Università degli Studi di Padova
Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Software tools for philological access of historical audio documents

Virtual Magnetic Tape Recorder developed using Web technologies

Candidato:
Francesco Anderloni
Matricola 1034160

Relatore:
Prof. Sergio Canazza

Correlatore:
Prof. Antonio Rodà

Anno Accademico 2013–2014

Francesco Anderloni: *Software tools for philological access of historical audio documents: Virtual Magnetic Tape Recorder developed using Web technologies,* Tesi di Laurea Magistrale in Ingegneria Informatica, © aprile 2014.

This thesis was formatted with the Tesi Classica LaTeX template, by Lorenzo Pantieri.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SOMMARIO

La conservazione la fruizione dei documenti sonori sono problemi complessi che richiedono competenze multidisciplinari e un approccio rigorosamente scientifico, ma che sono di fondamentale importanza in diversi settori scientifici: musicologico, sociologico, antropologico e tutte le discipline legate all'Ingegneria dell'Informazione. In questo lavoro vengono presentati innanzitutto i concetti alla base dei processi di conservazione filologica del patrimonio culturale musicale. Dopo aver introdotto lo stato dell'arte delle tecnologie Web e le *web application*, vedremo come la loro diffusione e la loro evoluzione, con l'introduzione di funzionalità sperimentali, possano essere punti di forza ai fini di facilitare l'accesso al patrimonio culturale. In particolare, vedremo come esse possano essere impiegate allo scopo di realizzare strumenti software utili all'accesso filologico dei documenti sonori, realizzandone un esempio: il magnetofono virtuale.

ABSTRACT

The preservation and fruition of audio documenta are complex issues which require multidisciplinary expertise and a rigidly scientific approach, but are also of paramount importance in various scientific sectors: musicological, sociological, anthropological and all the disciplines related to Information Engineering. In this work we will first describe the concepts that stand at the very basis of the musical cultural heritage philological preservation processes. Then, after having introduced the state of art of Web technologies and *web applications*, we will show how their diffusion and evolution, with the introduction of powerful experimental features, can be strengths towards the purpose of developing useful software tools for granting philiological access to audio documents, showcasing an example: the virtual magnetic tape recorder.

RINGRAZIAMENTI

Ai miei genitori, che mi hanno sempre sostenuto e mi hanno supportato in questo (lungo) cammino universitario, e alle mie sorelle che sono una parte molto importante della mia vita. Ai nonni, zii, cugini e parenti tutti.

Agli amici di una vita, quelli con cui sono cresciuto insieme, e con i quali ancora oggi intraprendo ancora nuove avventure. Agli amici conosciuti lungo il cammino, a scuola, e a quelli che hanno avuto il coraggio di sopportarmi per ben 5 anni come coinquilino. Ai nuovi amici conosciuti durante il periodo universitario, che mi hanno aiutato a viverlo alla grande.

A tutte le persone che hanno creduto in me, ma anche a quelli che non ci avrebbero scommesso: traguardo raggiunto!

Padova, aprile 2014 Francesco

INTRODUCTION

The history of humanity has been perpetuated in documents of different kinds: books, paintings, sculptures, photos, movies are some examples. All of these constitutes the tangible component of what is usually referred to as *cultural heritage*. *Audio documents* represents a very important part of the cultural heritage; since the introduction of recording techniques, in the second half of the 19th century, a lot of valuable information has been recorded as audio signals on different supports. Nowadays, audio lives mostly in the digital domain: from recording to production to storage, almost every phase of the creation of an audio document is carried out without resorting to physical audio devices. But, during the last century, many audio documents have been produced on physical supports; and while a small percentage was brought to the digital domain, a large part still lives on discs, tapes or even cylinders only, and is endangered by the inherent degradation process of physical goods. This issue is being tackled by the archival community, which not only is trying to preserve as much as possible the original supports, but is also taking action in order to let the information contained in these documents outlive their carriers. Maintaining the cultural heritage alive is also important in order to allow the access to these contents.

The evolution of information technology has been aiding these processes, allowing for more efficient preservation techniques and tools, and the widespread diffusion of the *World Wide Web* has ultimately allowed to make all the information contained in multimedia archives available remotely, revolutionizing the fruition of knowledge.

Web technologies have come a long way since the introduction of the World Wide Web in 1989, and have already proven to be very powerful tools to create all kinds of application. With this work, we will examine how experimental web technologies could allow archive to make accessible not only the audio documents, but also the devices needed to experience a *philological* listening experience. We will also prototype an application of this kind, the *virtual magnetic tape recorder*, describing the tools used, the development process and the features we were able to achieve.

THE FIRST CHAPTER will present the preservation of audio documents, the issues that have arose in the discussion on the philological value of re-recordings and the preservation of musical instruments.

THE SECOND CHAPTER will introduce the state of the art of Web technologies, examining in details the features that assumed a fundamental role in our application's development process.

THE THIRD CHAPTER will present the devices we based our work on, the Studer A810 magnetic tape recorder. We will discuss the requirement analysis for our application and illustrate our project design.

THE FOURTH CHAPTER will show the development process behind the back-end and front-end implementation of the app and briefly discuss the results of qualitative testing performed in order to evaluate the capabilities of future developments.

THE FIFTH CHAPTER will finally present our conclusions on the matter and provide some insights on what possible future applications could be.

THE APPENDIX A will contain code listings for the main components of our application.

1 PRESERVATION OF THE MUSICAL CULTURAL HERITAGE

We refer to all the information and artifacts created, preserved and handed down by humanity during its history as *cultural heritage*. This includes tangible culture (buildings, monuments, books, paintings..), intangible culture (languages, traditions, folklore) and natural heritage (landscapes, biodiversity) [19]. Preservation of the cultural heritage and distribution of information related to it are two of the main goals of UNESCO, a specialized agency of the United Nations. We call *musical cultural heritage* the particular subset of the cultural heritage related to human musical expressions, the results of artistic processes, as well as the influxes of music on other cultural fields such as theater, cinematography, art and so on. This comprises everything from music sheets to the musical instruments, as well as all the *audio documents*, the recordings of musical performances, that allow us to access the listening experience further down in time after its original execution. While the first methods of audio recording date back to the 9th century, when inscribed cylinders were used to play back a composition on a custom-build organ [64], most of the recordings dates to the period included between the introduction of the *phonograph*, in the second half of the XIX century, and present times [56]. This interval also coincides with the period of maximum technological progress, which resulted in a fast evolution of the technology related to the creation of audio documents. In the course of roughly 150 years, the recording support of choice moved from cylinders (Figure 1) to discs (first made of resin, then plastic) (Figure 2) to magnetic tapes (Figure 3) to optical supports and finally digital drives.

The United Nations Educational, Scientific and Cultural Organization (UNESCO) "was created in 1945 in order to respond to the firm belief of nations, forged by two world wars in less than a generation, that political and economic agreements are not enough to build a lasting peace. Peace must be established on the basis of humanity's moral and intellectual solidarity" [33].

The evolution process was so fast that often a method became the de facto standard even before actual standards could be introduced for the previous one. The main advantage of the rapid evolution of recording methods and supports is the fact that this allowed for progressively greater sound quality and less expensive production of supports and recording/replaying equipment, paving the way to mainstream diffusion of recorded music. The main disadvantage is that a great quantity of documents remained "stuck" in an older format and their perpetuation was not followed through, causing their progressive extinction and sometimes even their disappearance. It is now evident that something has to be done to maintain the cultural heritage as alive as possible in order to make its fruition available to future generations. For some parts of it, a lot of work has already been done: books and manuscripts have been preserved in libraries and transcribed to make them available in digital format; art is col-

Figure 1: A wax phonograph cylinder. Source: [55]

Figure 2: A vinyl disc. Source: [73]

Figure 3: A magnetic tape cassette. Source [38]

lected in museums and each piece of art is thoroughly analyzed with modern techniques in order to extract as much information as possible to extend its preservation. For musical cultural heritage, the discussion on what needs to be done is still relatively young. The most efficient solution proposed involves the use of modern technologies - specifically, computer science and engineering.

1.1 PRESERVATION OF AUDIO DOCUMENTS

Audio documents represent a big chunk of all the information that forms the music cultural heritage. Nowadays we are used to think of audio recordings as digital files, usually obtained through the Internet, which are easily playable on a great number of devices and are always accessible. As we saw, though, this has been just the last step in an evolutionary process that brought Information Technology in the digital domain. Before that, music recordings have always been associated with some kind of *physical support*, and therefore the vast majority of audio documents still exists in physical form. These objects are subject to the unavoidable natural degradation of their physical qualities, and this fact endangers the information they carry. It is clear then that *audio documents preservation* is necessary. *How* to perform this preservation, though, is still a very discussed topic. First of all, let us define the two main ways an audio document can be safeguarded:

A simple scratch on the surface of a vinyl disc can make it unplayable; oxidation of the reflective layer on optical supports such as CDs and DVDs will occur even if they are kept at rest, causing data loss.

1.1.1 Passive preservation

Direct passive preservation involves the actuation of preserving rules concerning the storing and access of physical, original copies in order to avoid the main causes of physical carriers deterioration. These causes are for the most part common to all types of supports, but the solutions proposed are different, due to the ways the various types of support handle the effects of each deterioration agent. Typical causes of deterioration of physical supports include humidity, temperature, dust and dirt deposits, mechanical deformations and mold. It is in the best interest of the archive to conserve physical copies of audio document in a controlled environment where the effects of these agents are continuously kept under close observation in order to limit their (inevitable) impact on the supports' quality and conservation.

1.1.2 Active preservation

Active preservation involves the implementation of a precise and defined *re-recording process* in order to produce copy of the document on a new support (possibly on a different *medium*), paired with an extensive description of any useful data related to the document itself. This new copy will then substitute the original one in every subsequent access in order to preserve the original document as much as possible, as well as carrying most of the information relative to the original one when that will not be longer playable.

Direct passive preservation can be carried out only if the main causes of the physical carriers deterioration are known and consequently avoided [6], and therefore requires knowledge of physics and chemistry involved for each different type of support. Standardized procedures have been proposed and discussed during the years and are now currently employed in archives all over the world.

Active preservation is a much more complex topic. The same physical and chemical knowledge required for passive preservation is needed, but a more technical understanding of all the theoretical and practical aspects of sound recording is also necessary, along with musicological and historic-critical knowledge, essential for the individuation and correct cataloging of the information contained in audio documents [6]. All these competences would ideally be personified by a unique figure, the so-called *Tonmeister*, but it is usually more easy to have them spread across a multidisciplinary team of experts. This is one of the main factors that obstruct the safeguard of audiovisual documents: teams like these are difficult and expensive to form, and the active preservation process, in order to be carried out correctly, requires time and economical resources. On the other hand, it must be kept in mind that active preservation will produce copies of the audio documents that one day will substitute the original ones, therefore it must be executed correctly in order to preserve as much information as possible.

The re-recording process has also been the focus of a fervid discussion in the last thirty years, and just recently the audio preservation community has started to agree on the methods and tools to use. It is now clear that the remediation process (the operation of transferring the information from the original medium to another one) should produce a copy of the document in the *digital* domain, since other types of archival media are subject to deterioration themselves (requiring the repetition of the re-recording process multiple times), and impose additional difficulties to the access to the information. The discussion has culminated in the two concepts "*preserve the content, not the carrier*" and "*distribution is preservation*" [6].

1.2 PHILOLOGICAL AUTHENTICITY OF AUDIO DOCUMENTS

To fully preserve all the information contained in an audio document, both passive and active preservation are needed. The digital copy obtained through the re-recording process is also used as a master to produce access copies. This poses particular attention to the methods and tools used in the process: unfortunately, this is subject to electronic, procedural and operative errors, and indulges in the aesthetic changes of the current times. Therefore, total neutrality in the process of the information transfer is not realistic, putting the spotlight on the *philological problem* of the documents authenticity [6]. First of all, in order to preserve as much information as possible, a standardized re-recording process should be outlined and presented to the community for discussion. The discussion should be based around some fundamental philological aspects from the literature, which describes the levels of interpretation in the re-recording process and their effect on the philological authenticity of the new copy. Then, more details about the actual techniques and tools can be introduced in order to obtain consistent results. This is exactly what happened in the last decades, with the discussion on active preservation starting with fundamental concepts of philological value to details regarding the standardization of procedures and tools.

1.2.1 Two Legitimate Directions

Storm at that time was Assistant Director of the Thomas A. Edison Re-recording Laboratories at Syracuse University Libraries [6].

In 1980, with an article titled "*The establishment of international re-recording standards*", William Storm tried to define a solution to address the lack of standard procedures for audio restoration. Storm stated that two *legitimate directions*, or types of re-recording, are suitable from the archival point of view:

1. one that would permit reproduction "*as the perpetuation of the sound of an original recording as it was initially reproduced and heard by the people of the era*" [67], highlighting the double documentary value of re-recording by proposing an audio-history sound preservation. This method puts the historical conditions and technology of the era in which the recording was produced in the spotlight, adding information related to the quality of recording and reproducing systems of the time, all with the intent of offering a historically faithful reproduction of the audio signal: "*how records originally sounded to the general public*" [67].

2. the other would imply a deeper level of restoration to obtain "*the true sound of an artist*" and "*the live sound of original performers*", permitting the use of "playback equipment other than that originally intended", as long as it is being proved objective, valid and verifiable by the researcher [54].

The contents of the article brought attention to the problem, eventually generating a discussion that was set to span well over two decades before any sight of a unified solution.

1.2.2 "To save history, not rewrite it"

Dietrich Schüller is a key figure in the audiovisual archival community, and is currently member of a number of Austrian and international organizations, including the Intergovernmental Council for the Information for All Programme of UNESCO, of which he is vice president [59].

Following Storm's work, Dietrich Schüller formulated new guidelines for re-recording based on a preliminary analysis of the *aim* of the process, obtained from a deep study of the technical and artistic content carried by the recorded composition. These guidelines were subsequently incorporated in an official document commissioned by UNESCO, titled "*Safeguarding the Documentary Heritage*" [5]. The guide follows the philosophical approach "*save history, not rewrite it*": the re-recording process should guarantee the best quality possible for the restored signal while limiting the audio processing to the minimum, in order to preserve the artistic vision of the author. The analysis should start with an accurate investigation of signal alterations, categorized in:

- *intentional*, due to an active action taken by the artist during or before the recording process, such as equalization and noise reduction systems;

- *unintentional*, either effects of the imperfection of the recording technique of the time or caused by misalignment of the recording equipment.

Then, a re-recording strategy should be established: either follow a similar approach to Storm's first *legitimate direction*, in order to obtain a representation of the audio signal as perceived by the people of the era (*Type A*), or manipulate the re-recording process to get as close as possible to the sound of the recording as it was produced, precisely equalized for intentional recording equalizations, compensated for eventual errors caused by misaligned recording equipment and replayed on modern equipment to minimize replay distortion (*Type B*) [5]. It should be noted that *Type B* re-recordings, while producing an historically faithful level of reproduction that is preliminary to any further possible processing of the signal, require the use of compensation derived from knowledge external to the audio signal, therefore a certain margin of interpretation of the document is required, both from historical acquaintance with it and from technical-scientific knowledge [6]. Finally, a third type of re-recording is defined as *Type C*, and sets the guidelines for obtaining an historically faithful reproduction of the recording "as produced, but with additional compensation for recording imperfections caused by the recording technique of the time" [60]. These compensations relate strictly to the area of equalizations to correct nonlinear frequency response caused by imperfect historical recording equipment and must be rigorously documented by the restorer in the reports which accompany the restored copy.

1.3 THE REMEDIATION PROCESS

Many different proposal for a standard re-recording methodology have been introduced and discussed between the end of the 20th century and the beginning of the new millennium. Some have been obtained as a result of years of refining consolidated archival practices; others have been created from scratch, keeping in mind the fundamental discussion at the very basis of the problem, to take advantage of the newest technologies in the field. An example of consolidated re-recording and active preservation process is the one proposed and used by the re-recording team of the Centro di Sonologia Computazionale of the Università degli Studi di Padova. We will briefly discuss the fundamental aspects on which this particular process is based as an example of the work involved. First of all, the team must discuss the philological and choose one of the approach. Schüller's *Type B* is usually the recommended choice as it allows a good degree of objectivity, since the compensations needed that are external to the audio signal are obtainable from historical acquaintance with the document being processed along with technical-scientific knowledge retrievable from the history of audio technology - topics that should be well-known among the members of the team [6]. Then a destination medium should be chosen: the current best solution is a transposition to the digital domain, which not only allows the conservation of the new copy in a truly non-degradable support, but also allows the storage of *secondary information* related to the document along with the audio signal and facilitates the access to the document itself, thanks to the widespread diffusion of digital communication technologies. A step-by-step operation protocol must then be outlined, describing in details the remediation process, articulated in procedures and sub-procedures, that in the end will produce the preservation copy, the organized data set that groups all the information represented by the source document, stored and maintained as the preservation master [6]. Figure 4 depicts the remediation process, in which three distinct steps can be observed; each set can be split up in several different sub-steps - for example step 1, "*Preparation of the carrier*", can be composed of a first procedure of physical documentation gathering, followed by a visual inspection and a chemical analysis of the support, and concluded by an optimization of the carrier based on the results obtained from the previous sub-steps.

Secondary information includes photo and videos of the original support and the re-recording process, as well as every bit of information that could be useful for the philological fruition of the document.

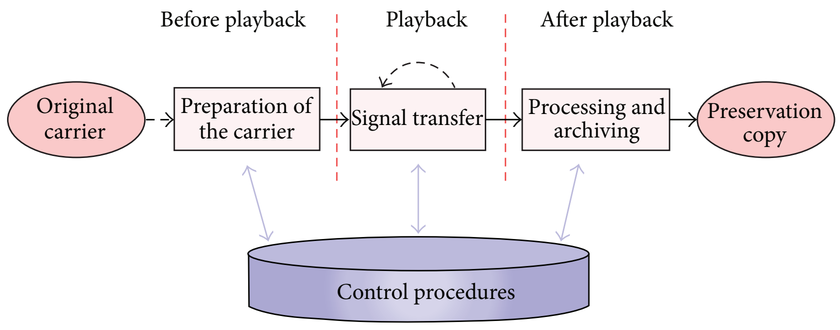


Figure 4: Operational protocol for the remediation process. Source [6]

The input of the remediation process is an audio document, and the expected output is its preservation copy, along with the source document ready to be stored again. If each step is carried out according to the operational protocol, the preservation copy will fulfill the requirements of accuracy, reliability, and philological authenticity [21]. Authenticity is, in fact, the result of a process, it cannot be evaluated by means of a boolean flag, and it is never limited to the document itself but extended to the information/document/record system [24]. Finally, particular tools or techniques can be suggested for different use cases, explaining in which step of the protocol they should be incorporated and how they should be used. Sometimes these tools are developed by the team members itself in order to exactly fulfill the specific requirements of the re-recording process.

