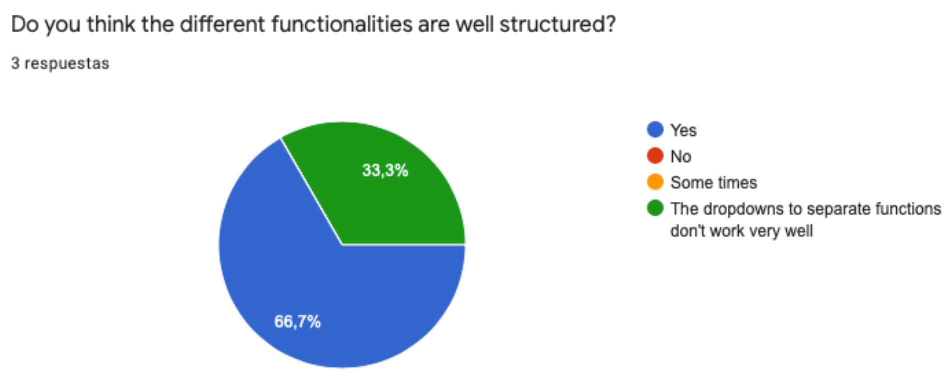
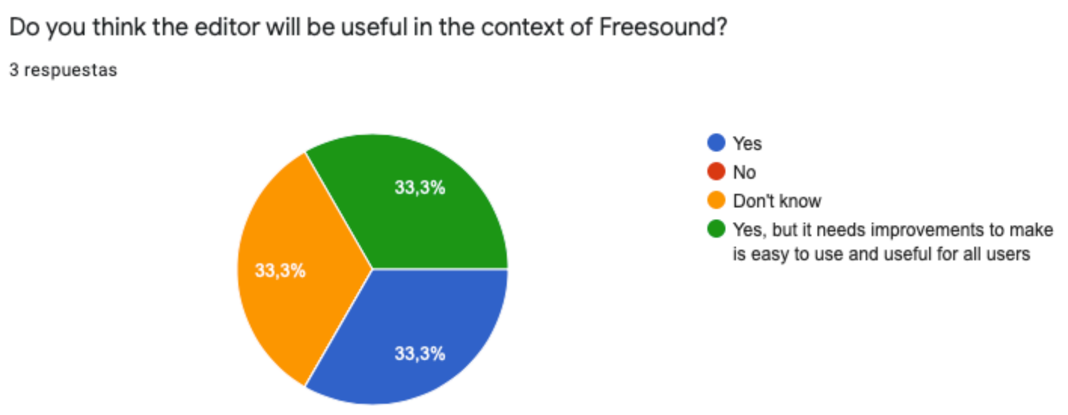


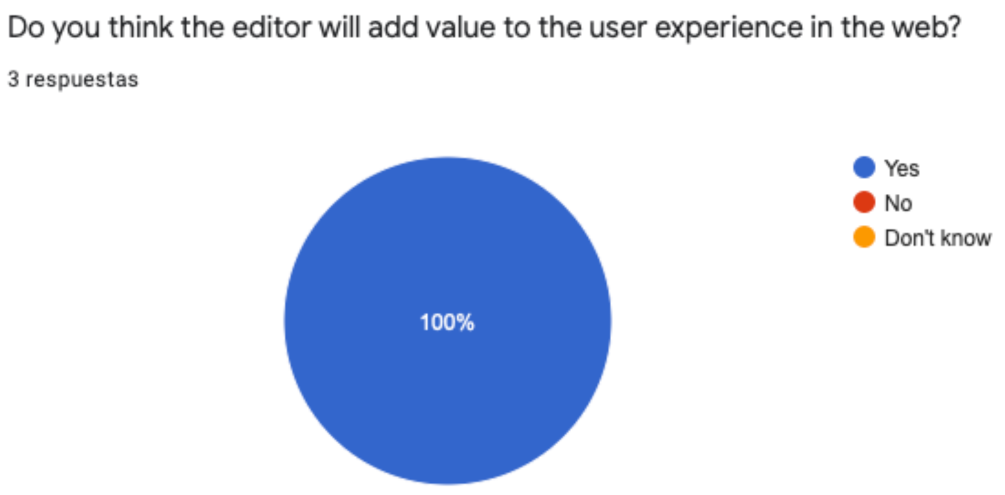
Figure 24. Interface: Question 5

Finally, the general impression about the editor was that it would definitely be useful in the context of Freesound, adding huge value to the user experience in the web. However, the editor needed some changes like the mentioned above, and some other functionalities like being able to change the playback rate (developed in second version) in order to be really useful.


Do you think the editor will be useful in the context of Freesound?