1.4 PRESERVATION OF MUSICAL EQUIPMENT

Let us think about not only the music generated by the use of instruments, but about the instrument themselves. How should the instruments be preserved? They are physical objects, just as the supports that store music - so one could argue *passive preservation* is needed. This consideration is absolutely right, but poses the same issues we analyzed with passive preservation of musical recordings: sooner or later, no matter how good the preserving process is, the instrument is going to deteriorate, eventually losing the information it carries.

This means it is necessary to delineate an *active preservation* process, a way to define and store the information embedded in the instrument in order to make it easily replicable within a certain approximation. The problem here, though, is much more complex: it is not sufficient to extrapolate the signal from the support and let it live in digital form - it is necessary to preserve and recreate the whole structure of the instrument, starting from its physical components. This is a work that requires specific knowledge, adequate funding within institutional frameworks and interdisciplinary collaboration among several experts in the field [4].

While recordings of sounds and musical performances have been around since the end of the 19th century, and re-recordings techniques and standards have been discussed since the 1980s, active preservation of instruments is a much younger problem, especially for *electroacoustics* instruments, which have started to gain significant cultural importance only in the second half of the 20th century. Awareness towards this complex issue has raised in recent years, with the launch of preservation projects such as DREAM , funded by the European Union, with the goal to save at least a part of the huge cultural heritage of electroacoustic music.

The Digital Re-working/Re-appropriation of Electro-Acoustic Music is a EU funded project, aimed at preserving, reconstructing, and exhibiting the devices and the music of the Studio di Fonologia Musicale di Milano della Rai [20].

1.4.1 Active preservation of electroacoustic music

We define *active preservation* of electroacoustic music as the set of all the actions aimed at keeping alive the musical compositions, by transferring the recordings and the instruments to the digital domain, allowing performance and functionality both for musicological research and for philological analysis [4]. We already discussed about preservation of musical compositions, so in this part we will focus about two processes strictly related to the instruments: *preservation* and *restoration*. The first is related to the actions aimed at maintaining cultural heritage in its original form, while the second is related to the actions aimed at making cultural heritage available, following subjective aesthetic principles [4].

Electronic instruments are usually just a part of the cultural information related to a composition: often, they are accompanied by textual and graphic materials (scores, schemes), audio documents and software. All of these must be preserved as well, both actively and passively, to minimize the loss of the information related to the electronic music composition. This, however, is relatively easy once a standardized preservation process has been designed. The hard part is defining a solid process aimed at transferring the information carried by electronic instruments in the digital domain.

First, let us examine the different types of electronic instruments, and how their inherent features may relate with the restoration process. Electronic instruments differs from traditional ones in many ways: the use of electric energy as the main sound producing mechanism, rapid obsolescence, the dependance on scientific research and available technology [10]. It is useful to group different types of electronic instruments in three categories, based on their characteristics:

1. **Electroacustic instruments** transform the acoustic pressure signal generated by the vibration of a body into a voltage variation. The sound is subsequently processed through an amplification system;

Figure 5: The *microphone* is the most immediate example of electroacustic instrument. Source: [41]

2. **Electromechanical instruments** transform the effect of electromechanical, electrostatic or photoelectric phenomenons generated by the motion of a tape or a disc into voltage variations. In this case amplification is *needed* to make the sound audible;

3. **Electronic instruments** generates sounds directly by using electronic components - no acoustic or mechanical vibration is needed. These instrument evolved with the scientific progress regard-

Figure 6: A turntable, or *phonograph*, is an electromechanical instrument. Source [56]

ing electronic components during the course of the last century, gaining the ability to synthesize sounds by using valves, semiconductors, integrated circuits. They also became more and more complex with the years, combining more *elemental* parts such as oscillators and filters to produce a modular, programmable and interactive system.

For each of these types, several hundreds or even thousands of models have been designed and manufactured, and millions of copies already populate the Earth. Some of those have been released commercially, produced on a larger scale and made available to the general public, while some other existed just as prototypes or personal experiments of dedicated musicians and engineers. From this point of view, the preservation process must include as much information as possible about the design and building process of the instrument, a task that may be easy enough in the first case, given the availability of official documentation and specifics by the manufacturer, but definitely harder in the second one. Maintaining original units functional through the years would also be more problematic since the components used in the devices are usually custom, or even reused, older components taken straight from other devices.

Reusing electronic components subtracted from other devices is a practice known as cannibalism *[4].*

In light of the above discussion, we propose to transpose the categories of passive preservation and active preservation - usually applied to document preservation - to the field of electrophone instruments. In this context, passive preservation is meant to preserve the

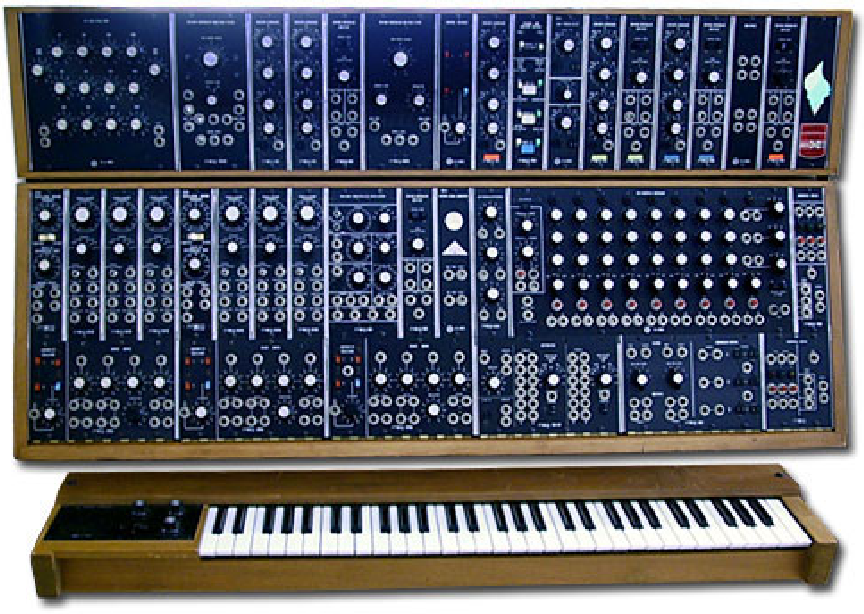


Figure 7: The *Moog Modular* synthesizer, an example of electronic instrument. Source: [68]

original instruments from external agents without altering the original electronic components, while active preservation involves a new design of the instruments using new electronic components or a virtual simulation of the instrument [4].

As we saw with the preservation of audio documents, passive preservation is essential for audio equipment too, and every possible measure should be taken in order to preserve the original instruments as much as possible, especially when it comes to custom-built devices that would otherwise disappear from the cultural heritage. That said, active preservation becomes important too not only because it is the only way to maintain the knowledge of those instrument after they inevitably disappear, but for the fruition of that knowledge. Active preservation with modern techniques, as discussed in this work, may allow a wide number of users to interact with the replicas, effectively gaining access to all the information associated with the instrument, while avoiding to put additional stress on the already frail original device. The discussion on active preservation of physical devices is very much open nowadays - detractors argue that it is a pointless practice due to lack of a standard process of emulation and archiving, the inherent loss of information due to approximation of the replicas and their rapid obsolescence due to rapid improvements in technology. On the other hand, no other viable solution to the problem has been proposed and active preservation, if done right, is actually reputed to be a desirable solution by many experts in the field [4].

1.4.2 Active preservation of electrophone instruments through virtualization

In the previous section we have seen why the active preservation of an instrument is far more complex than the active preservation of recorded music. The fact that the most interesting cases of instruments to be preserved are often built from original designs of expert musician and technicians, combined with the use of different electrical components and the lack of documentation, make this task even more daunting. Sure, the specific function of a given instrument could easily be understood and replicated with modern tools, analogue or digital, but this procedure would generate a *loss of information*: all the imperfection of the original instruments are in fact addictions to the character and timbre of the sounds it generates, and they have to be replicated as accurately as possible, in order to preserve its identity. What is important to notice is that the fundamental concept on which active instrument preservation is based is *sound synthesis* - the same concept that makes the generation of sounds from electronic instruments possible. In this case, just like for recorded music, what is necessary to do is transpose this concept in the digital domain: we are now talking about *virtualization* of musical instruments. There are different ways to synthesize a sound, and all of them translate into different ways of virtualization. The are two main *models* used to represent and manipulate sound synthesis:

1. The **signal based model** utilizes simple generators (oscillators, noise generators, waveform generators) that are subsequently combined with other components able to apply linear and non-linear transformations to the signal. The sounds outputted from the generators can be totally synthetic (e.g. pure sinusoid, white/brown/pink noise...) or waveforms obtained by *sampling* real sounds. For example, the famous *TR–909* drum-machine (figure 8), developed by Japanese electronic equipment manufacturer Roland Corporation in the early 80s, was capable of generating ten drum sounds, seven of which (*kick*, *snare*, *toms*, *clap* and *rim shot*) were synthesized from oscillators and effects, while the remaining three (*hi-hats* and *cymbals*) were obtained through digital sampling of actual instruments [17, 58]. The transformation techniques can be *time-based*, if they manipulate the generators' output samples in the time domain, or *frequency-based*, if instead they operate on the frequency spectrum of the signals with operations like digital filtering. This model is relatively simple to implement and manipulate, and has the advantage of offering complete control to the operator - its main con is that it is quite limited in its capabilities.

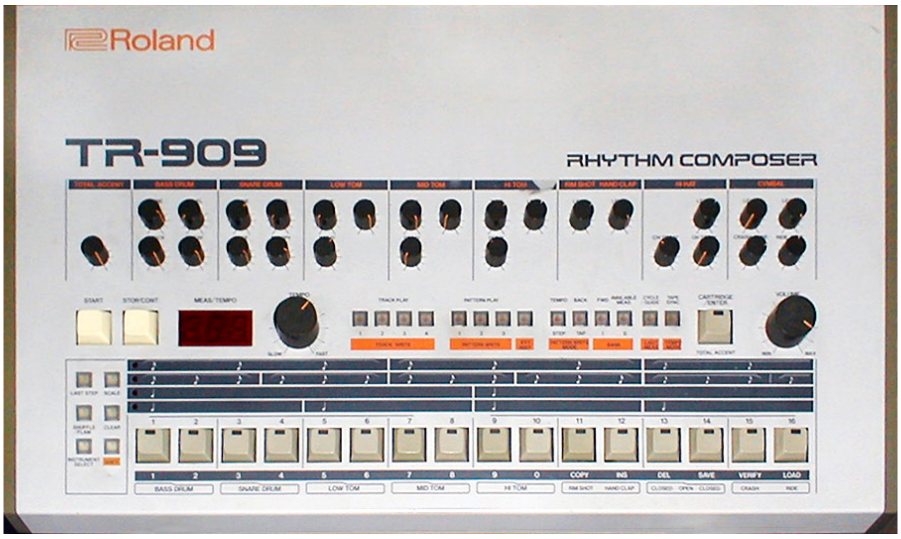


Figure 8: Roland TR-909 drum machine. Source: [58]

2. The **physical based model** is based on the actual physical interaction that happens between the components of an instrument. While the signal-based model is used to describe the sound itself, as it is perceived by the listener, the physical-based model utilize physics notions to describe the *source* of the signal, and the way it behaves to generate a particular sound. For example, a guitar could be modeled by providing a system that describes its vibrating strings and its resonating body, with equations that permit to obtain the output sound based on the insertion of input parameters (i.e. description of the strings plucking). This method is rather complex and requires the calculation of large number of parameters in real time: for this reason it has not been possible to realize an efficient implementation until recently, when the computing power of CPU became sufficient to quickly calculate the large number of values that constitute the numeric solution of the physical model's equations.

As we said, an accurate replication of all the dynamics involved in the generation of a sound from a custom analogue device in the digital domain should include possible imperfections and erratic behaviors, since they are part of the identity of the instrument itself. For this reason, the signal-based model often isn't enough flexible for a faithful virtualization. Implementing each component of the instrument with an accurate physical model, on the other hand, could make the replication of the exact physical interactions happening during the generation of the sound, and is therefore the preferred method. In cases where the sound-producing (or sound-processing) device to be simulated is an analogue electronic system, rather than a mechanical or acoustic system, the term "virtual analogue synthesis" is commonly used to refer to physically-based sound synthesis of these devices [4].

1.4.3 Interaction with virtual devices: control parameters and interfaces

One of the advantages of using the physical-based model to realize the generation of sound in virtualized instruments is that *control parameters* (i.e. the variables that contribute in producing a variation of the sound) are *hard-wired* in the sound synthesis scheme, as it is part of the equations that originated the synthesis algorithm. This allows a more natural interaction with the virtualized instrument itself, provided that an *interface* resembling the original device is present.

We call *user interface* the part of an instrument which interacts with the player in order to produce a particular sound [61]. For traditional instruments, the user interface is usually immediately recognizable as it has been developed over a long time and it has entered common knowledge. For electronic instruments, interfaces can be more complex and their comprehension is usually not as immediate as with traditional instruments. There is not a direct correlation between the *shape* of an electronic instruments and its sound, nor there are *standardized* ways of accessing a particular control. Some electronic instruments partially resemble traditional instruments: for example, a great number of *synthesizers* sports keys just like a piano that achieve the same exact effect (play a determinate note with a certain velocity), though they are usually accompanied by a multitude of other controls to manipulate the sound's parameters. There are *knobs*, *switches*, *faders*, *pads*, *resistive* and *capacitive touch sensors*... Each of those has a particular function - we say there is a *mapping* between that part of the interface and a parameter in the sound generation. For example, if a knob controls an oscillator the mapping is represented by the connection between the motion of the knob and the behavior of the oscillator. Mapping is a crucial part of designing the interface of an instrument: it is extremely important for *mapping strategies* to be natural, understandable and intuitive. It is also important that the action performed by the user provides an immediate *feedback*, be it auditory, tactile or visual [61].

In a piano, for example, the interface consists of the keys and the pedals that allow the musician to play a certain note. The whole piano itself is part of the interface in the sense that it produces the sound.

Twisting the knob that controls the frequency of an oscillator can provide both a tactile and a visual feedback (the sound changes in real time), as well as a visual feedback depending on the interface (for example, if there is a display showing the oscillator frequency in Hertz).

When we consider virtualized instruments, it is important not only to replicate the sound of their original counterparts, but also build an interface that can provide a comparable experience for the users. Interfaces for virtual instruments can be made with physical components similar to the ones used in the original devices, laid out in an accurate replication of the original design, and providing the same mapping and the same feedback. In this case, the only difference between the original instrument and its virtual version would be the lack of the original internal components, emulated by the physical-model representation running on a calculator. If it is possible to "hide" the calculator behind the interface, then it is possible to obtain a virtual device that sounds, looks and *feels* just like the original one, which is the full accomplishment of the goal of researches on this subject. Sometimes, though, it is not possible (or it is simply easier) to realize a physical interface for virtual instruments. In these cases a *virtual interface* is then provided: an implementation of the mapping of the instrument's parameters that also resides on the calculator running the emulation. This implementation can be visualized on a screen, and the parameters can be manipulated using the calculator's standard input methods (mouse, keyboard, trackpad. . .) or devices more suitable such as MIDI controllers. This is the case for a great number of commercial products that emulate the sounds of classic electronic equipment for use in digital music production - the so-called *plug-ins*. These pieces of software bundles together the sound generation virtualization and the interface emulation and can be used in *Digital Audio Workstation* software. Usually the software house that develops a plugin works closely with the original manufacturer in order to get every aspect of the virtualization right; for the interface, a *skeumorphic* approach is often used, replicating the original design of the instrument in detail. Figure 9 shows a classic vintage synthesizer, the *Minimoog*, produced by Moog Music between 1970 and 1981. Its plugin version, commercialized by software house Arturia with parternship with Moog Music, is shown in figure 10. All the original controls that constituted the original interface are immediately recognizable in the virtual interface.

There are different standard used in the production of plugins, the most notable ones being Virtual Studio Technology *(VST, [74]) and* Audio Units *(AU, [3]).*

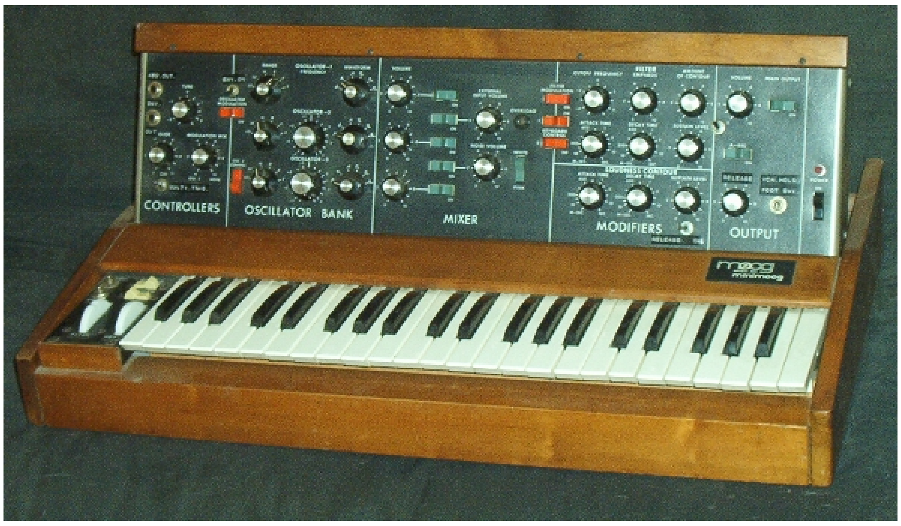


Figure 9: The *Minimoog* synthesizer. Source: [42]

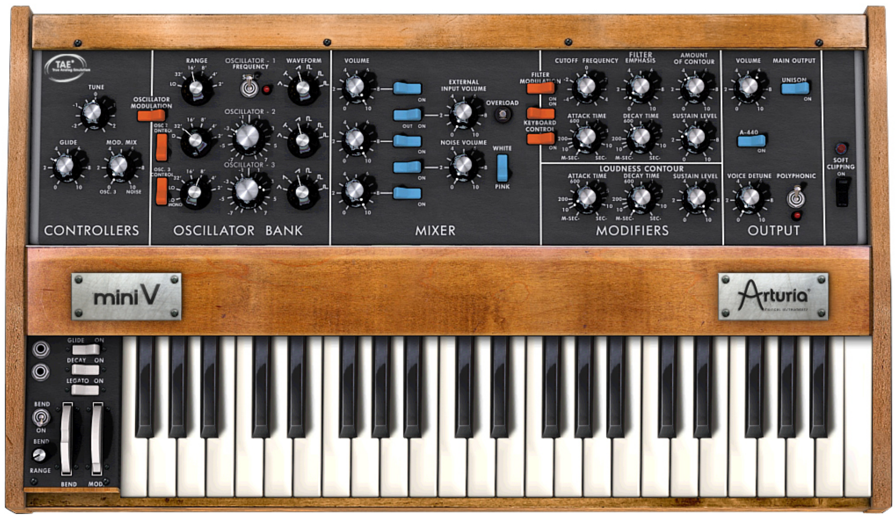


Figure 10: Arturia's *MiniV* plugin. Source: [43]

1.5 INSTRUMENTS FOR PHILOLOGICAL ACCESS OF AUDIO DOCUMENTS

There is a particular subset of audio equipment which undertakes additional importance from the functionality they provide: *recording* and *replaying systems*. They are important pieces in the creation and fruition of audio documents, and their impact on the signal being recorded or reproduced is definitely not negligible. In fact, as discussed in section 1.2 , these instruments, being part of the original recording processes of musical performances of all kind, become part of "*the sound of the artists*" itself, other than being a reflection of the technological capabilities of the era. As we saw, a lot of audio documents, before being subject to active preservation measures, exist only in physical form, most of the times on a particular type of support or format - discs, tapes, optical supports etc. Each different format requires a different replaying system, and sometimes even different versions of the same format requires completely different systems. Vinyl records, for example, are meant to be reproduced at a fixed rotation speed, and therefore require a phonograph able to operate at that specific speed. Recording and replaying standards have been introduced during the years for all kinds of different supports, but for practical reasons their number is still fairly large, and this caused the design and commercialization of many different instruments. While one could argue that these pieces of audio equipment have less cultural value than actual electronic instruments that produces distinct sounds, and they are not as endangered as custom-build electronic instruments due to their more widespread diffusion and more efficient conservation (the most important are still functional and actively used in recording studios all over the world), they are subject to the same deterioration problems discussed in the previous section and therefore should be actively preserved as well. Adding the fact that nowadays recording technologies and techniques are progressively shifting towards the digital domain, with analogue equipment being used less and less by professionals in the industry, and that physical replaying systems are almost extinct due to the diffusion of digital audio files that can be reproduced on a variety of multi-purpose devices, the issue at hand becomes even more relevant. Active preservation of this class of instruments then acquires notable relevance not only for the reasons discussed in the previous section, but also as a way to make these (often rare and expensive) instruments more accessible to the general public, allowing a truly philological access to audio documents. In fact, the digital access copy of an audio document could be paired with the digital emulation of the instrument(s) that would be required to reproduce the physical copy of the document revolutionizing the access procedure for archives, and most notably, eliminating the need to locate both the original support and the replaying system.