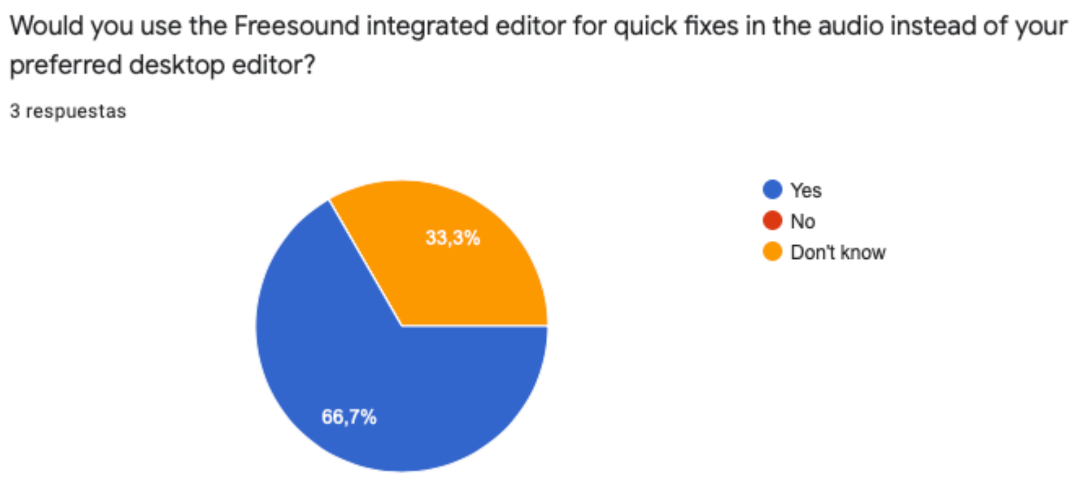
3 respuestas

Figure 25. General impressions: Question 1


Do you think the editor will add value to the user experience in the web?

3 respuestas

Figure 26. General impressions: Question 2


Would you use the Freesound integrated editor for quick fixes in the audio instead of your preferred desktop editor?

3 respuestas

Figure 27. General impressions: Question 3

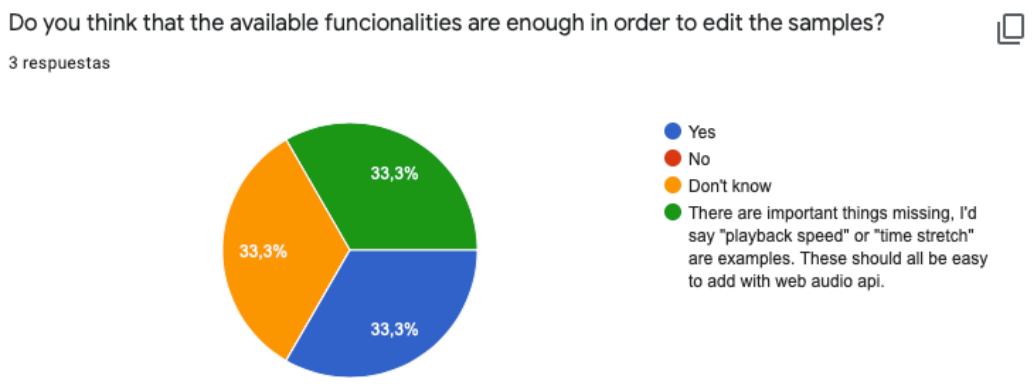
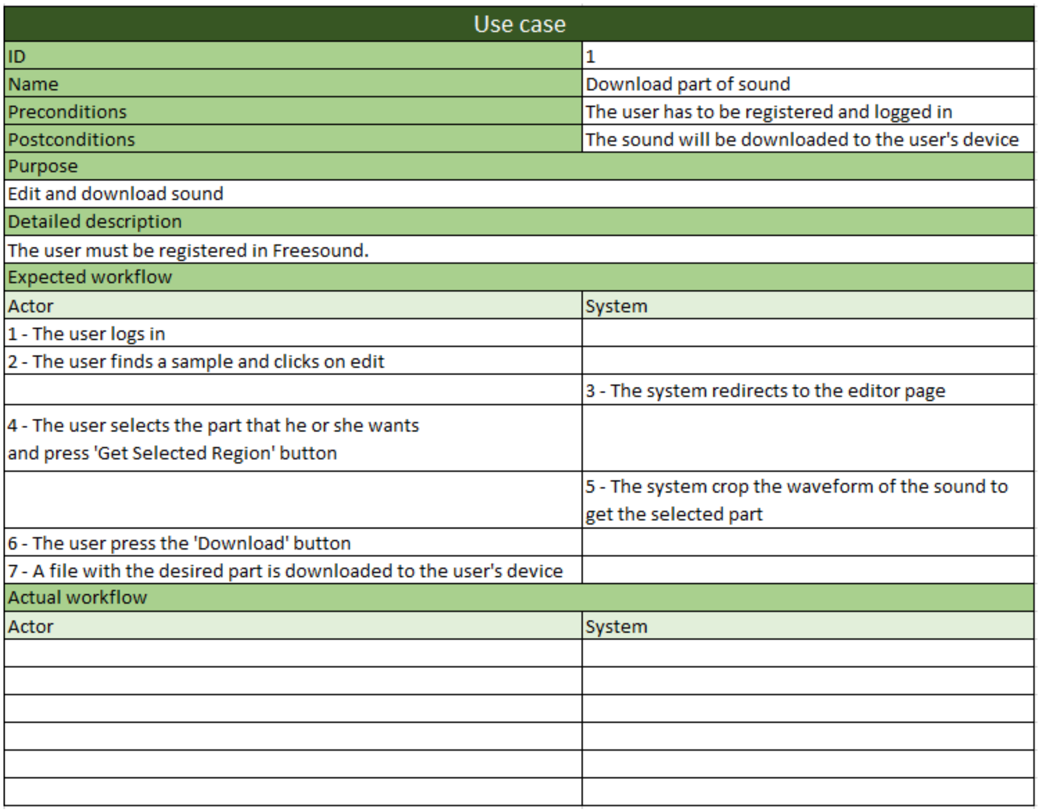


Figure 28. General impressions: Question 4

3.3.2. User cases evaluation

For a second evaluation, the performance of several user cases will be needed by some volunteer potential users of the application. These user cases will include different tasks to be done using the editor, as well as its expected behavior. Taking into account this expected behavior, users will need to take notes about the workflow they are coming across with and compare it with the expected one. This type of evaluation will give me deeper knowledge about the right or wrong behavior of all the functionalities of the editor.



Use case	
ID	1
Name	Download part of sound
Preconditions	The user has to be registered and logged in
Postconditions	The sound will be downloaded to the user's device
Purpose	
Edit and download sound	
Detailed description	
The user must be registered in Freesound.	
Expected workflow	
Actor	System
1 - The user logs in	
2 - The user finds a sample and clicks on edit	
	3 - The system redirects to the editor page
4 - The user selects the part that he or she wants and press 'Get Selected Region' button	
	5 - The system crop the waveform of the sound to get the selected part
6 - The user press the 'Download' button	
7 - A file with the desired part is downloaded to the user's device	
Actual workflow	
Actor	System

Figure 29. Example of user case

3.3.2.1. *Structure*

I designed a total of 14 user cases in order to test all the functionalities of the editor:

1. Playback
2. Clear region
3. Delete region
4. Zoom
5. Get selected region
6. Get original sample
7. Reverse
8. Help
9. Tooltips
10. Undo and redo
11. Filters
12. Gain
13. Fades
14. Playback rate

All of them follow the structure presented in *Figure 13*. The complete user cases spreadsheet is available in Google Drive[20] and in the files attached to this document.

3.3.2.2. *Results*

As for this second evaluation, the idea was to have a representative group of 5 to 10 people that would run the different user cases in order to test all the functionalities of the editor. However, and due to time and schedule constraints, this approach was not possible, and I finally performed all the user cases on my own.

For the first iteration of the user cases, the goal was to test if the actual workflow of all the functionalities was the expected one. This was done following the steps of each individual user case and comparing the output of the editor to the expected behaviour of the action performed. After this first iteration of the user cases, the workflow for all the functionalities was the expected one, with the exception of the fade in and out effects.

[20] https://drive.google.com/file/d/1fXqwKNxQAxGGUA2viFV7juhWBbbinviD/view?usp=sharing

When testing these effects, I noticed that the changes on gain at the start and end of the sample were noticed when playing the sound on the editor but did not reflect in the downloaded sample. In order to solve this problem, an approach using the OfflineAudioEditor was used to render the buffer just after applying the fade, loading the rendered one in the waveform to be sure that the progressive change in gain was applied and saved. Furthermore, the workflow used for the fades was changed in order to match the one used in other editors such as Audacity and similar ones: instead of selecting the duration of the fade, the user will select a region of the sample in which the fade will be applied and then apply the effect.

After the fades were fixed, another iteration of the user cases was held with the same purpose as the first one. Although only the fades workflow was changed, all the user cases must be retested because some functionality might stop working during the development of the new features due to incompatibilities or typos in the code. Finally, all the user cases, including the fades one passed, that is, all the actual workflows of the editor worked as expected and the editor was ready to go.