2 WEB TECHNOLOGIES AND WEB APPLICATIONS

The first *web applications* were introduced to take advantage of the widespread diffusion of the Internet and the constantly increasing capabilities of web browsers, which allowed the evolution of *Web technologies* both on the server and the client side.

Nowadays, with Internet connectivity spreading out to more and more devices, the possibility to develop a multi-platform application that can leverage the potential of Web standards and offer a rich experience while providing a uniform user experience across different devices is a crucial decision factor in the design and development of applications of all kinds.

The standardization of some of these technologies by overseeing organizations coordinated by the *World Wide Web Consortium*, paired with the constant efforts of the developers community, keeps the interest in producing new pieces of software that enhance the capabilities of web applications high, and allows the introduction of new features that can subsequently become largely adopted.

In this section, we will define the general aspects that characterize *web applications*, as well as introducing the main Web technologies that were vital for the development of our project, the *HyperText Markup Language revision 5* (HTML5), which introduced experimental features related to audio management in the browser we adopted to satisfy our project requirements, and *Node.js*, a back-end framework for web applications.

2.1 WEB APPLICATIONS

A *web application* or *web app* is any application software that runs in a web browser or is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a common web browser to render the application [79].

Usually, a web application is composed of several different *tiers*, each assigned to a different role in the application's structure. By definition, the *presentation tier* is managed by the web browser through a *rendering engine* which translates client-side code to a user interface. The browser can also run client-side business logic to modify the content on the page or to communicate with a server, usually using a scripting language such as JavaScript. The other tiers usually reside on a remote infrastructure which interacts with the client in order to provide data (*storage* tier) and perform calculations (*business logic* tier).

This general definition is by no mean always applicable, as there can also be a various number of intermediate tiers performing different tasks.

The simplest type of architecture that can be realized with this tier structure is the so-called "dumb terminal", where all the elaborations are made on the server and the client just provide an interface for visualizing results and issue commands. Given the capabilities of modern web browser, the unusual "dumb server" architecture could also be implemented, with all the elaboration done client-side and the server performing only as data storage. These two, though, are limit cases and generally web applications divide their load on both the client and the server, to optimize execution and communication performance [79].

Web applications exploit the ever-growing capabilities of browser engines and web technologies to provide rich user interfaces on the client side. For this reason, the design and implementation of the client side of the application are realized using HTML and CSS for content organization and displaying and JavaScript for dynamic scripting, being these the standard for the Web industry.

Implementation of server-side business logic, on the other hand, is possible with a wide selection of different technologies and programming languages, ranging from solid enterprise solutions to independently developed experimental frameworks. The same goes for data storage, with different types of databases and management systems interoperable with almost any business logic structure thanks to the work of programming companies and independent developers.

For both these aspects, it is interesting to note the impact of the *open source* culture in software development, which has made the creation and maintaing of important pieces of software possible. The evolution of the so-called *Cloud Computing* also helped the shift towards web applications and web services, with IaaS ("Infrastructure as a service"), PaaS ("Platform as a service") and SaaS ("Software as a service") functionalities becoming easier and cheaper to adopt [12].

2.1.1 Advantages and disadvantages of web apps

The only requirements for running a web application are a web browser and the presence of an active network connection, both vastly diffused. Web browsers exist for a great number of different hardware and software platforms, making web apps cross-platform compatible "out of the box".

In 2013, the number of devices connected to the Internet exceeded 10 billion. Analysts expect the number of connected objects to reach 50 billion by 2020 [13].

Since there is no "installation" process, web apps are usually easier to use and set up even for inexperienced users. The fact that upgrades, maintenance and introduction of new features are performed on the server and automatically rolled out means that the end users don't have to worry about keeping the app "*up to date*". On the other

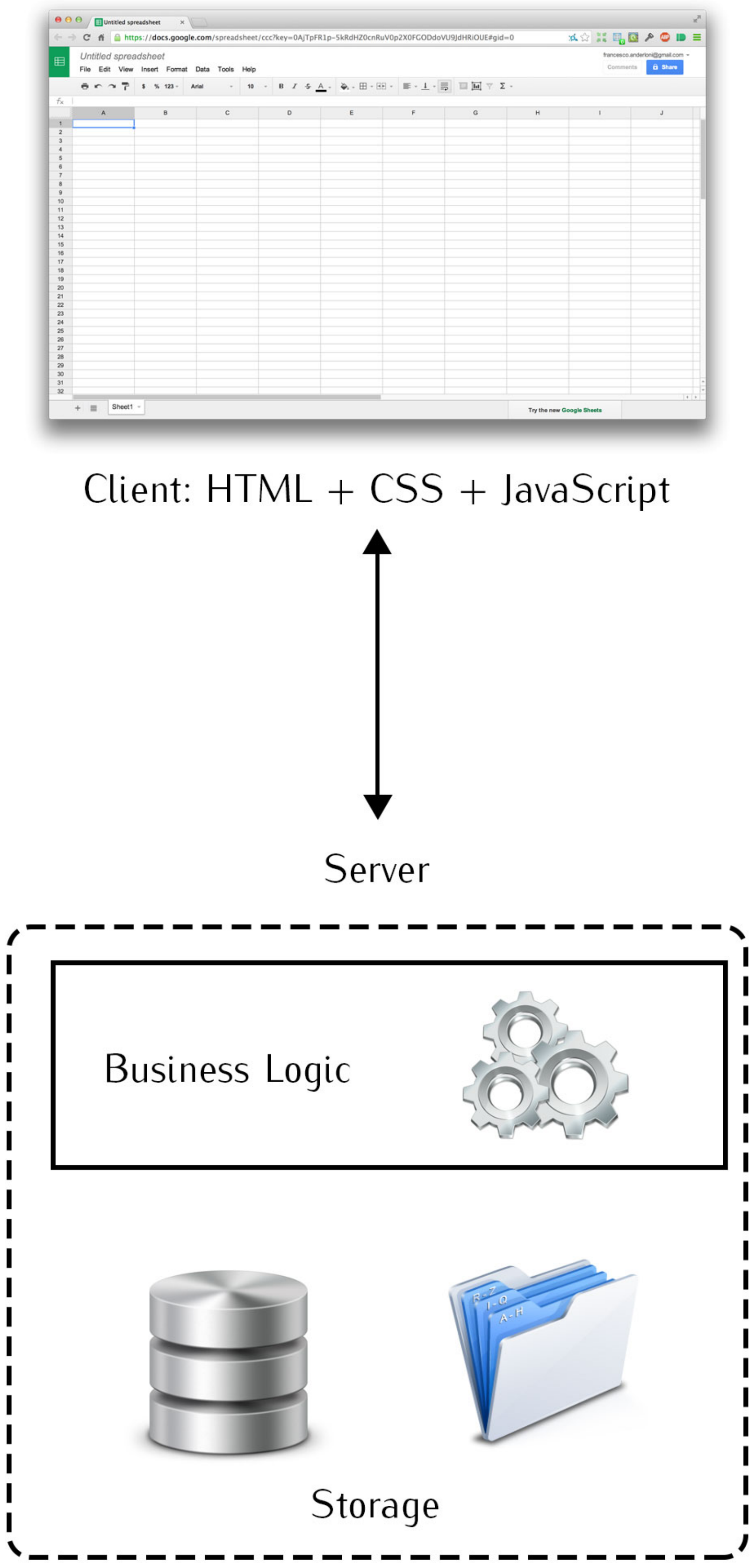


Figure 11: Representation of the architecture of a web app.

hand, they have to make sure to use a recent version of their web browser of choice in order to be able to support all the newest features provided by the Web standards. This is an issue that proves to be more serious than one would expect, therefore web apps developers often have to make sure that their applications are supported by the vast majority of web browsers, including outdated ones, by disabling advanced features or recurring to *fallbacks*.

Another big advantage of web applications is the fact that most of the application data and computation usually resides on the server, making them perform decently even on outdated client hardware; since the data is stored remotely, syncing functionalities can be introduced to make it accessible from different devices.

Even with the introduction of various solutions to overcome the "always online" requirement, though, web applications inherently fall short when there is no connection available. Despite the continuos evolution of web technologies it is still difficult to reach the level of efficiency and usability of *native* applications (i.e. programs developed for a specific platform). Another disadvantage is the loss of control on customization and the lack of flexibility for the end users, which are forced to adopt each change introduced by the developer(s), since the code resides on the server. For these reasons, web applications users are often susceptible to unwanted modifications, creating a volatile market where it is hard for companies to retain a satisfied user base.

Famous web applications such as Facebook *and* Youtube *experienced negative feedbacks from their users after interface layout changes [23].*

2.2 WEB TECHNOLOGIES

With the term *web technologies* we refer to all the software and hardware components that allow the communication between two devices over a network. Client-side, web technologies mostly comprise of software tools and frameworks which leverages on the three main building blocks of the Web, the *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) and the *JavaScript* scripting language. The first have sustained an increasing standardization process overseen by the *World Wide Web Consortium*, an institution composed by member organizations which maintain full-time staff for the purpose of working together in the development of standards for the World Wide Web [78].

In addition, a great number of third party tools and libraries has been developed by third party organizations and independent programmers, and has been released either as commercial products or distributed free of charge on the Internet. Notable examples include *jQuery*, a JavaScript library to facilitate manipulation of the DOM and add functionalities such as AJAX, which is open sourced and released for free under the MIT License [35], CSS frameworks such as *Boot-*

Asynchronous JavaScript and XML *is a technique used to create asynchronous web apps [1].*

Figure 12: The official HTML5 logo, and two unofficial CSS3 and JavaScript logos.

strap, maintained by Twitter's developers [71], and tools to aid the application design and development process such as the *Dev Tools* included in Google's web browser *Chrome* [11].

On the server-side, technologies are implemented using a variety of programming languages and design models. Languages such as PHP, Ruby, Perl, Python, as well as Enterprise Java (J2EE) and Microsoft.NET Framework, are used by developers to output data dynamically using information from files and databases [79]. Server-side web technologies include back-end frameworks, such as *Ruby on Rails* or *node.js*, server software, such as *Apache Tomcat* or *nginx*, database management systems, middleware for handling the coordination of the application over distributed servers, and many other pieces of software, as well as optimized runtime environments and operating systems tied to the hardware implementation of the server's architecture. Giving a thorough description of all these is difficult given the great number of components that could be catalogued as "Web technologies"; therefore, in the following sections we will discuss only the ones that were vital in the development of our project, mainly HTML5 for the client side and Node.js for the server side. Other third party tools and frameworks will be shortly described with their use cases in chapter 4.

2.2.1 HTML5

HTML5 (HyperText Markup Language version 5) is the fifth revision of the HTML standard, which defines the markup language used to structure and present content for the World Wide Web. HTML allows users to explicitly declare how the content present on their *webpages* should be organized in a standardized way thanks to use of *HTML elements*. These elements are represented by tags, keywords enclosed in angle brackets, that commonly appears in opening-closing pairs which holds part of the content: for example, the code line

```
<h1>This is a header</h1>
```

The backslash ("/") in the second `h1` tag makes it a closing tag, while the first `h1` tag is the opening tag of the pair.

shows a `h1` tag which encloses the phrase "*This is a header*". The `h1` tag is used to indicate that the content included between the opening and closing tag is a header, and it is just one of many kinds of tag available to structure and organize content on the page. Tags are used to standardize the representation of the document: each web browser will render the page in the same way, i.e. a `h1` header will always appear as a header.

HTML tags allow the use of multimedia content (such as pictures, audio and video files) and *hyperlinking*, the use of hypertext to link other web pages or content to the current displayed page.

The first version of HTML dates back to 1990–1991, when physicist Tim Berners-Lee of CERN proposed the adoption of the standard for research documents. In 1993 Berners-Lee filed a proposal draft for the first HTML specification, backed by the IETF (Internet Engineering Task Force, an organization that coordinates a large number of working and discussion groups on web-related topics); in 1995 the IETF completed the work on HTML 2.0, the first HTML specification intended to be treated as a standard [28]. Since 1996 the HTML specifications have been maintained by the World Wide Web Consortium. The W3C and the Web Hypertext Application Technology Working group started the renewal work for HTML5 in 2004, 4 years later the last updates to the incumbent web standards, HTML 4.01 and XHTML 1.01 [32].

The term HTML5 is usually used to represent two different concepts:

- the new version of the HTML markup language, which includes new elements, attributes, and behaviors;
- a larger set of technologies that allows more diverse and powerful Web sites and applications. This set is sometimes called *HTML5 & friends* and often shortened to just HTML5 [30].

New elements have been introduced to enrich the semantic content of documents (such as the `<article>` or `<section>` elements) and improve the use of rich media (specific `<audio>` and `<video>` tag, as well as support for Math Formulas and vector graphics). Some older elements have been improved and standardized to grant uniform behavior across all browsers, and others have been completely removed and taken over by newer ones. The *Document Object Model* (often shortened as *DOM*), the tree representation of any X/HTML document, assumes much greater relevance in HTML5 and its manipulation is being made more immediate thanks to the introduction of new APIs that can be used with JavaScript. These new APIs extend and improve the already existing interfaces and make for a dramatically more dynamic web experience, with functionalities like 2D and 3D drawing, media playback and in-browser application support [32].

Canvas

Added in HTML5, the HTML `<canvas>` element can be used to draw graphics via scripting in JavaScript. For example, it can be used to draw graphs, make photo compositions, create animations or even do real-time video processing or rendering [45], through the use of the Canvas 2D Context API [75]. Canvas was initially introduced by Apple for use inside their own Mac OS X WebKit component in 2004 and was adopted by other web browsers vendors in the following years, along with the initiation of the standardization process by the Web Hypertext Application Technology Working Group (WHATWG) on new proposed specifications for next generation web technologies [9]. It is currently in *Candidate Recommendation* state [75] .

A Candidate Recommendation is a document that W3C believes has been widely reviewed and satisfies the Working Group's technical requirements [77].

Essentially, the canvas element is a rectangular area on the screen that you can draw into [39]. The 2D Context API gives you the possibility of drawing lines, shapes, images and text, or to apply transformations to drawn elements or add effects such as shadows [75].

Canvas is lower level than other drawing methods such as the SVG Vector API, so you can have more control over the drawing and use less memory, making it a great choice for charts, graphs, dynamic diagrams, and interactive interfaces, such as video games [39]. The major disadvantage is that usually it requires more lines of code, although this can be mitigated by the use of additional libraries built on top of the main API.

Canvas is one of the new features introduced as part of the HTML5 specification that is supported by the vast majority of web browsers: complete or partial support is provided in web browsers used by almost 90% of the global users at the time of this writing [8].

Audio

The `<audio>` element is used to represent sound content in documents without recurring to third party plugins. Added as part of HTML5, it may contain several audio sources, represented using the src attribute or the `<source>` element; the browser will choose the most suitable one. Fallback content for browser not supporting the element can be added too [44]. `<audio>` falls under the *Media Element* interface definition along with another element introduced in HTML5, `<video>`, sharing the same attributes. Their attributes spec-

The Media Element *specification was defined in order to add other compliant elements later on.*

ify the behavior of the element: in particular, it is possible to decide if the reproduction of the audio source should be controlled to the user agent's default controller, or if the author has provided its own using the associated scripting directives; an element could play its source sounds as soon as the page finishes loading thanks to the `autoplay` attribute; there are also attributes that regulates reproduction of the sound such as `loop` and `muted`.

The source of the sound playing in each `<audio>` element should be declared as the value of the `src` attribute, or with one or more `<source>` children element(s). The latter solution is recommended to provide multiple copies of the same file encoded with different codecs in order to implement a fallback chain. Not all browsers, intact, support the playback of all the common codecs used. In fact, the adoption of HTML5 audio and video has become polarized between proponents of free and patented formats [29]. Currently, different web browsers supports different subsets of the proposed codecs pool, which includes the royalty-free WebM and Ogg Vorbis formats and more popular formats such as MP3 and AAC [40]. HTML5 also provides an implementation for displaying synchronized text (such as captions) for Media Elements through the use of children elements.

Video codecs and audio codecs are used to handle video and audio, and different codecs offer different levels of compression and quality. A container format is used to store and transmit the coded video and audio (both together, the case of a video with a soundtrack) [40].

2.2.2 Web Audio API

Web Audio API is high-level JavaScript API for processing and synthesizing audio in web applications. The goal of this API is to include capabilities found in modern game engines and some of the mixing, processing, and filtering tasks that are found in modern desktop audio production applications [15]. The design and development process followed previous attempts by independent developers, which created third-party implementations of similar concepts to overcome the limitations of audio management in modern web browsers. The result is a versatile API that can be used in a variety of audio-related tasks, from games, to interactive applications, to very advanced music synthesis applications and visualizations [63].

One notable example is the Audio Data API that was designed and prototyped in Mozilla Firefox. Mozilla's approach started with an element and extended its JavaScript API with additional features [63].