3.3.3. Changes timeline

Taking into account the output of each evaluation iteration, I made the following changes to the appearance and functionalities of the editor (Version 0.1 corresponds to the one before the usability questionnaire, the 0.2 after the first user cases iteration and 0.2.1 after the second user cases iteration)

Version 0.1	Version 0.2	Version 0.2.1
Big knobs	Smaller knobs	
Knobs not applying in real time. Apply with a button	Knobs applying in real time when changing value	
Actions in dropdown and text	Actions on top of the waveform and with icons	
Actions without tooltips	Tooltips added to actions	
Filters in dropdown	Filters in panel under waveform	
8 filters available	3 filters available (lowpass, bandpass and highpass)	

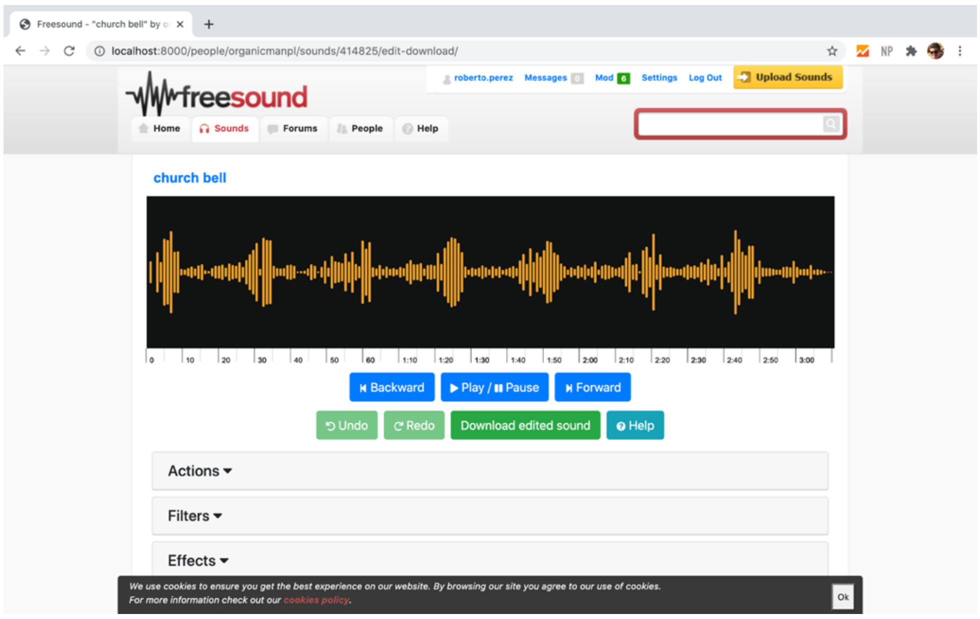
Effects in dropdown	Effects in panel under waveform	
Bootstrap CSS	Custom CSS to match Freesound look and feel	
Interface too big, user has to scroll	Interface in just one screen	
Fade in and out with knob and time duration	Fade in and out with knob and time duration	Fade in and out with region and button

Figure 30. Changes timeline

4. Prototypes

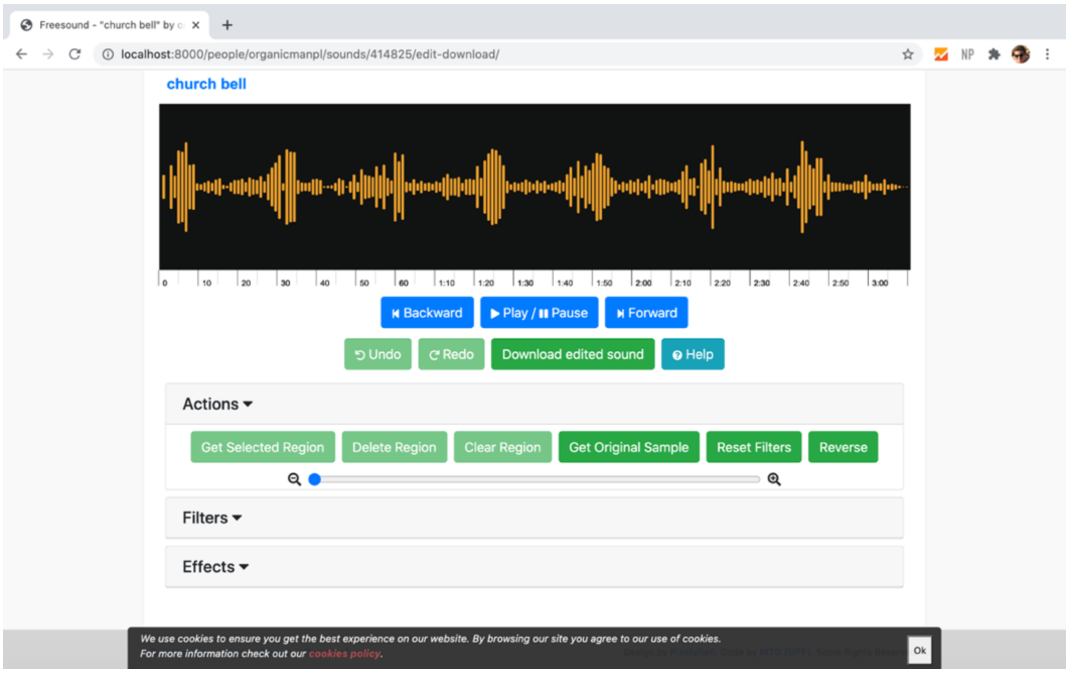
4.1. First version

The result of this first version of the editor, which was likely to change due to the results of the first evaluation, was the following:



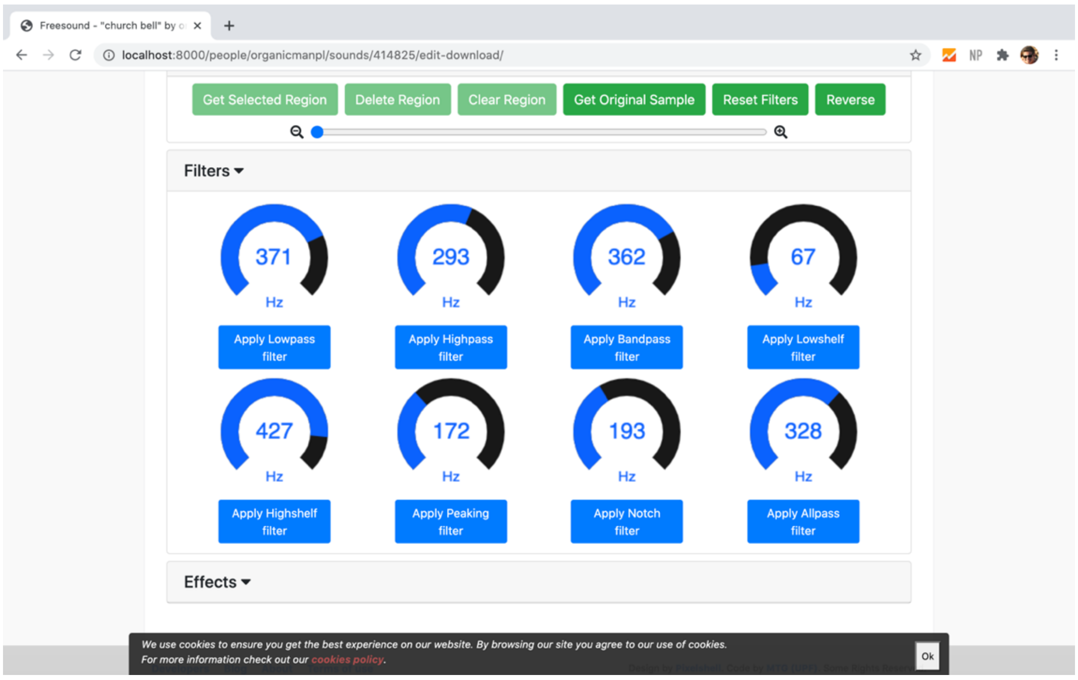
We use cookies to ensure you get the best experience on our website. By browsing our site you agree to our use of cookies. For more information check out our cookies policy.

Figure 31. General view



We use cookies to ensure you get the best experience on our website. By browsing our site you agree to our use of cookies.
For more information check out our cookies policy.

Figure 32. Actions dropdown



We use cookies to ensure you get the best experience on our website. By browsing our site you agree to our use of cookies.
For more information check out our cookies policy.