The APIs have been designed with a wide variety of use cases in mind. Ideally, they should be able to support any use case which could reasonably be implemented with an optimized C++ engine controlled via JavaScript and run in a browser. That said, modern desktop audio software can have very advanced capabilities, some of which would be difficult or impossible to build with this system. Nevertheless, the proposed system will be quite capable of supporting a large range of reasonably complex games and interactive applications, including musical ones. And it can be a very good complement to the more advanced graphics features offered by WebGL. The API has been designed so that more advanced capabilities can be added at a later time [15].

Main Features

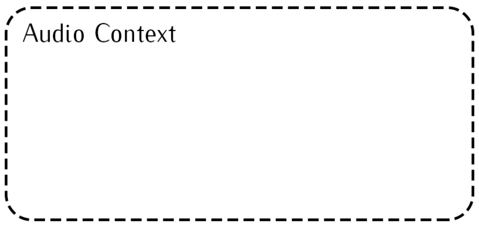
The W3C Audio Working Group is actively working on Web Audio API; the progress have been documented in a series of *working drafts* on the W3C website and is open for community discussion. The W3C Technical Report Development Process will eventually produce a *recommendation* that will phase through different stages of maturity before receiving the endorsement of the W3C Members and the Director [77]. The latest published working draft is the fifth on the specification, and was published on the 10th of October, 2013 - although minor updates have been subsequently issued [15].

The main features introduced up to this point include processing of audio from different sources such as an `audio` or `video` media element, a remote WebRTC stream, or the user's camera and microphone; audio synthesis and processing directly in JavaScript as well as through the use of pre-built structures that allow the user to easily access a library of effects, wave generators and waveshaping tools; modular routing for simple or complex mixing/effect architectures, including multiple sends and submixes; automation of audio parameters for envelopes, fade-ins/fade-outs, granular effects, filter sweeps, LFOs etc.; flexible handling of channels in an audio stream, allowing them to be split and merged; spatialized audio supporting a wide range of 3D games and immersive environments; efficient real-time time-domain and frequency analysis/music visualizer support [15].

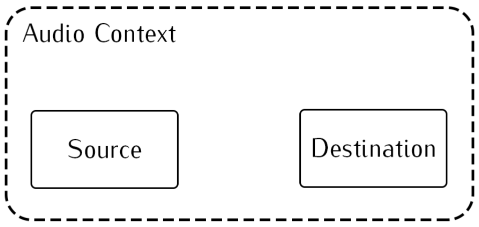
WebRTC is a free, open project that enables web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs [82].

Modular routing

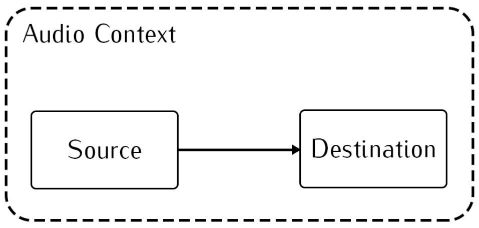
The Web Audio API is built around the concept of an *audio context*. The audio context is the main element of the specification, and it is implemented by the `AudioContext` interface. In practical terms, the audio context is a directed graph of nodes that defines how the audio stream flows from its source to its destination. As audio passes through each node, its properties can be modified or inspected [63]. These nodes are implemented by the `AudioNode` interface, which describes the main characteristics each node must possess. Let us examine how the implementation works: the `AudioContext` interface represents a set of `AudioNode` objects and their connections. It allows for arbitrary routing of signals to the `AudioDestinationNode` (what the user ultimately hears). Nodes are created from the context and are then connected together. In most use cases, only a single `AudioContext` is used per document. Figure 13 depicts the process of creating and connecting the AudioNodes in an AudioContext for the simplest structure available, a direct connection between a *source* node and a *destination* node. Figure 14 depicts a more intricate routing graph with different kinds of nodes connecting multiple sources to manipulation nodes and finally to the destination node.



(a) Creation of the `AudioContext`.



(b) Creation of two `AudioNodes`.



(c) Connection of the two `AudioNodes`.

Figure 13: Modular routing with Web Audio API.

Audio Nodes

Audio nodes are created using the `create` method of the `AudioContext` interface. In the current implementation, there is a specific method for each type of node. Each node can have inputs and/or outputs: for example, a source node has no inputs and a single output, while an `AudioDestinationNode` has one input and no outputs and represents the final destination to the audio hardware. Most processing nodes such as *filters* will have one input and one output. Each type of `AudioNode` differs in the details of how it processes or synthesizes audio but, in general, each `AudioNode` will process its inputs (if it has any), and generate audio for its outputs (if it has any) [15].

Each output has one or more channels. The exact number of channels depends on the details of the specific `AudioNode`. An output may connect to one or more `AudioNode` inputs, thus *fan-out* is supported. An input initially has no connections, but may be connected from one or more `AudioNode` outputs, thus *fan-in* is supported. Each `AudioNode` input has a specific number of channels at any given time. This num-

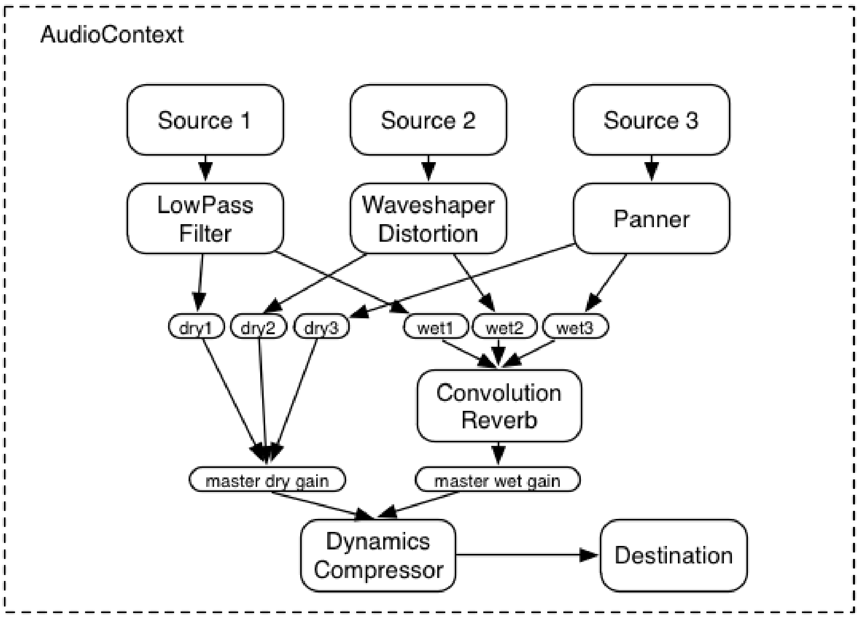


Figure 14: A more complex example of modular routing with Web Audio API. Source: [15].

ber can change depending on the connection(s) made to the input. If the input has no connections then it has one channel which is silent [15].

For performance reasons, practical implementations will need to use block processing, with each `AudioNode` processing a fixed number of sample-frames of size block-size. In order to get uniform behavior across implementations, this value has been defined explicitly: block-size is defined to be 128 sample-frames which corresponds to roughly 3ms at a sample-rate of 44.1KHz [15].

Connection between nodes is performed using the `connect()` method, which accepts the name of the node we want to connect to as a parameter. In the example of figure 13, the connection is performed by executing `Source.connect(Destination);`. Connections can also be removed by calling the `disconnect()` method, which operates in a similar way. There are different kinds of audio nodes that can be routed together to form the desired audio graph. We can arrange them in four main groups based on their role in the routing graph:

1. **Source Nodes** are used to generate sounds in the `AudioContext`. The source of the sound can be an audio asset stored in memory (represented by an `AudioBuffer` object), for which an `AudioBufferSourceNode` is used; the content of an audio or video HTML element, for which a `MediaElementAudioSourceNode` is used; a MediaStream, for which a `MediaStreamAudioSourceNode` is used. It is also possible to generate a periodic waveform sound using the `OscillatorNode`, which allows the user to define the type

of waveform (*sine*, *square*, *sawtooth*, *triangle* or *custom*) and the wave frequency in Hertz;

2. **Modification Nodes** accept one or more inputs and operate linear or non-linear transformations on the signal which is then outputted. Examples are the `GainNode`, which alter the level(s) of its input signal(s), the `BiquadFilterNode`, which allows simple filtering using a second-order filter structure of which it is possible to define the type of filtering (*low-pass*, *high-pass*, band-pass, *notch* and others) and filtering parameters such as the cut/center frequency and the *Quality factor*, and the `DelayNode`, which allows to delay the input signal by a certain amount of seconds. Two particular modification nodes are the `ConvolverNode`, which allows to apply a linear convolution effect to the signal given an impulse response saved as a `.wav` PCM file, and the `ScriptProcessorNode`, which allows to operate directly on the samples of the input signal using JavaScript;

3. **Analysis Nodes** operate as *pass-through* nodes (the audio stream is passed un-processed from input to output) and provide information regarding their input signals. The `AnalyserNode`, for example, can provide real-time frequency and time-domain analysis information using the `getByteFrequencyData()` and `getByteTimeDomainData()` methods;

4. **Destination Nodes** gather the final signal obtained through the routing graph and send it to a specific destination. The `AudioDestinationNode` is an AudioNode representing the final audio destination and is what the user will ultimately hear. It can often be considered as an audio output device which is connected to speakers. All rendered audio to be heard will be routed to this node, a "terminal" node in the AudioContext's routing graph. There is only a single `AudioDestinationNode` per AudioContext, provided through the `destination` attribute of `AudioContext` [15]. The `MediaStreamAudioDestinationNode`, instead, can be used to generate a stream of the input signal and send it for example, to a remote peer using the RTCPeerConnection `addStream()` method.

Browser Support

Unlike other HTML5 features, Web Audio API is still in *Working Draft* state [15], therefore support for all its functionalities is still not guaranteed in web browsers. Currently, about 60% of the Internet user base utilizes a browser which supports Web Audio API [80]. A big part of the remaining ~40% is represented by mobile browsers, which, while being based on the same basic core as their desktop counterpart, often sacrifice advanced functionalities in order to improve performances.

2.2.3 CSS3

Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language. CSS is a cornerstone specification of the web and almost all web pages use CSS style sheets to describe their presentation [18].

CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple pages to share formatting, and reduce complexity and repetition in the structural content [18], as well as making maintenance easier [46].

CSS can also allow the same markup page to be presented in different styles for different rendering methods, such as on-screen, in print, by voice (when read out by a speech-based browser or screen reader) and on Braille-based, tactile devices. It can also be used to allow the web page to display differently depending on the screen size or device on which it is being viewed [18].

CSS specifies a priority scheme to determine which style rules apply if more than one rule matches against a particular element. In this so-called cascade, priorities or weights are calculated and assigned to rules, so that the results are predictable. The CSS specifications are maintained by the World Wide Web Consortium (W3C). Developed in levels, CSS1 is now obsolete, CSS2.1 a recommendation and CSS3, now split into smaller modules, is progressing on the standard track [46].

As HTML grew, it came to encompass a wider variety of stylistic capabilities to meet the demands of web developers. The ideal way to handle styling was to separate the structure from the presentation and give the user different options for at least three different kinds of style sheets: one for printing, one for the presentation on the screen and one for the editor feature [18].

To improve web presentation capabilities, nine different style sheet languages were proposed to the World Wide Web Consortium's (W3C) www-style mailing list. Of the nine proposals, two were chosen as the foundation for what became CSS: Cascading HTML Style Sheets (CHSS) and Stream-based Style Sheet Proposal (SSP). Several key figures in the web development community worked together to develop the CSS standard (the 'H' was removed from the name because these style sheets could also be applied to other markup languages besides HTML) [18]. The CSS level 1 Recommendation was published in

December 1996; in 1997 the W3C formed the CSS Working Group, a specific organization to prosecute the work on the CSS standard [18].

The CSS Working Group began tackling issues that had not been addressed with CSS level 1, resulting in the creation of CSS level 2 on November 4, 1997. It was published as a W3C Recommendation on May 12, 1998. CSS level 3, which was started in 1998, is still under development as of 2014 [18].

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties. A style sheet consists of a list of *rules*, each formed by one or more *selectors* and a *declaration block*. In CSS, selectors are used to declare which part of the markup a style applies to; a selector could then comprise all elements of a specific type (declaring the equivalent HTML tag), or all the elements specified by a particular attribute, such as *class* or a unique identifier (*id*). Selectors may be combined in many ways, especially in CSS 2.1, to achieve great specificity and flexibility [18]. A declaration block consists of a list of declarations in braces. Each declaration itself consists of a property, a colon (:), and a value. If there are multiple declarations in a block, a semi-colon (;) must be inserted to separate each declaration [18].

Rules are applied to page elements according to their *specificity*, a value that describes the relative weights of various rules and determines which styles are applied to an element when more than one rule could apply. Based on specification, a simple selector has a specificity of 1, class selectors have a specificity of 10, and ID selectors a specificity of 100 [18]. A combined selector assumes a specificity value equal to the sum of the specificity of the elementary selectors that are part of it.

The final style for an element can be specified in many different places, which can interact in a complex way. Three main sources of style information form a cascade. They are:

- The browser's default styles for the markup language.
- Styles specified by a user who is reading the document.
- The styles linked to the document by its author in an external file, in a definition at the beginning of the document or on a specific element in the body of the document.

Different styling priorities are assigned to rules originating from different sources. For styles in the cascade, author stylesheets have priority, then reader stylesheets, then the browser's defaults [46]. Other types of styling have intermediate priority collocations [18].

A fundamental aspect in CSS is *inheritance*, the mechanism by which properties are applied not only to a specified element, but also to its descendants. Inheritance relies on the document tree, which is the hierarchy of (X)HTML elements in a page based on nesting. Descendant elements may inherit CSS property values from any ancestor element enclosing them. Inheritance prevents certain properties from being declared over and over again in a style sheet, allowing the software developers to write less CSS. It enhances faster-loading of web pages by users and enables the clients to save money on bandwidth and development costs [18]. For inherited styles, a child node's own style has priority over style inherited from its parent [46].

CSS3 Features

As with HTML5, the W3C is pursuing development of the third *level* of CSS development with the usual *maturation process*. The work on CSS3 has been split into different *modules*, each related to a different feature, each being developed independently. So far only four modules have been published as *formal recommendation*:

- Color (level 3);
- Selectors (level 3);
- Namespaces;
- Media queries;

with other modules currently in *candidate recommendation* status that can be used safely in any modern browsers [14]. Unfortunately, the most interesting modules (such as *animations*, *transforms* and *filters*) are still in early development stages; browsers vendor have made an effort to provide support for most of them, but their utilization should be backed by a thorough analysis of browsers adoption and should provide *fallback* solutions for unsupported browsers [31].

2.2.4 Node.js

Node.js is a platform developed for easily building fast, scalable network applications. It uses an event- driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [48].

Node.js was created by Ryan Dahl in 2009 as a set of asynchronous libraries taking advantage of the *V8* JavaScript Engine (developed and open sourced by Google and powerful component of their *Chrome* web browser), which translate JavaScript code to native, optimized machine code [72], as well as a custom HTTP parser and the *libev*, *libeio*, *evcom* and *udns* libraries for asynchronous I/O and event loop management [62] – later replaced by the *libUV* astraction layer [49]. Before its inception, few other asynchronous libraries and frameworks existed – notable examples include the *Twisted* framework for Pyhton

(2002) and *EventMachine* for Ruby (2003) [34] – and the use of JavaScript as a server-side programming language was well known since the late '90s. However, up until that point nobody even thought about a combination of the two. Dahl believed there was some potential in the development of such thing and focused his energy on the project, finally managing to showcase his progresses at *JSConf 2009* in Berlin [25]. From there on, Node.js gained the support of thousands of independent developers, and at the end of 2010 it also obtained the financial support of the software and services company Joyent [70].

Node.js is currently distributed as an installer package available for download on the official website [48]. Once installed, any Node application can be launched through the use of the "`node`" command; the installer also makes the access to the Node Packaged Modules [47] (third-party-developed modules that expand Node's capabilities) directory possible through the "`npm`" command.

Main features

The main feature of Node.js is its *event loop*. The event loop makes asynchronous coding possible without requiring overhead-heavy solutions like multithreading (the "traditional" way of dealing with concurrent requests). Using callbacks the developer can stop worrying about what happens in the backend, and he/she is guaranteed that his/her code is never interrupted and that doing I/O will not block other requests without having to incur the costs of thread/process per request (i.e. memory overhead) [69]. The event loop handles and processes external events and converts them into callback invocations – I/O calls are the points at which Node.js can switch from one request to another. At an I/O call, the code saves the callback and returns control to the Node.js runtime environment, which proceeds with the execution of the code. The callback will be executed later when the data is actually available, and the results will be handled using powerful constructs such as first-class functions and anonymous functions [69]. Figure 15 shows how the event loop works.

The event loop is a single thread that handles multiple concurrent connections; this makes the overhead of Node.js grow relatively slowly as the number of requests it has to serve increases because there's no OS thread/process initialization overhead. All long-running tasks (network I/O, data access, etc...), instead, are always executed asynchronously on top of worker threads which return the results via callback to the event loop thread. JavaScript's language features (functions as objects, closures, etc...) and Node's programming model make this type of asynchronous/concurrent programming much easier to utilize – there's no thread management, no synchronization mechanisms, and no message-passing nonsense [66]. The single-thread nature of the event loop makes the execution of programs virtually deadlock-free [37].

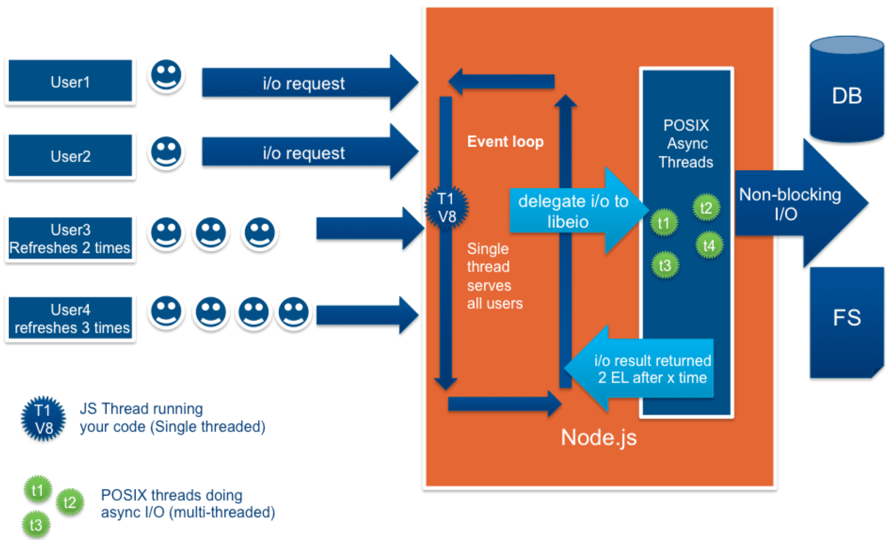


Figure 15: The *event loop* in Node.js. Source [26].