Figure 33. Filters dropdown

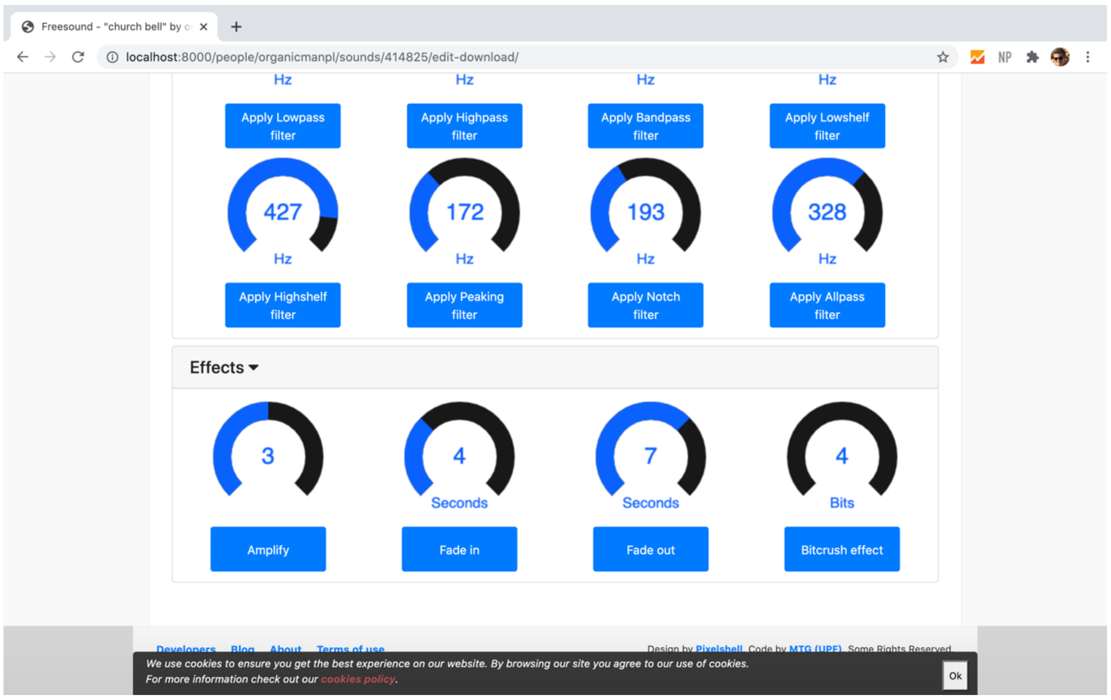

Freesound - "church bell" by o
localhost:8000/people/organicmanpl/sounds/414825/edit-download/
Hz
Apply Lowpass filter
Apply Highpass filter
Apply Bandpass filter
Apply Lowshelf filter
427
172
193
328
Apply Highshelf filter
Apply Peaking filter
Apply Notch filter
Apply Allpass filter
Effects
3
4
Seconds
7
Bits
Amplify
Fade in
Fade out
Bitcrush effect
Developers Blog About Terms of use
Design by Pixelshell. Code by MTG (UPF). Some Rights Reserved.
We use cookies to ensure you get the best experience on our website. By browsing our site you agree to our use of cookies.
For more information check out our cookies policy.
Ok

Figure 34. Effects dropdown

4.2. Second and final version

Taking into account the information and opinions gathered in the usability questionnaire, as well as the advice of my supervisor for this thesis, the second and final version of the editor is the following:

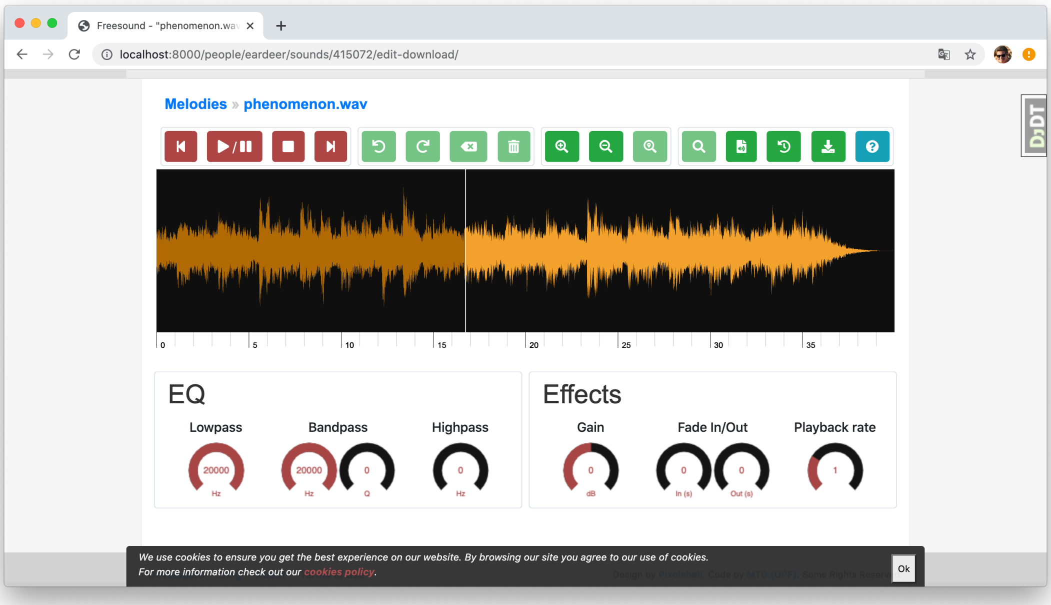

Freesound - "phenomenon.wa
localhost:8000/people/eardeer/sounds/415072/edit-download/
Melodies » phenomenon.wav
EQ
Lowpass
Bandpass
Highpass
20000
Hz
Q
Effects
Gain
Fade In/Out
Playback rate
dB
In (s)
Out (s)
We use cookies to ensure you get the best experience on our website. By browsing our site you agree to our use of cookies.
For more information check out our cookies policy.
Ok

Figure 35. Second version before user cases

As explained in previous chapters, all the functionalities (actions, filters and effects) are displayed in a single screen, improving the user experience while using the editor. Moreover, the waveform display and colors have been changed in order to match the style used in the rest of the Freesound website.

But, after the first iteration of the user cases evaluation, the workflow of the fade in and out effects was changed, so the interface changed as well in order to reflect those changes. The knobs for the fades were replaced by two buttons, one for each fade, which are disabled by default and only activate when a region is selected in the waveform.

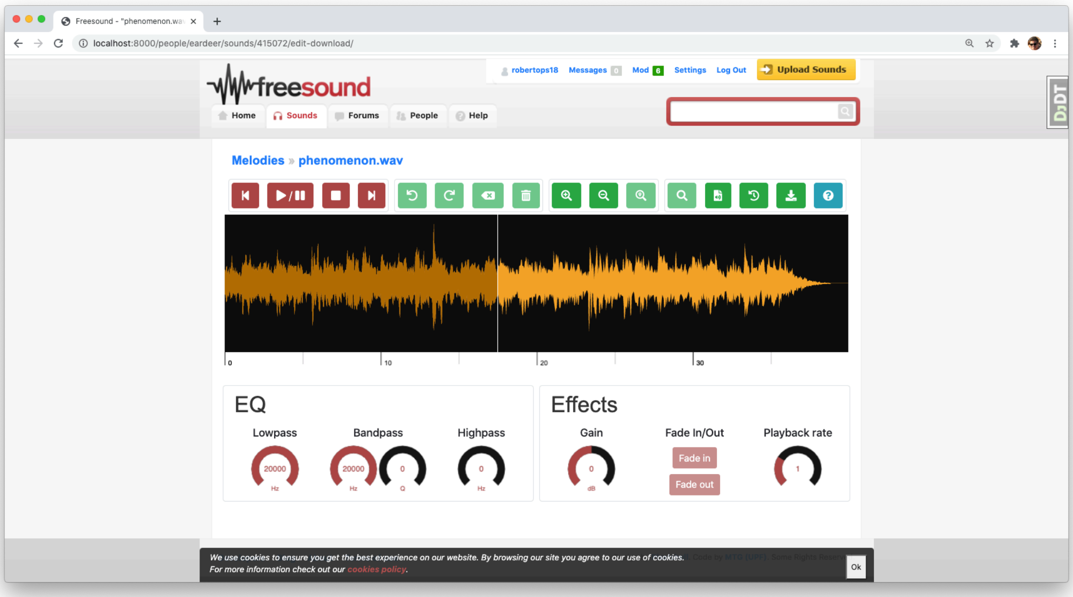


Figure 36. Final version after user cases

5. Solved problems

During the development of this editor, I encountered several challenges due to my inexperience when using the Web Audio API and its frameworks. One of them was how to export the whole sample including the actions, filters and effects applied to it.

From the beginning, I knew this was going to be one of the biggest challenges of the whole development. After some research, I came across the way to export the AudioBuffer contained in the AudioContext of the Web Audio API to a WAV file. This worked perfectly, and all the buffer related operations worked as expected. But, when developing the filters and effects, I realized that the buffer approach was not fulfilling my needs because filters and effects are not applied to the buffer, the Web Audio API creates new nodes in its graph in order to apply them. After more research, and thanks to the help of my supervisor, I was able to export the whole sample including the filters and effects using an OfflineAudioContext, which renders the whole graph used in the website into a buffer, which I can then export to a WAV file. The OfflineAudioContext is created at the momento the user presses the download button, taking into account all of the nodes present in the original AudioContext.

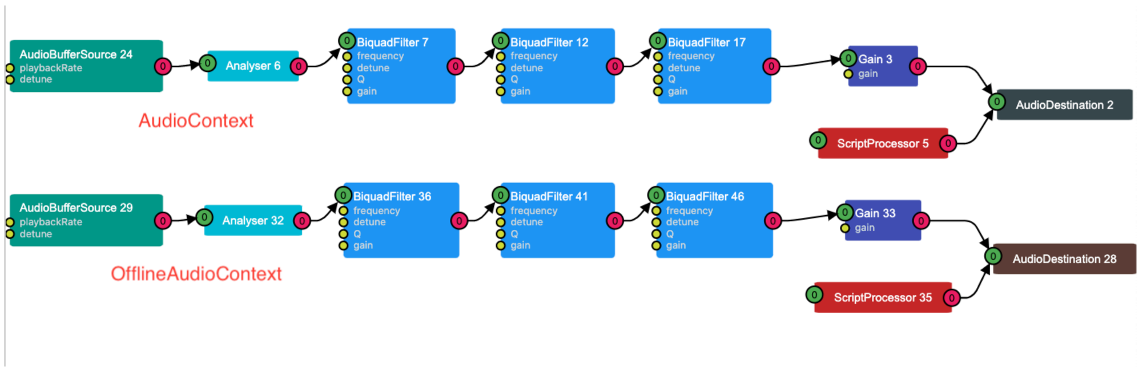


Figure 37. AudioContext and OfflineAudioContext graph

The same OfflineAudioContext approach was the one used to fix the fade in and out effects to the waveform. The changes were only noticeable just after applying the change in gain, so I used an OfflineAudioContext to render the gain change of the fades into a buffer to later load it into the Wavesurfer object and save the fade changes.