Node.js is a server-side software system designed for writing scalable Internet applications, notably web servers. Programs are written on the server side in JavaScript. Node.js creates a web server by itself, making it unnecessary to use web server software such as Apache or Lighttpd and allowing full control of how the web server actually works [49].

Node.js is extremely developer-friendly. First of all, it enables web developers to create an entire web application on both server-side and client-side in one language, JavaScript, which is easy to learn and largely present on the Internet. Node is simple to install and configure and it just works "out of the box" on a variety of operative systems. It has a built-in package manager called *NPM* which lets users install additional modules with a single command-line instruction. Modules are additional libraries which introduce support for new functionalities such as database connection, encryption and view templating. Modules are the result of the hard work of the ever-growing developers community surrounding Node, which since its launch has collaborated in refining and expanding its capabilities [83], [50].

Pros & cons of using Node.js, and typical use cases

Nowadays Node.js is being used in a lot of applications all over the internet, meeting the needs and requirements of a vast number of web application. It is not, however, the "*Holy Grail*" of webapp development: there are some use cases for which Node just is not cut for and will not bring any advantage – sometimes even producing a performance loss. Applications that are very heavy on CPU usage, and very light on actual I/O, such as video encoding software, artificial intelligence or similar CPU hungry software, can be implemented with Node, but you will probably get better results with C or C++ [27]. If you still need to do CPU-heavy computation and interfacing with the web a good solution lies in realizing your web services with Node, coding the heavy part in C/C++ and then using one of the many C/C++ interfaces for Node to realize the communication between the two ends. Node will not bring any more benefits than PHP, Ruby or Python for *CRUD* ("*Create, Read, Upload, Delete*") or simple HTML applications. You will most likely end up sacrificing the power of tested and reliable frameworks for a little scalability (although there is an increasing number of interesting frameworks for quick and efficient webapp creation being developed for Node lately) [27]. That said, there are some use cases in which Node.js is actually recommended for the advantages it brings to the table: all of them pretty much rely on of the fact that a Node web server can satisfy a great number of requests thanks to asynchronous coding. An example of this would be implementing lightweight REST/JSON APIs: Node's non-blocking I/O model combined with JavaScript actually make it a great choice for wrapping other data sources such as databases or web services and exposing them via a JSON interface [27]. Node is also great for soft real-time applications such as chats/instant messaging networks, *Twitter*-like feed apps and sports betting interfaces, as well as *AJAX/websocket* single page apps (a good example of this is Google's *GMail*). The ability to process many requests per seconds with low response times, combined with features like sharing components such as validation code between the client and server make it a great choice for modern web applications that do lots of processing on the client [27]. Finally, data streaming is another good use case: in fact, traditional web stacks often treat http requests and responses as atomic events, but they actually are streams, and many cool Node.js applications can be built to take advantage of this fact. One great example is parsing file uploads in real time, or building proxies between different data layers. This was actually the use case which inspired the creation of Node – in the beginning, Dahl wanted to find the best way of notifying a user, in real time, about the status of a file upload over the web [25].

3 MAGNETIC TAPE RECORDER VIRTUALIZATION: PROJECT ANALYSIS AND DESIGN PROCESS

The main aim of this work was to develop the virtualization of an electroacustic instruments using some of the Web technologies described in chapter 2, in order to research the validity of an implementation of this kind. Our goal was to realize a valid digital emulation of a physical instrument that could take advantage of the latest development in Web technologies in order to profit from the advantages of web applications. In particular, we decided to dedicate our attention on that subsets of instruments described in section 1.5: recording and replaying systems. For the moment, we focused only their playback functionality, given its importance in the philological access of audio documents.

In this section we will outline the requirements for our application, as well as introducing the device on which we based our work, the *Studer A810* magnetic tape recorder. We will then describe the design process that allowed us to emulate the main features of this particular instrument.

3.1 REQUIREMENT ANALYSIS

First of all, let us define the requirements and goals of this work. As we saw in section 1.4.2, active preservation of an instrument must accomplish two major requirements:

- **Preserving the identity of the instrument by accurately replicating all of its sound features**. In the case of playback systems, this means that all the possible modifications to the audio signals introduced by the device must be emulated as close as possible;
- Providing the users an **interface that resembles the original one**, either by using physical components or by designing a virtual interface.

While it may not seem obvious, even playback systems introduce a certain level of signal alteration, often to improve the quality of the output which is usually affected by alterations produced during the recording and playback processes.

In addition, we wanted to develop an application that can be accessed remotely, allowing the user to have at least part of the experience he would have by using the physical device, but from his current location.

In a typical situation, a user interested in accessing a particular audio document located in a multimedia archive would physically reach the building, compile an access request, receive the original support (usually for a short amount of time) and then be granted access to the equipment needed to replay the document (if he/she does not own it him/herself). Our implementation of the virtual device instead could make remotely accessing both the document and the playback equipment possible, simplifying the fruition process for both the users and the archive. Another notable advantage introduced is the fact that the user would receive digital access copies specifically prepared for this process (for example, compressed and watermarked, in order to prevent unauthorized diffusion), safeguarding both the original, physical support and the main preservation copy.

3.2 THE STUDER A810 MAGNETIC TAPE RECORDER

The electromechanical instrument on which we based our work is the *Studer A810* Magnetic Tape Recorder. One specimen of this device is currently in use at the Centro di Sonologia Computazionale of the Università degli Studi di Padova, and it still being used to perform the re-recording process of audio documents preserved on magnetic tapes. The A810 is a professional machine introduced in the early 1980s by Swiss equipment manufacturer Studer. This instrument is suited for both recording and playback of audio from a variety of magnetic tapes, supporting most of the recording standards.

The A810, as all the other professional tape recorders, has been designed to optimize the quality of the recorded signal during a recording session, and the quality of the playback signal when a tape is played. In practical terms, this means that sensitive, noise-free amplification and special correction and equalization are applied to the output signal, in order to compensate distortion and alteration introduced by the magnetic flux of the tape [7].

The most effective correction applied by the tape recorder while playing a tape is the equalization of the signal, which is introduced according to a specific frequency response associated with an analog circuit. This equalization is introduced with an analog circuit obtained from the serie of two RC Networks: figure 17a and 17b show the two structures of theses networks.

Each network is characterized by the value of their resistance and capacitance components, R and C: these can be chosen to implement a different equalization curve. In particular, each frequency response is associated to the product of these two values, which is called *equivalent time constant*: we call $t_1 = R_1C_1$ the product of the resistance and capacitance value of the network of type (a), and $t_2 = R_2C_2$ the same product, but for the network type (b). As the name suggests,

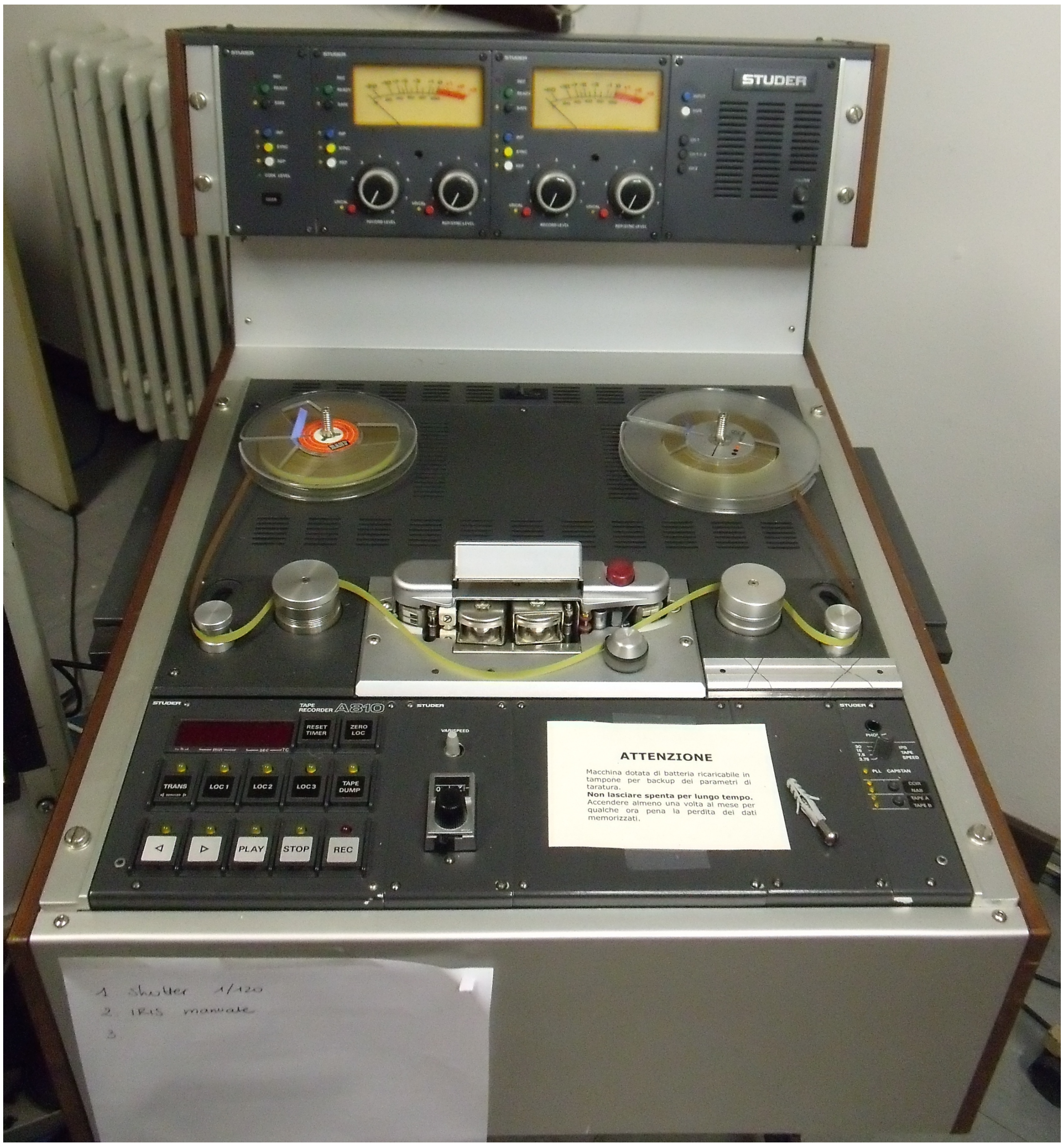
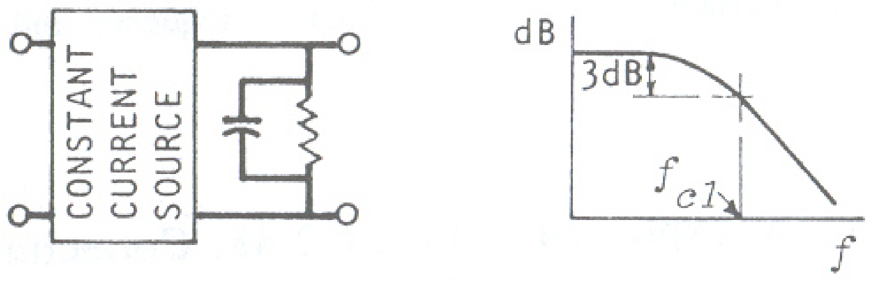
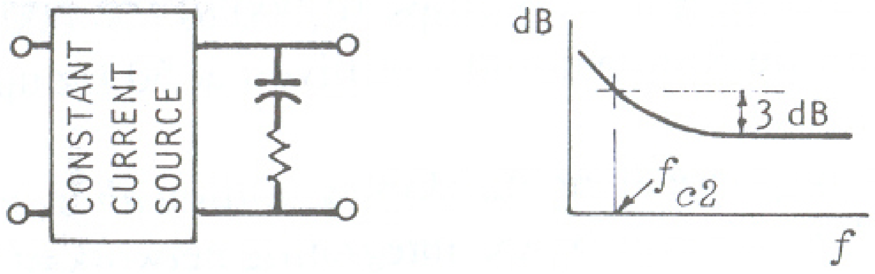

STUDER
STUDER
TAPE RECORDER A810
RESET TIMER
ZERO LOC
TRANS
LOC 1
LOC 2
LOC 3
TAPE DUMP
PLAY
STOP
REC
VARISPEED
ATTENZIONE
Macchina dotata di batteria ricaricabile in tampone per backup dei parametri di taratura.
Non lasciare spenta per lungo tempo.
Accendere almeno una volta al mese per qualche ora pena la perdita dei dati memorizzati.
1 Shutter 1/120
2 IRIS manuale
3

Figure 16: The Studer A810. Photo taken at the Centro di Sonologia Computazionale, Padova.



(a) Parallel RC network.



(b) Series RC network.

Figure 17: RC networks used in tape equalization. Source: [7]

t_1 and t_2 assume a *time* value and are usually expressed in microseconds (µs). It is then possible to associate each equalization curve to a specific time constant, with different values producing slightly different frequency responses. Figure 18 shows some of these curves, where the left-hand side (*low frequencies emphasis*) equalization is introduced by the network of figure 17b, while the right-hand side (*high frequencies filtering*) is due to the network of figure 17a.

The effect of varying the values of t_1 and t_2 are also shown: in particular, it is possible to notice that increasing the value of t_1 results in a slightly more steep right end of the curve, while increasing t_2 "flatten" the left end.

These frequency responses can be plotted for different values of t_1 and t_2 by using the following equation:

$$N_{dB} = 10\log_{10}\left[1 + \frac{10^{12}}{(2\pi f t_2)^2}\right] - 10\log_{10}\left[1 + 10^{-12}(2\pi f t_1)^2\right] + K$$

where K is a constant that can be chosen to shift the zero dB reference to 1000 Hz (typical for professional standard) or 315 Hz (typical for domestic standard), without changing the shape of the curve. From the time constant we can also obtain the cutoff frequency of each filter, with the relation

$$f_{c_i} = \frac{10^6}{2\pi t_i}$$

To ensure that tapes made on a different model of recorder could be properly reproduced on another system, equalization standards

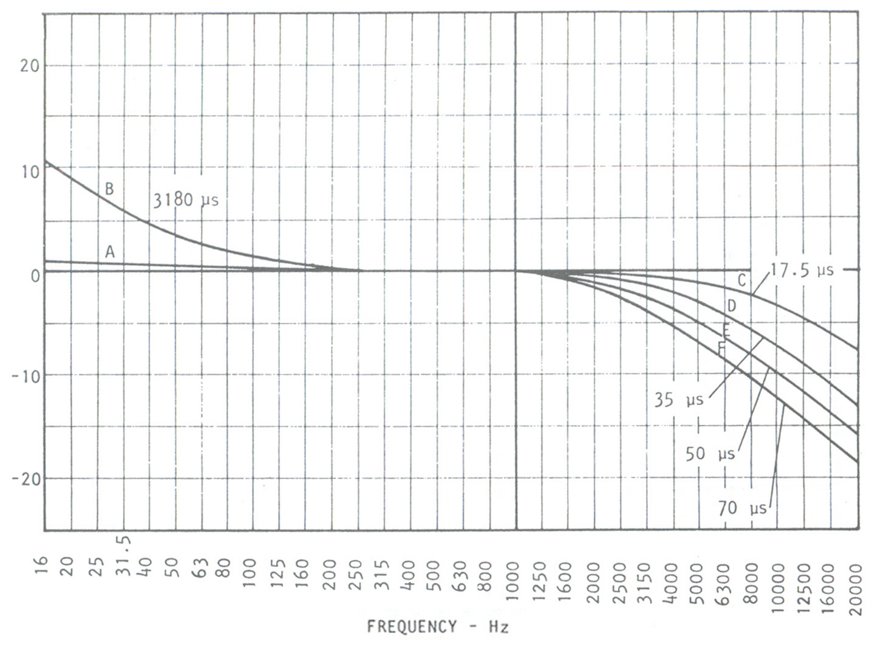


Figure 18: Frequency response curves for professional equalization standards. Source: [7]

were introduced by relevant organizations in the industry, such as the International Electrotechnical Commission and the National Association of Broadcasters. Given the variety of tape models in common use, a family of equalization standards have been designed, with one appropriate for each chosen playback speed, head gap, and application, making recordings freely interchangeable (within their own classification) on different machines [7].

Frequency response curves for professional tape standards are visible in figure 18, while those for commercial and home use are shown in figure 19.

In our work, we emulated four professional standards introduced by the IEC, and three domestic standards proposed by the NAB. Table 1 shows these standards along with their tape speed in mm/s and in/s and the values of the two time constants t_1 and t_2 that shapes the equalization curve.

3.2.1 Replication of the equalization in the digital domain

In order to accomplish our first requirement, we started off by outlining a operational strategy to transpose in the digital domain the equalization effects applied by the magnetic tape recorder to the output signal. Given the amount of information available, we managed to emulate the frequency response curves (with good approximation) with digital filters designed in Matlab.

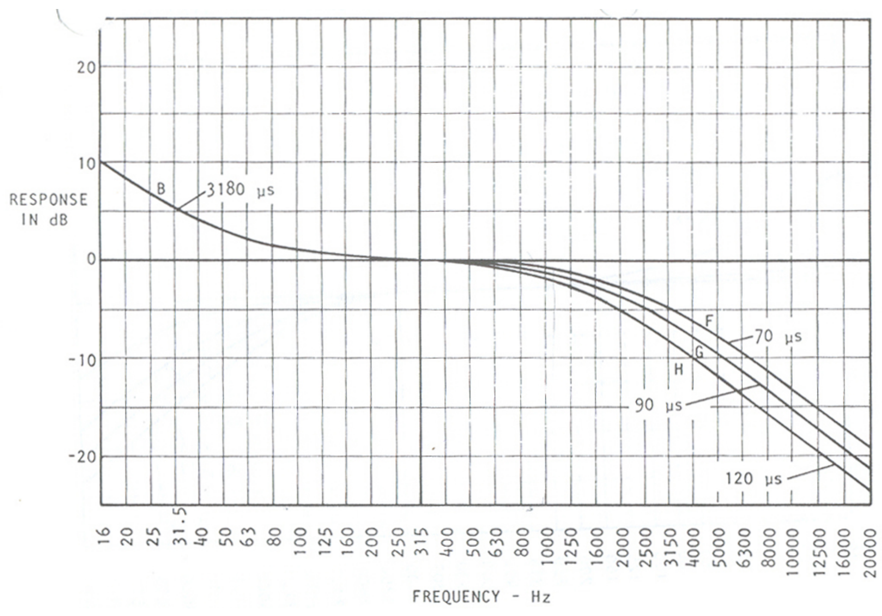


Figure 19: Frequency response curves for commercial and home use tape standards.

The analog-to-digital process was fairly simple: first, we started off by eliciting the transfer function of the series of the two networks of figures 17a and 17b which is of the following form:

$$H(s) = \frac{1 + st_2}{st_2(1 + st_1)}$$

The bilinear transform maps the analog complex variable s to the digital complex variable z with the following substitution: $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$, *where T is the numerical integration step size of the trapezoidal rule used in the bilinear transform derivation.*

Then, we applied the *bilinear transform* method to obtain the transform function of the equivalent digital filter; this was done by using the `bilinear` function in Matlab, which receives as inputs the vectors representing the numerator and denominator factors of the analog transfer function and the sampling frequency used in the digital filter (which we set at 44100Hz as we will work with compressed files

Table 1: Recommended Response Time-Constants. Source: [7].