6. Conclusions and future work

This idea of adding a simple editor to the Freesound webpage will add some huge value to it and will differentiate it from other similar environments and communities, because, to the best of my knowledge, no other similar webpage offers this feature to their users. On top of that, the users workflow when downloading a sound for their projects will be really simplified, because of the fact that they will only have to use to Freesound webpage in order to edit and download the sample, without having to open and use the editor of their choose. Moreover, the editor has been built in order to let unexperienced users without editing skills be able to use the simpler actions and effects without any previous knowledge about the field. The selection of this type of design over other options like integrating the open source AudioMass project, a fully featured and powerful editor, was made because not every user that uses Freesound knows how to properly use all the audio features that a complete editor offers, so it would be better to have a simpler option where everyone is able to slightly edit the sample prior to its download if needed, but including some more advanced options, like the usage of the available filters in the EQ panel.

In addition, the technologies used for the development of the editor makes it available in all operating systems, including mobiles, and modern available browsers. The code of the project is open-source and available in Github too, so anyone can add or request functionalities that are not currently available in the editor.

Finally, regarding the possible extensions for this editor, more advanced features can be developed like including more gain related effects or more configurable parameters for the filters, like resonance and gain for all of them, as well as adding more filters like the peaking or notch ones. Another great addition will be to be able to adapt the editor in order to let the users edit the sample prior to its upload to the website. This feature will add even more value to the user experience on the website, giving them a full sound editing and downloading online workflow.

7. References

[1] Choi, H., & Berger, J. (2013). WAAX: Web Audio {API} eXtension. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 499–502.

[2] Smus, B. (2013). *Web Audio API*. Retrieved from http://it-ebooks.info/book/2072/%5Cnpapers3://publication/uuid/F8F580CC-5306-47D0-BEA7-4AFDBD0F5D32

[3] Font, F., Roma G., Herrera P., & Serra X. (2012). Characterization of the Freesound online community. *2012 3rd International Workshop on Cognitive Information Processing, CIP 2012*.

[4] Web Audio API. (last visited 09/02/2020). Retrieved from https://developer.mozilla.org/es/docs/Web_Audio_API

[5] Mahadevan, A., Freeman, J., Magerko, B., & Martinez, J. C. (2015). EarSketch: Teaching Computational Music Remixing in an Online Web Audio Based Learning Environment. *Proceedings of the Web Audio Conference (WAC)*, 0–5.

[6] Fonseca, E., Pons, J., Favory, X., Font, F., Bogdanov, D., Ferraro, A., … Serra, X. (2017). Freesound datasets: A platform for the creation of open audio datasets. *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*, (October), 486–493.

[7] "The Best Audio Editing Software: 11 Audio Editors for Any Situation." [Online]. Available: https://zapier.com/blog/best-audio-editor/.

[8] "The 5 Best Free Online Audio Editors on the Web." [Online]. Available: https://www.makeuseof.com/tag/free-online-audio-editors/

[9] B. Dias, M. E. P. Davies, D. M. Matos, and H. S. Pinto, "Time Stretching & Pitch Shifting with the Web Audio API: Where are we at?" Proc. Int. Web Audio Conf., 2016.

[10] A. Milo, S. Rudan, A. Xambó, G. Fazekas, M. Barthet. "Deliverable D6.12: Report on the evaluation of the ACE from a holistic and technological perspective". Available: https://www.audiocommons.org/materials/

[11] A. Xambó, M. Barthet. "Deliverable D6.8: Report on novel methods for measuring creativity support". Available: https://www.audiocommons.org/materials/

[12] Font, F. The Freesound Blog: 2019 in numbers. Retrieved from https://blog.freesound.org/?p=1084

[13] Pérez, R. (n.d.). Initial usability questionnaire. Retrieved from https://docs.google.com/forms/d/e/1FAIpQLScTs7AMX6wXjqvGbWhz_O4GIxe5qfSIwFxq5FotHVMq7P8UrQ/viewform?usp=sf_link

[14] HTML. (n.d.). https://en.wikipedia.org/wiki/HTML

[15] Bootstrap. (n.d.). https://getbootstrap.com/

[16] CSS. (n.d.). https://en.wikipedia.org/wiki/Cascading_Style_Sheets