Application	$t_1(\mu s)$	$t_2(\mu s)$
Professional - Reel-to-reel		
380 mm/s (15 in/s) IEC 1 (1968)	35	∞
380 mm/s (15 in/s) IEC 2	50	3180
190 mm/s (7.5 in/s) IEC 1 (1968)	70	∞
190 mm/s (7.5 in/s) IEC 2	50	3180
Commercial and Domestic (Home-Use) Reel-to-reel		
380 mm/s (15 in/s) (NAB 1965)	50	3180
190 mm/s (7.5 in/s) (NAB 1965)	50	3180
95 mm/s (3.755 in/s) (NAB 1965)	90	3180

which use this frequency value) and outputs two vectors containing the coefficients for the digital filter's transfer function's numerator and denominator.

Figures 20, 21, 22 and 23 show the frequency response curves of the digital filters obtained for different values of t_1 and t_2, which are comparable with good approximation to the analog frequency response curves of figures 18 and 19.

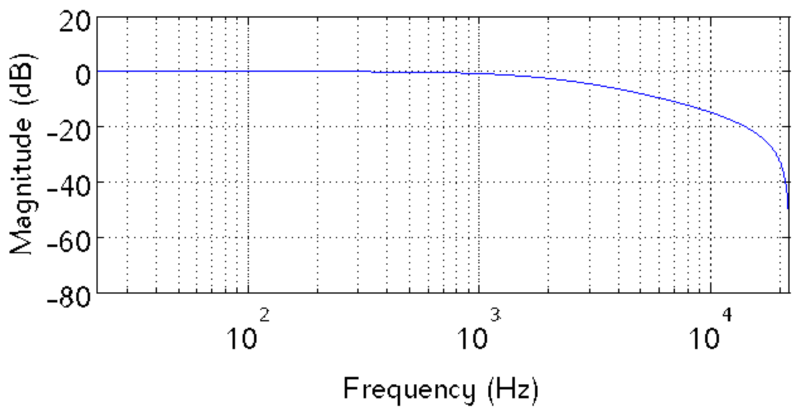


Figure 20: Frequency response curve obtained for the digital implementation of standard IEC 1 at 7.5in/s.

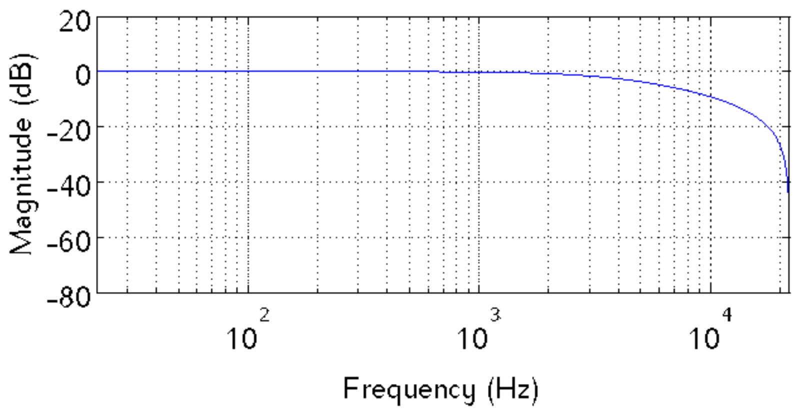


Figure 21: Frequency response curve obtained for the digital implementation of standard IEC 1 at 15in/s.

Finally, we stored the impulse response for each pair of values of t_1 and t_2 as `.wav` PCM files for use in our implementation with the Web Audio API `ConvolverNode`, as we will see in section 4.3.

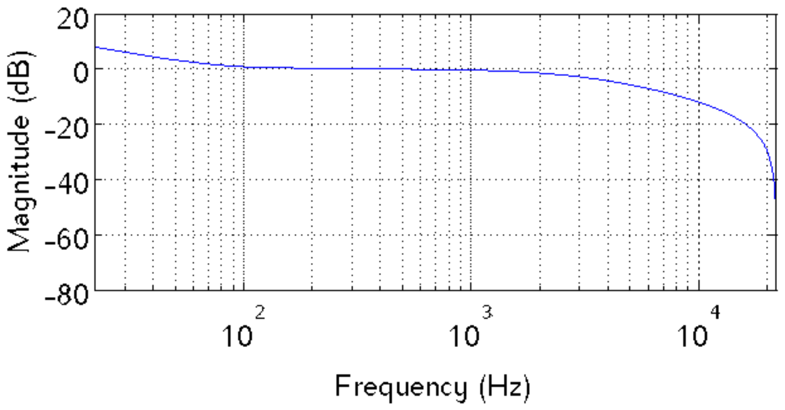


Figure 22: Frequency response curve obtained for the digital implementation of standard IEC 2 at 7.5in/s and 15in/s and standard NAB at 7.5in/s and 15in/s.

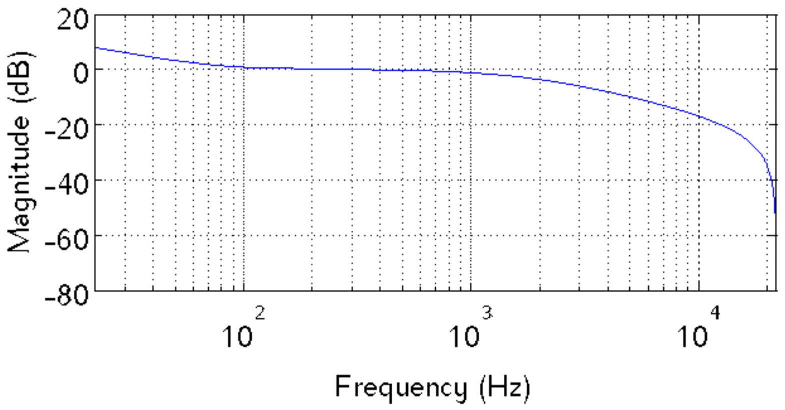


Figure 23: Frequency response curve obtained for the digital implementation of standard NAB at 3.75in/s.

3.2.2 Interface design

Given our goal of realizing the virtualization of the tape recorder using only web technologies, the fulfilling of the second requirement implicated the realization of a *virtual* interface. The design process started with a visit at the Centro di Sonologia Computazionale, where it was possible to observe and operate the A810. The visit was well documented with photos (such as the one shown in figure 16) and videos of the tape recorder in function. One of the researcher of the CSC was kind enough to guide us through the various configuration and control parameters, as well as showcasing some of the material obtained in a typical re-recording session, such as close-up videos of the magnetic tape scrolling through the tape heads.

We then proceeded to design the interface trying to adhere as much as possible to the original design while factoring in sizing and proportion concerns that are typical in Web projects. Using the graphics editing program *Adobe Photoshop* we first set up a grid to outline the main elements of our interface: this is shown in figure 24.

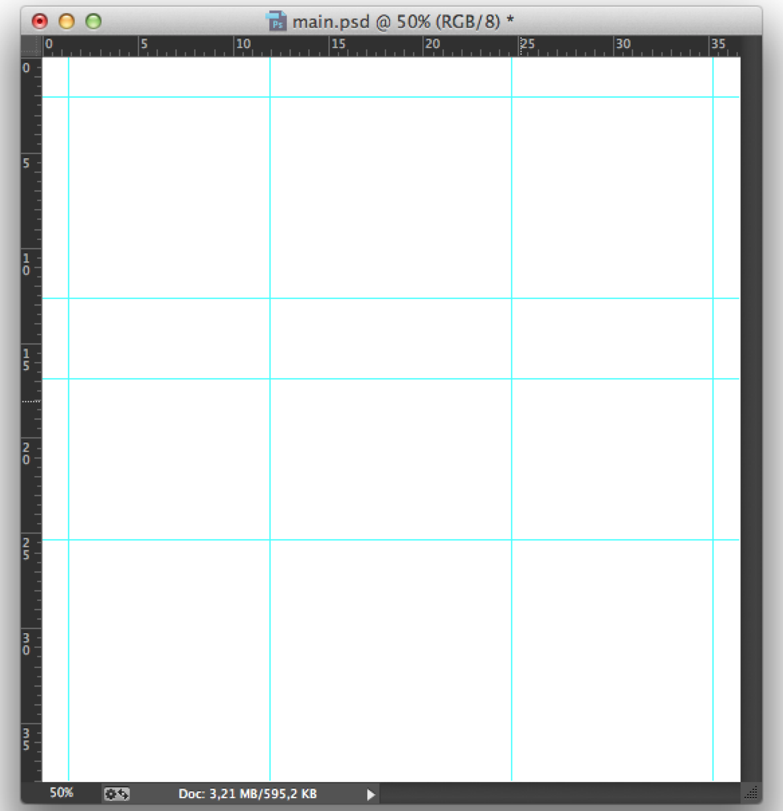


Figure 24: Starting grid for our interface design.

High-definition screens have a resolution of `1920x1080` *pixels.*

We sized our interface at `960x1040` pixels, in order to fill exactly (with an added `40` pixels bar to be placed on top or bottom) half of an HD screen. This design choice may turn out useful in the case, for example, of dedicated emulation machines in libraries and archives, where the app could be launched full-screen, paired with another emulated device of the same size or using the remaining screen estate to show additional informations. Setting the main width to `960` pixels is also a common choice in web design lately, as it is usually a multiple of the screen width of mobile devices, allowing for the introduction of a *responsive* layout later on .

Responsive Web Design is a Web design approach aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices.

After setting up the grid, we started the design by drawing the shapes of the main element of the interface, then refining the details such as gradients, shadows and lighting. For the control mapping, we decided to not replicate all the knobs and buttons available on the machine, but to insert only the controls that map the functionalities that we decided to implement. This resulted in a simpler interface, allowing us to make the controls bigger and easier to detect too, which may help inexperienced users. Figure 25 shows an intermediate step of the design, while the final version will be shown in section 4.3.

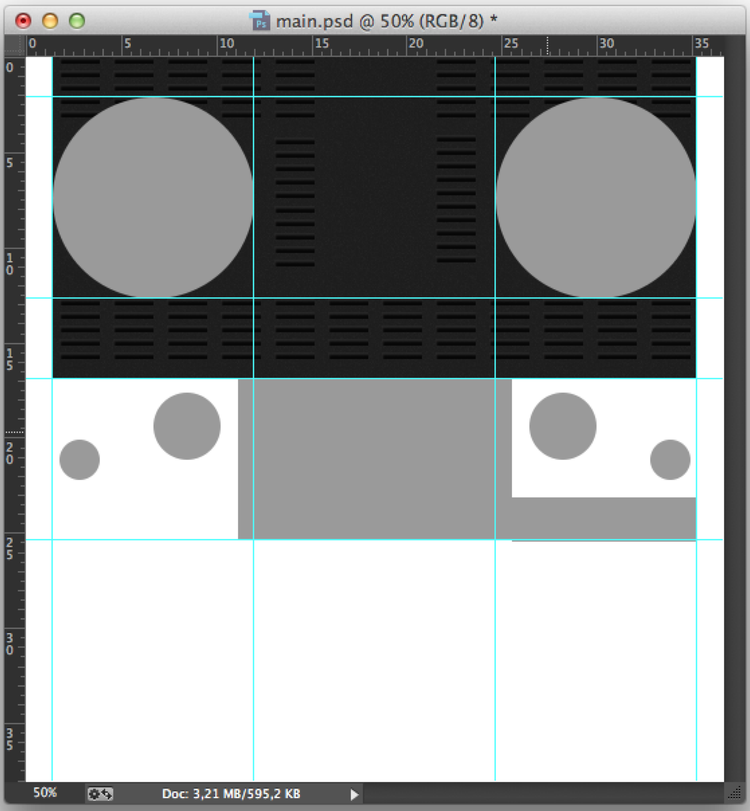


Figure 25: An intermediate step in our interface design process.

4 WEB APPLICATION DEVELOPMENT AND TESTING

After having analyzed and replicated the sound features and the interface of the instrument, we proceeded by implementing the web application using the technologies discussed in chapter 2: HTML5 for the front-end and Node.js for the back-end.

In this section we will first introduce other software tools and frameworks that helped us in the development process; then, we will describe the actual implementation of the two tiers of our application, explaining in details its behavior and all the functionalities achieved. Finally, we will discuss the results of a brief testing process that followed the deployment of the application on a remote server.

4.1 SOFTWARE TOOLS USED

4.1.1 Express.js

We chose Node.js as our backend platform, paired with the powerful *Express.js* framework. Express.js, distributed through Node Package Manager, is a light-weight web application framework which provides a robust set of features for building single and multi-page, and hybrid web applications [22]. Basically, Express.js operates as an upper layer abstracting the functionalities of the *Connect Module*, a third party middleware layer that is itself operating above the *HTTP Module* built in into Node.js. The three main features that Express.js makes available are:

Express.js is the most popular module in the `NPM` registry, and it is used by organizations and companies such as Mozilla *and* MySpace*.*

- **Simple routing** for mapping different requests to specific handles. In particular, each request URL could be mapped to an handle performing a *HTTP Verb* (`GET`, `POST`, `PUT`, `DELETE`, ...) action. It also possible to bypass part of the current route's callback and proceed with another route, making the evaluation of pre-conditions on routes possible;

- **Request handling** with augmented request and response objects that allows the execution of methods not available in vanilla Node, such as `redirect` or `sendfile`;

- **Views rendering** through a specified templating engine, allowing dynamic rendering of web pages. Express works with a number of templating engines, requiring only a simple setup process in the app initialization code.

All of these features were useful in the development of our applications and will be exemplified in the application's code itself for further explanation.

4.1.2 Jade

Even if our application was fairly simple and didn't rely heavily on data models, we chose to render views through one of templating engines supported by Express.js, *Jade*. Jade views are saved as `.jade` files written with a specific syntax which translate directly to HTML, while providing templating capabilities such as the use of variables for dynamic rendering and template inheritance.

4.1.3 Yeoman

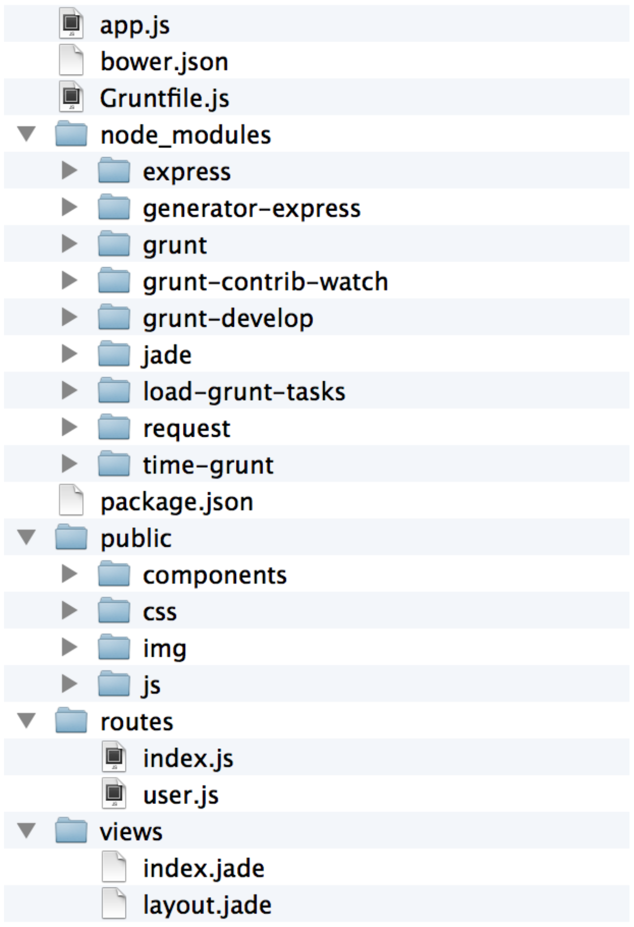
To set up our working directory we used Yeoman, a workflow optimizer tool for web apps [84], which is also available for quick and easy install through `NPM`. The initialization procedure actually installs three pieces of software: *yo*, a scaffolding tool, *grunt*, a build tool, and *Bower*, a package manager. We won't delve into details about these three applications, as their comprehension is not required to understand the development process. What it is necessary to know is that to scaffold out the working directory *yo* uses special plug-ins called *generators*, which can be obtained through various sources, such as code repositories and NPM again. In our case, we used the community-provided *generator-express* [16] plugin, which created the work directory of figure 26.

The scaffolding tool set up the typical root structure of a node app, with the following elements:

- `app.js` is the main application file, which includes the lines of code necessary to launch the application. To start up the web app, it will simply be necessary to launch the `node app` command from the command line;

- `package.json` holds various metadata relevant to the project. This file is used to give information to NPM to allow it to identify the project as well as handle the project's dependencies. It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information and configuration data;

- the `node_modules` directory includes all the necessary node modules to run the application, such as `express.js`. When a new module gets installed with the nom install command, its code gets copied in this directory and made it available to use in the project;



```
  app.js
  bower.json
  Gruntfile.js
▼ node_modules
  ▶ express
  ▶ generator-express
  ▶ grunt
  ▶ grunt-contrib-watch
  ▶ grunt-develop
  ▶ jade
  ▶ load-grunt-tasks
  ▶ request
  ▶ time-grunt
  package.json
▼ public
  ▶ components
  ▶ css
  ▶ img
  ▶ js
▼ routes
    index.js
    user.js
▼ views
    index.jade
    layout.jade
```

Figure 26: The starting work directory for our project.

- the `public` folder contains static assets such as stylesheets, scripts and images, organized in subfolders;
- the `routes` folder contains Express' route files, which hold the code to handle request. They are organized in subfolders with a structure that must pair with the one of the URL we want to use in our project, i.e. handlers associated to the root path ("/") will be stored in the `index.js` file contained in the `routes` folder; handlers associated to the "`/play/`" URL will be stored in the `index.js` file contained in the `play` subfolder, and so on;
- the `views` folder contains the templates for the webpages, in our case some `.jade` files.

From this base structure we started coding our application with a text editor (*Sublime Text 3*), backing up our progress on a *GitHub* repository [2], using the *Git* version control system.

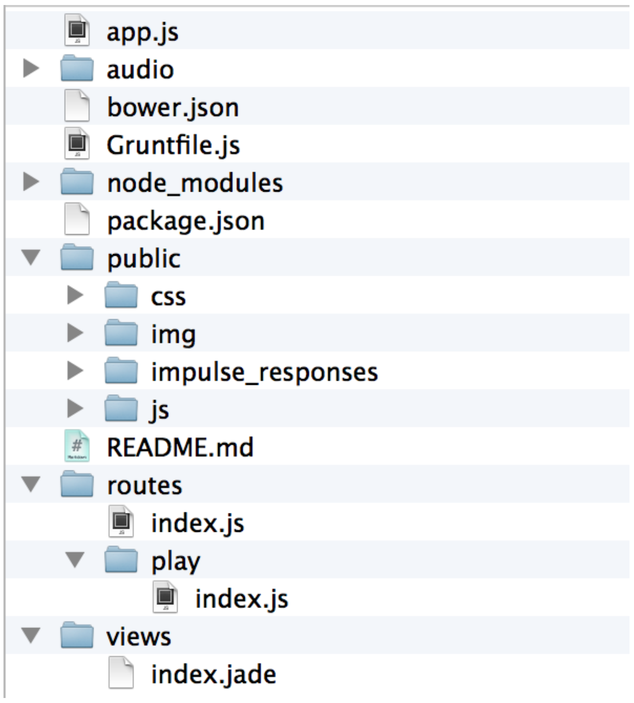
4.2 BACK-END DEVELOPMENT

Following the project set-up phase, we started the development of the back-end of our application. First of all, we made some modifications to the project's structure, by deleting unnecessary files and adding two important folders, `audio`, located in the project's root folder, where placed the access copies (lossy files) of the audio documents we wanted to make accessible through the app, and `impulse_responses`, which contains the `.wav` files of the impulse response generated with Matlab, as seen in section 3.2.1. The final structure of the project at the end of the development process is shown in figure 27.

Now, let us describe the implementation of the web app. We wanted the server-side of our application to respond to three kinds of requests from the client, each associated with a different URL:

- The root URL, to which the server should respond by rendering and sending the main view of the application - which in this case is the only one available, as we decided to implement our web app as a single page application for the moment. This means that a client visiting the URL "`http://serverip:serverport`" on a browser would obtain the `index.html` file generated by the rendering of the `index.jade` file and all the static assets (scripts, stylesheets and images) linked to it to initialize the application and visualize its interface;
- The "`/list`" URL, which would return some sort of list of the playable files to be displayed in a selector on the application interface. We decided to format this list as a `JSON` object, which is a common choice in this type of applications;

JavaScript Open Notation (JSON) is a format used to transmit data objects consisting of attribute-value pairs [36].



```
app.js
▶ audio
bower.json
Gruntfile.js
▶ node_modules
package.json
▼ public
    ▶ css
    ▶ img
    ▶ impulse_responses
    ▶ js
README.md
▼ routes
    index.js
    ▼ play
        index.js
▼ views
    index.jade
```

Figure 27: The final structure of our application.

- The "`play/filename`" URL, to which the server should respond by starting the transfer or a specific audio file (indicated by its `filename`) for reproduction in the application.

To associate each URL with the necessary server logic we used Express.js's routing functionalities. In our main `app.'s` file, following some setup lines, we have the following lines of code:

```
// Request handling
app.get('/', routes.index);
app.get('/list', routes.list);
app.get('/play/:file', play.file);
```

which associate each `GET` request URL to a handler function contained in one of the files in the `routes` folder; for example, the handler for the root URL request implemented in the `index.js` file of the main `route` folder is simply

```
// Returns the Homepage rendered view
exports.index = function(req, res){
  res.render('index', { title: 'Reel2Virtual' });
};
```

where it is possible to see that the handler receives as parameters the `req` (*request*) and `res` (*response*) objects associated with the HTTP `GET` call. The mechanism is the same for the other type of requests (full code listings are available in Appendix A). We made a design choice to simplify the process of extraction of the information related to the audio files (sent through the */list* call), which dictates that the filename of each file contained in the `audio` folder should adhere to the following naming convention:

```
equalizationstandard.filename.filetype
```

where `equalizationstandard` is a string that indicates the equalization standard associated with the tape model of the original support from which the audio file was obtained, `filename` is the title of the audio documents and `filetype` its filetype (i.e. extension). For example, a file named "`15IEC1.Fantasie Impromptu - Frederic Chopin.ogg`" is an `Ogg Vorbis` file containing the recording of "Fantaise Impromptu" by Frederic Chopin, obtained from a tape following the IEC 1 standard with a reproduction speed of $15in/s$.

Finally, we have the `index.jade` template which describes the DOM of the main page of our application.

Jade syntax allows the developers to speed up the writing of webpages' code by taking advantage of features like whitespace-sensitivity *while maintaining a structure similar to that of a regular* `.html` *file.*

4.3 FRONT-END DEVELOPMENT

Once our back-end service was set up and running, we focused our efforts on the development of the client side of the application. Thanks to the capabilities of the features introduced in HTML5, we were able to move most of processing the to the client: as shown in the previous section, in fact, the server does not perform any additional computation other than serving the requests coming from the client.

Application interface

Let us discuss the main application interface first. Figure 28 shows the appearance of the application in a web browser, after all the resources have been loaded. This design is the result of the process outlined in section 3.2.2; it is possible to see that the top of the interface resembles the actual appearance of the device (see image 16), while the bottom part, which contains the controls of the tape recorder, has been vastly simplified. This followed our decision to not show part of the interface that maps controls to features that have not been implemented in our application (for example, some of the control buttons in the lower-left side of the machine), and to re-use the space gained from their removal to increase the size of active controls and overall simplifying the interface.

The interface is composed of two main components:

- the top part (red box in figure 29) is all contained in a big `<canvas>` element, and is actually composed by three layers: a background, a layer containing only the metal component of the two reels and the metal pins that hold them. This because we

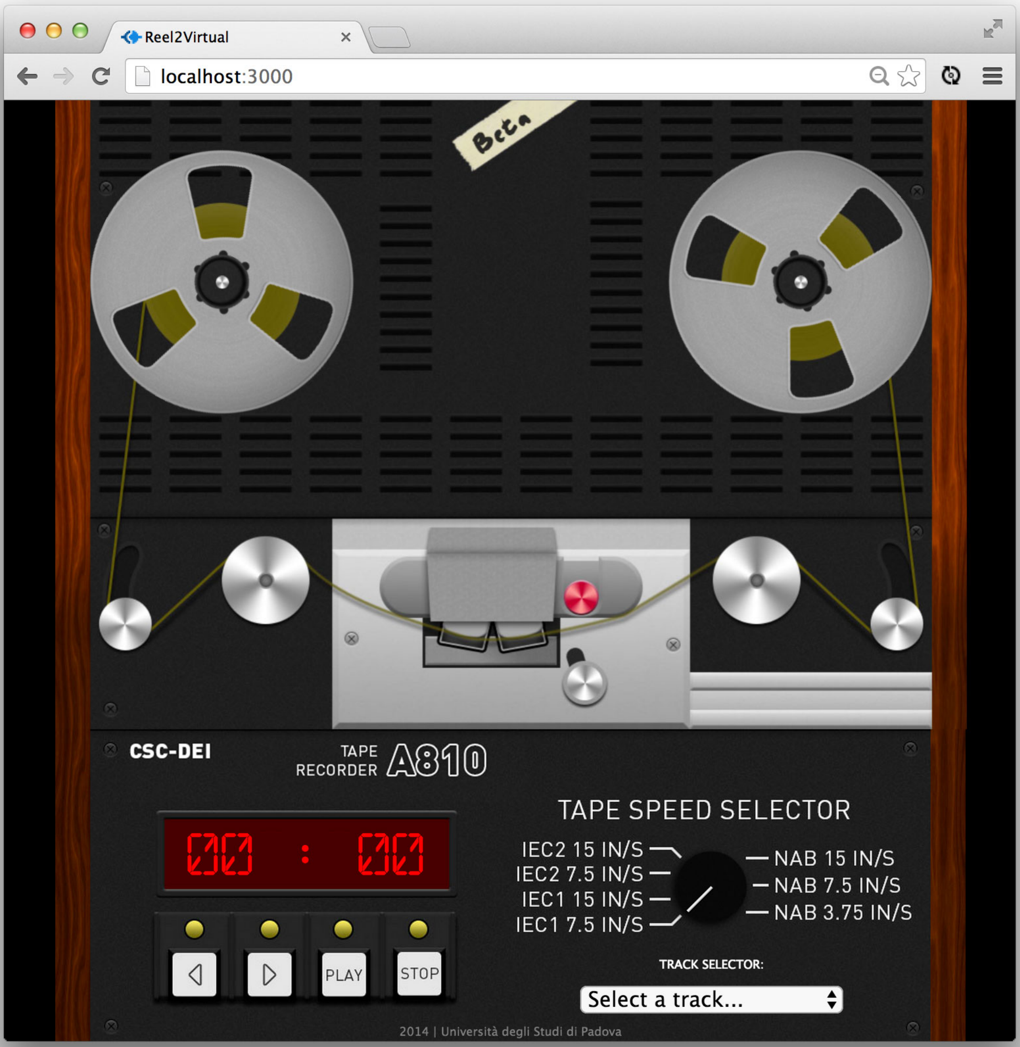


Figure 28: The interface of our web application, as rendered by the web browser.

wanted to reproduce the rotation of the reels during playback. To draw and animate these layers we used *Kinetic.js*, a third-party JavaScript library which greatly simplifies the coding process of `<canvas>` animations. Not only it allowed us to assign *z-indexes* to each layer (in order to stack them correctly), but also to define a function associated to the rotation animation, with the possibility to change the rotation speed during execution. We stored all the code relative to this part of the interface in the `interface.js` script, which is linked to the `index.html` page and therefore automatically executed upon loading;

The z-index sets the stack order of specific elements in various contexts, including CSS.

- the bottom part (blue box in figure 29), instead, was realized using traditional HTML elements such as *divs* (`<div>`) and *paragraphs* (`<p>`). This choice was made in order to link specific elements to the various functions that allow the users to perform actions. These functions, as well as the initialization scripts, are stored in the `eqcontrol.js` file, which is also linked to the main `.html` file. Some functionalities required the use of the popular third party JavaScript library *jQuery*, that we decide to link through the use of a *Content Delivery Network* (CDN), rather than downloading a copy and adding it to our scripts folder, just like we did for the *Kinetic.js* library introduced before.

Initialization

Now, let us describe the functioning of the application. When the root URL is accessed, all the files needed are served by the server. The `interface.js` drawing script runs and render the initial frame of the `<canvas>` element; at the same time, some setup operation are executed in the `eqcontrol.js` script:

The `audio` element does not need to be shown on the page; playback can be managed through a set of dedicated JavaScript APIs.

- a new `audio` element (with a blank `source`) is created; this will host the file currently being played by the application;
- the Web Audio API `AudioContext` is also created to host the routing graph of the application. The graph contains the following elements, which are created in order:
  - a `SourceNode` assuming as input the `<audio>` element generated in the previous step;
  - seven `ConvolverNodes` that are used to implement the seven standards of equalization we analyzed in section 3.2.1. Each `ConvolverNode`, in fact, has a `buffer` property which must hold a representation of the impulse response we want to convolve with the input signal to perform the filtering. Therefore, for each of them, a new request is sent to server, which responds by sending to each of them the correct `.wav` file storing the correspondent impulse response.

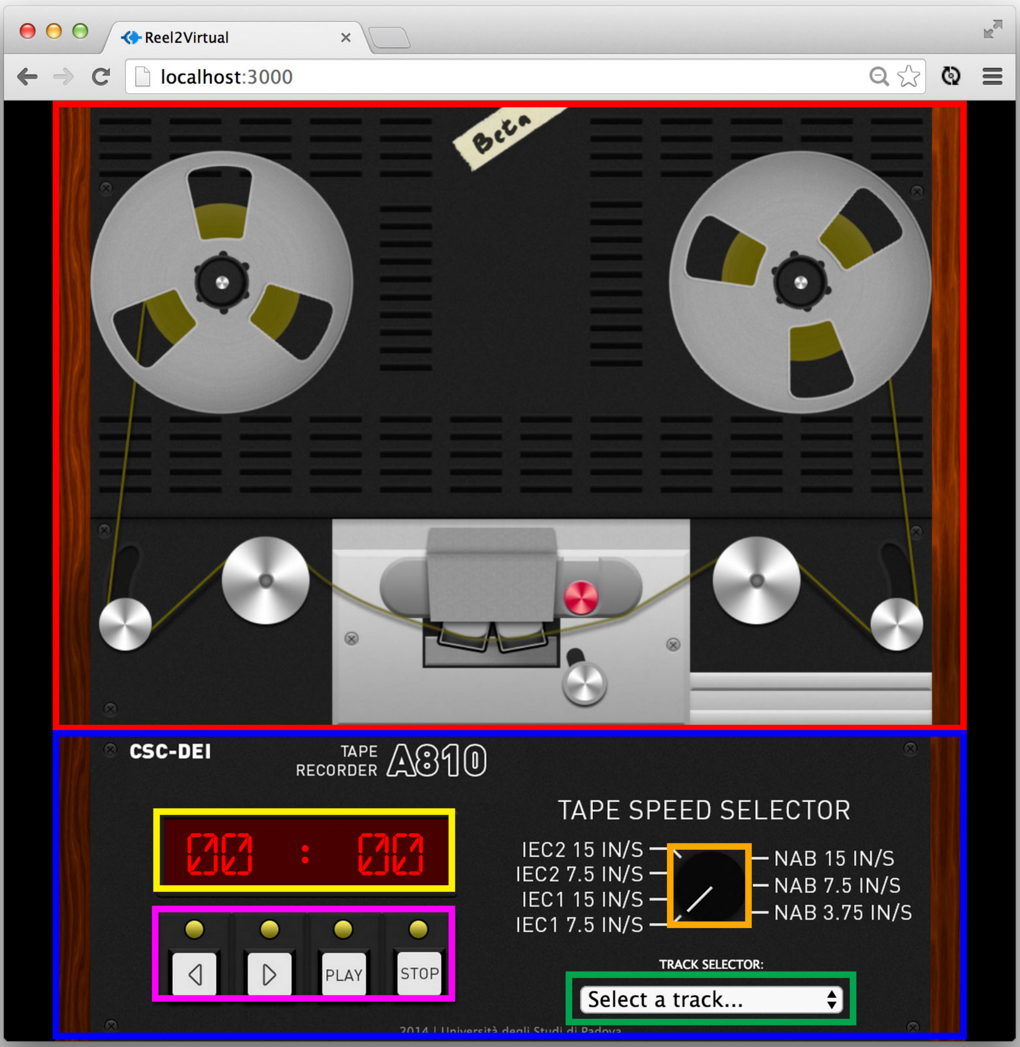


Figure 29: Main interface elements breakdown.

The `ConvolverNodes` are also loaded with additional properties, such as the reel rotation speed in deg/s (estimated from one of the operational videos);

  – an additional `GainNode`, inserted in the `AudioContext` to compensate for the low volume output of the `ConvolverNodes` in Webkit browsers.

The only connection made at this point is the one between the the `GainNode` and the (unique) `DestinationNode` of the `AudioContext` (which will pass the output signal to the system), since the others will be handled automatically by other functions. Figure 30 shows the modular graph obtained at the end of the initialization process;

- a call to the "`/list`" URL is performed to receive the list of playable files stored on the server. When the server responds, the JSON object received is *parsed* and a pull-down menu (green box in figure 29) is automatically populated, creating an option showing the track title and the associated equalization standard for each file.

Application functionalities

Once the page has loaded and the initialization scripts has been executed, the control passes in the hands of the users, which can start using the application. First, a track has to be loaded: this could be done simply by selecting one of the options that populates the drop-down menu (green box of figure 29). As soon as the track has been selected, a few things happen:

- a request is sent to the server at the "`/play/filename`" URL, where `filename` is the filename of the requested track. The server responds by sending back the requested file;
- the audio file requested is set as the *source* of our only `<audio>` element, by changing the value of its `src` attribute;
- the requested file starts buffering;
- the reels animation is triggered at the correct speed;
- the *Tape Speed Selector* knob (orange box in figure 29) is rotated to indicate the selected tape equalization standard used;
- new connections are introduced in the Web Audio API modular routing graph in order to let the sound "flow" from the source to the destination. In particular, the `SourceNode` is connected to the `ConvolverNode` associated to the equalization standard in use (indicated by the Tape Speed Selector knob), and the `ConvolverNode` is subsequently connected to the `GainNode`. At

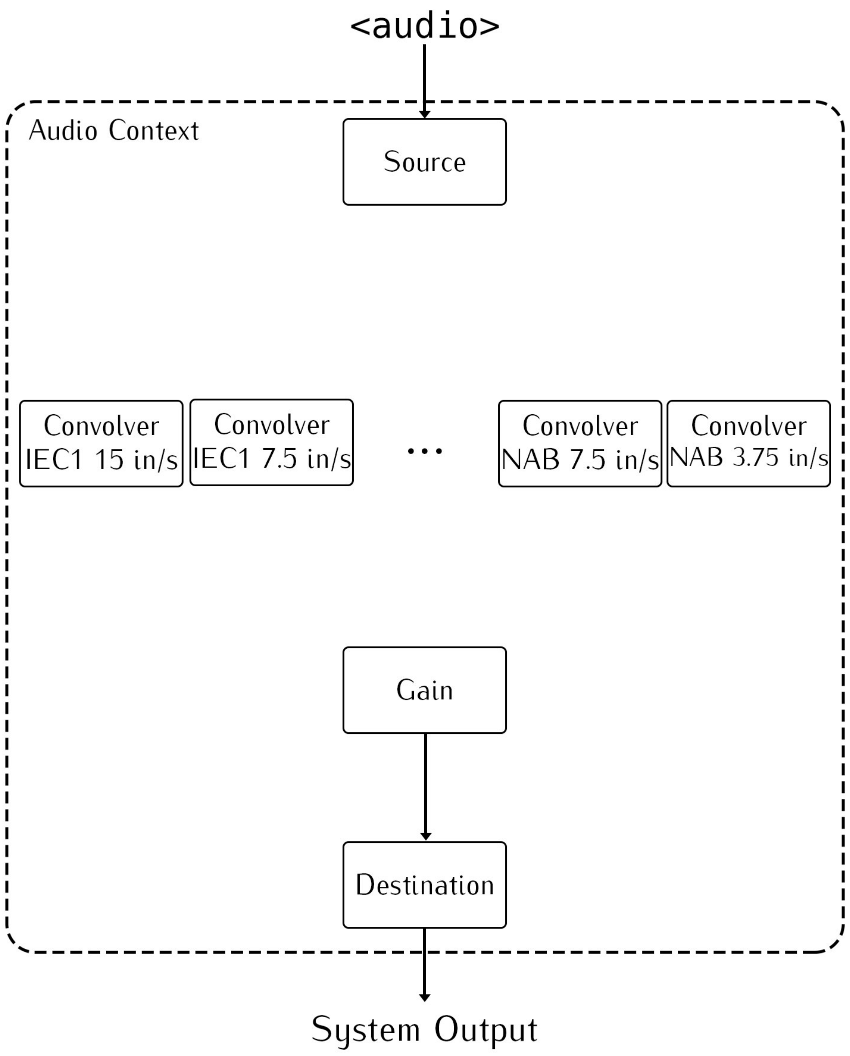


Figure 30: The Web Audio API modular routing graph for our application.

this point, a path from the `Source` to the `Destination` exists, so the sound can be outputted to the system. Figure 31, for example, shows the `AudioContext` after a track with equalization IEC 1 at 7.5in/s has been selected;

This phase is followed by a few seconds of inactivity due to the initial buffering of the audio file. As soon as the first chunk of the file has been receive, the sound start flowing through the `AudioNodes` and gets outputted to the system (speakers). At the same time, the track timer (yellow box of figure 29) starts updating, to reflect the track's elapsed time, replicating the behavior of the original machine.

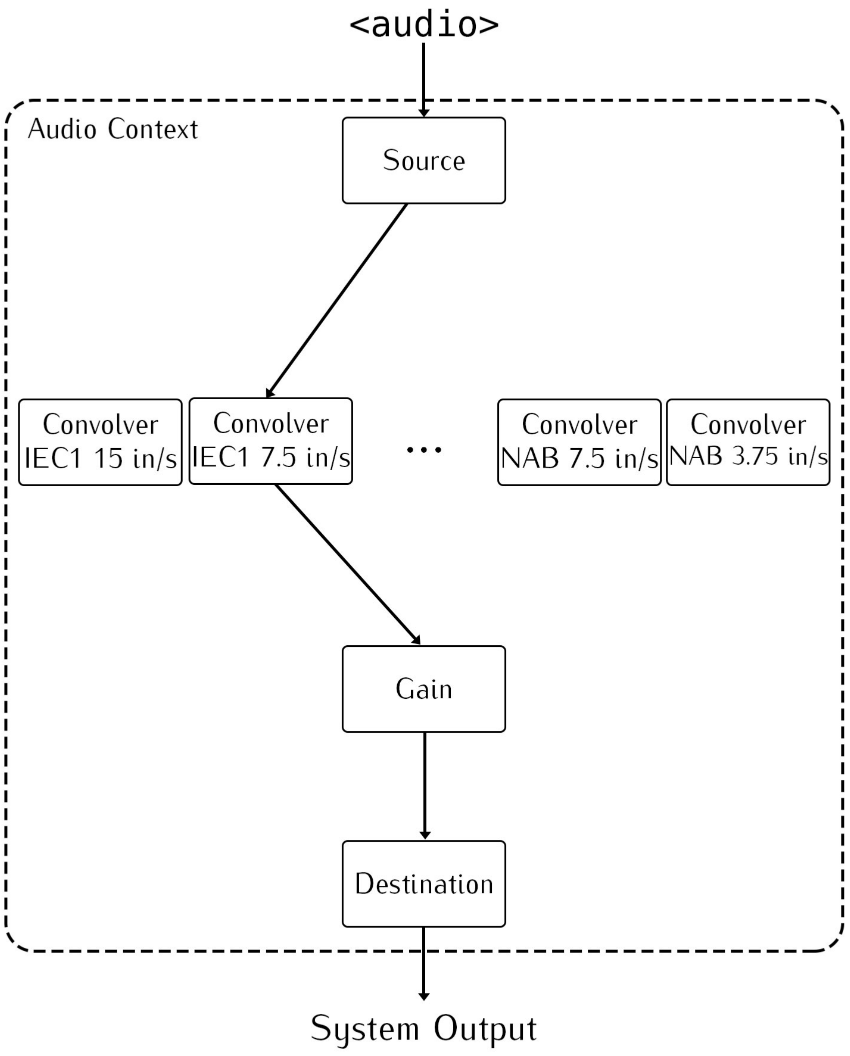


Figure 31: Connections introduced in the `AudioContext` after a track has been selected.

The user can decide to change the track playing simply by selecting another option in the drop-down menu; this implies another execution of the actions of the previous bullet list. Alternatively, the user can decide to manipulate the playback of the current track using the Tape Speed Selector knob or the *control buttons* (pink box of figure 29).

Clicking on the former produces the following effects:

- the knob rotates to indicate that another equalization standard has been selected. Each click on the knob moves it to the next standard in clockwise order;
- the modular routing graph is altered to reflect the choice. In particular, existing connections coming from and going to the `ConvolverNode` associated to the previous standard in use the are disrupted, and new connections from/to the `ConvolverNode` associated to the newly selected standard are introduced, to properly filter the signal;
- the rotation speed of the reels is changed to reflect the new selection;
- whenever possible, the `playbackRate` attribute of the `<audio>` element is changed, in order to provide a more realistic experience. This attribute, in fact, actually changes the reproduction speed of the audio track, exactly like changing the tape speed on the actual device would do.

In current implementations, `playbackRate` *can assume any value greater than* 0*. A value of* 1 *makes the audio play at normal speed, values between* 0 *and* 1 *makes it play slower and values greater than* 1 *makes it play faster. Audible effects are currently limited to a subset of values around* 1 *for performance reasons.*

The control buttons, instead, offer some basic playback control for the the track:

- the "`STOP`" button emulates the action of halting the rotation of the reels, resulting in the both the animation and the audio playback being stopped. The playback is actually being *paused*, and the effect is visible in the track timer, which holds the current elapsed time;
- the "`PLAY`" button allows to proceed with the playback of a track after this has been stopped. As one would expect, both the reel animation and the audio playback are resumed. Pressing this button while the track is already playing doesn't produce any effect;
- the "◁" button allows to *fast-rewind* the current track. The reels starts spinning in the opposite direction (as one would expect) and the playback is reversed and accelerated. The action continues while the button is being pressed; upon release, normal playback is reinstated;
- the "▷" button implements a similar *fast-forward* functionality, allowing to scroll through the track at a faster speed. The behavior is similar to the one of the rewind button, with the track being play at normal rate when the button is released.

4.4 TESTING THE APPLICATION

While our application leverages experimental functionalities that are not completely supported by the majority of the browsers yet, we wanted to be able to evaluate its performances at least in some limited cases, in order to provide a basic reference for future developments. This section outlines some evaluations and measurements as well as a breakdown of the status of support of the various features at the time of writing.

4.4.1 Deployment in the cloud

We saw that the majority of the logic of the application is run on the client, but since the server has to handle the requests and serve all the assets that are needed for the application to run, we wanted to have an idea of the times involved in the loading process. In order to produce realistic results for our tests, we decided to deploy our application on a remote server and access that instead of running it on a local server on the same machine performing the request.

We executed the deployment on *OpenShift* [53], Red Hat's *Platform-as-a-Service*, which provides built-in configurations for running popular back-end framework in the cloud. Without delving into the details, OpenShift's web interface and command-line tools allowed us to create a new instance of Node.js server to run the code fetched from our GitHub repository [**github**]. After having performed some basic configuration and having installated our app, we were ready to begin testing by accessing a dedicated URL.

4.4.2 Tests performed

Testing tools

SERVER SPECIFICATIONS Our server was what OpenShift calls *gear*, a virtual machine with scalable performances operating on top of *EC2*, Amazon's *Infrastructure-as-a-Service* [51]. OpenShift provides a free "small" gear to test out simple application, which can use up to 512 MB of memory and up to 1 GB of storage. An exact measure for CPU performance is not available since computational power is scaled based on the need of the application (within an upper bound).

CLIENT SPECIFICATIONS Front-side testing was made on a Intel Core i5 (2.4 Ghz) machine running Mac OS X 10.9.2. The application was developed and debugged using the *Canary* version of Google's browser Chrome, which is the product of the experimental release channel and therefore introduces new features ahead of the stable release, with all the pros and cons of the case. Additional testing was made on the stable release of Chrome; at the time of writing, this was version 33.0, while Canary was on version 35.0.

We limited our tests to this browser because was the best fit, according to the following reasons:

- it was our browser of choice during development;
- other browsers such as Mozilla Firefox and Opera requires different implementations of core features of our app like the use of Web Audio API, requiring the writing of additional code that could have slowed down our development process, which was just aimed at exploring the possibilities of bleeding-edge features;
- it is capable of reproducing all the most diffused audio formats;
- it sports a powerful set of *Dev Tools* which provides a number of interesting statistics on performance as well as assisting the debug process.

Such limited testing is aimed at giving a qualitative assertion of the feasibility of such a project in the future, when appropriate precautions could be taken in order to provide support to other browsers.

Page load performance

To test the loading and rendering time of our application we used the online tool *WebPageTest* [81], which allows the execution of several kind of tests to evaluate the page load performance of websites and web applications.

We performed multiple runs of the standard test proposed, which analyzes the generation of the document as all the assets are being served to the client, and provides an insightful breakdown of the times involved per each requests.

The results were encouraging: figure 32 shows a document completion time under 1s, with the main interface being displayed in less than 1.5s, upon reception of the static assets (images). The median *Speed Index* result is 1358, meaning that it should be possible to bring it down to around 1000 (a value usually associated with good page performance) with little optimization. This results were achieved by keeping our page structure simple and not adding unnecessary code. We believe that carrying out an optimization process could maintain good performances while allowing for the introduction of advanced functionalities.

The Speed Index is the average time at which visible parts of the page are displayed. It is expressed in milliseconds and dependent on size of the view port [65].

Runtime performance

General runtime performance was analyzed using Chrome's Dev Tools. Again, the simple structure of the application yielded good

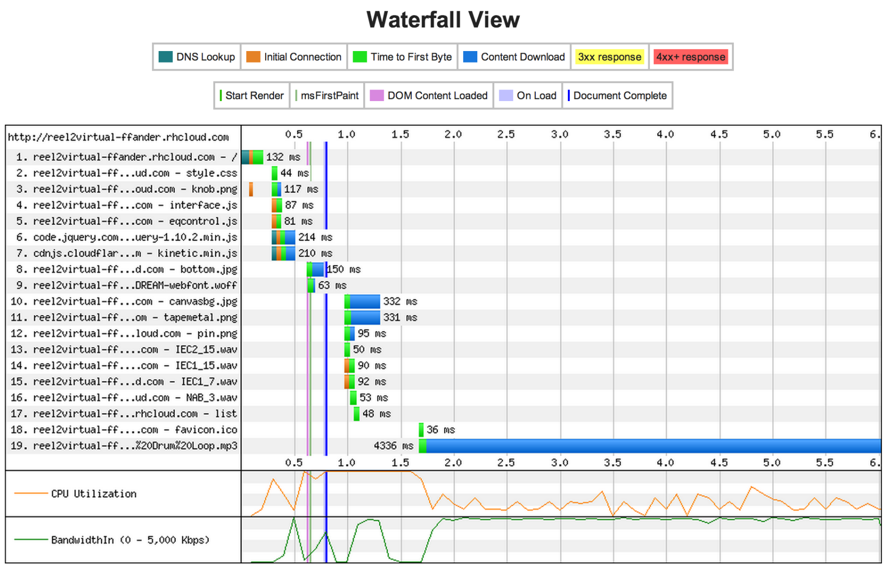


Figure 32: Page load performance analysis results - *waterfall* view.

results, with CPU use never spiking over 15% and layout framerate assessing around the optimal value of 60FPS (*frames per second*) for almost the time of execution. This shows how the use of `<canvas>` to realize the animation was a good development choice, as it has a lower impact on performances.

Mobile testing

As an additional note, we thought it might have been interesting to test our application on a mobile device too, as future developments will surely take into account the increasing relevance of mobile web developing.

Brief tests performed on an a smartphone and a tablet both running the latest version of the Android mobile OS (4.4.2 at the time of this writing) show encouraging results. The latest mobile version of Chrome, while being stripped-down in terms of functionalities in respect to its desktop counterpart, runs the application with no major issues, allowing the reproduction of audio files without significant performance hits. The only unsupported feature, for the moment, was the use of the `playbackRate` attribute for the `<audio>` element, which we deemed acceptable as it is not absolutely necessary for the functioning of the application.

A dedicated development and optimization process for mobile devices, with the adoption of a responsive layout and lightweight scripting could definitely be an interesting road for future work.

5 CONCLUSIONS AND FUTURE WORK

The goal of this work was the design and implementation of a virtual instrument, following the practices and requirements introduced by the archival community, but with additional focus on accessibility and broad diffusion. This particular requirement was accomplished by making use of the recent developments in Web Technologies, which have progressively evolved along with the expansion of the World Wide Web, and can now offer innovative features and tools to develop more and more complex web applications. The type of instrument we virtualized, the magnetic tape recorder, is an important piece of equipment, which assumes a fundamental role in the philological access of audio documents stored on magnetic tapes. Its sound features and its interface, in fact, are essential in order to reproduce a fully philological experience. The realization of our application stands as a first example of how a device like the magnetic tape recorder could be virtualized and made more accessible to users all over the world through the Internet, just like multimedia documents - a paradigm shift that has already been involving archives in recent years. The possibility to serve the users not only the information stored in the access copy of a document, but also (part of) the philological experience associated with it is a great accomplishment, and this work shows how this could be made possible in the near future.

The development process of our web application followed the introduction of innovative, experimental features regarding the managing and manipulation of audio sources in the browser, along with some more established technologies for content serving and rendering. The innovation process, fueled by the constant work of organization such as the W3C and a vibrant community of developers, is definitely going to continue in the next years. This means that incumbent technologies will be constantly improved and new, more efficient technologies will be introduced, but also that the support for them will increase, spreading to more devices and platforms.

The ability to develop such kind of application, leveraging the capabilities of Web Technologies, means that within few years it will become accessible to more and more devices and therefore to more and more users, allowing the spreading of human knowledge, which is the ultimate goal of organizations such as UNESCO.

We believe this work can pose the basis for the realization of more complex applications that could become an important resource in the preservation process of the musical cultural heritage. From this starting point, future developments could spread in different directions. A web application of this kind could be incorporated in the re-recording process by interfacing with the other software tools already available to the archival community. For example, after having obtained the master copy as the result of the (digital) re-recording process, access copies could be generated automatically and made instantly available for access through the web-virtualized version of their playback systems. The application could be improved to display the secondary information related to the audio document that is also generated during the re-recording process, providing a richer experience. This could also be automated by exporting information to and from the access files' metadata, or by interfacing the web app with archive's databases. The application could also be improved to provide different interfaces for common users and archive administrators, with the possibility to manage files and information directly from the browser, becoming a uniform web platform for archive systems.

Since mobile devices already had a great impact on how we interact with information every day, larger attention could be posed on how to improve the mobile experience of these applications, creating dedicated versions more suited to the different screen sizes and capabilities of these devices, and providing fallback solutions to make up for unsupported features, while trying to bring the experience on par with the desktop version as much as possible.

With the evolution of audio-related web technologies and the general improvement of web standards performance it could be possible to develop better emulations of audio devices, making it possible to virtualize instruments using refined methods such as physical modeling. Instruments virtualization could be made easier with the development of libraries and additional tools to facilitate the implementation process of sound features and interfaces.

We believe that in the future web applications will continue to play a great role in the interaction between users and information. Starting to work early on experimental features could yield better results later on, when they will become part of standards specification and become widely adopted. Getting the archival community interested in this scenario could provide great results, and we hope that our simple experiment could become a useful starting point in this regard.

A CODE LISTINGS

BACK-END CODE

app.js – Main application file

```
// Module dependencies.
var express = require('express'),
    routes = require('./routes'),
    play = require('./routes/play'),
    http = require('http'),
    path = require('path'),
    util = require('util');

var app = express();

// Express.js configuration
app.configure(function(){
  app.set('ipaddress', process.env.OPENSHIFT_NODEJS_IP || '127.0.0.1');
  app.set('port', process.env.OPENSHIFT_NODEJS_PORT || 3000);
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  app.use(express.favicon());
  app.use(express.logger('dev'));
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(app.router);
  app.use(express.static(path.join(__dirname, 'public')));
});

// Development environment only for the moment
app.configure('development', function(){
  app.use(express.errorHandler());
});

// Request handling
app.get('/', routes.index);
app.get('/list', routes.list);
app.get('/play/:file', play.file);

// Dispatching public files (css, js, imgs) requests
app.get('/*.(js)', function(req, res){
  res.sendfile("./public/js"+req.url);
});
app.get('/*.(css|eot|svg|ttf|woff)', function(req, res){
  res.sendfile("./public/css"+req.url);
});
app.get('/*.(jpg|png|gif)', function(req, res){
  res.sendfile("./public/img"+req.url);
});
// Impulse responses for the ConvolverNodes are also
// served as static files
app.get('/*.(wav)', function(req,res){
  res.sendfile("./public/impulse_responses/"+req.url);
});

// Web Server Startup
http.createServer(app).listen(app.get('port'),
        app.get('ipaddress'), function(){
  console.log("Express server listening on port " + app.get('port'));
});
```

routes/index.js – Route file

```
var path = require('path'),
        fisy = require('fs');

// Returns the Homepage rendered view
exports.index = function(req, res){
  res.render('index', { title: 'Reel2Virtual' });
};

// Returns a JSON containing a list of playable files
exports.list = function(req, res) {
  var music_path;
  music_path = path.join(path.resolve(__dirname, '..'), 'audio');
  return fisy.readdir(music_path, function(error, files) {
    var file, output, _i, _len;
    output = [];
    for (_i = 0, _len = files.length; _i < _len; _i++) {
      file = files[_i];
      var split = file.split(".");
      if (split[0] !== "") {
        track = {};
        track['ips'] = split[0];
        track['title'] = split[1];
        track['filetype'] = split[2];
        track['filename'] = file;
        output.push(track);
      }
    }
    return res.json(output);
  });
};
```

`routes/play/index.js` – Route file

```
var path = require('path'),
    fisy = require('fs');

// Send the selected audio file
exports.file = function(req, res) {
  var filePath, stat;
  filePath = path.join(path.resolve(__dirname, '..', '..'),
                       'audio', req.param('file'));
  stat = fisy.statSync(filePath);
  res.header('content-type', 'audio/ogg');
  res.header('content-length', stat.size);
  return res.sendfile(filePath);
};
```

`index.jade` – Index page template

```
!!! 5
html(lang='en')
  head
    meta(charset='utf-8')
    title Reel2Virtual
    link(rel='stylesheet', href='style.css')
    script(type='text/javascript',
      src='http://code.jquery.com/jquery-1.10.2.min.js')
    script(type='text/javascript',
      src='http://cdnjs.cloudflare.com/ajax/libs/kineticjs/5.0.1/kinetic.min.js')
  body
    #container
    #bottom
      p#timer
        | 00:00
      .clickcontainer
        a(href='#rewind').click#rewind
        a(href='#ffward').click#ffward
        a(href='#play').click#play
        a(href='#pause').click#pause
      a.knob(onclick='nextEQ(currentEQ);')
```

```
        img#kn(class='IEC1_7', src='knob.png', alt='Speed selector knob')
      .trackSelector
        p
          | TRACK SELECTOR:
        select#selector(onchange="switchSong(value);")
          option(disabled)
            | Select a track...
      p.footer
        | 2014 | Università degli Studi di Padova
    script(type='text/javascript', src='interface.js')
    script(type='text/javascript', src='eqcontrol.js')
```

FRONT-END CODE

`interface.js` – Interface drawing script

```
// Global variable
var anim;

// Reel rotation speed (initial value set to zero)
var angularSpeed = 0;


window.addEventListener('load', function(e) {
  // Kinetic.js Stage object is associated to the
  // <canvas> element
  var stage = new Kinetic.Stage({
    container: 'container',
    width: 1040,
    height: 720
  });

  // Graphics are grouped in a layer
  var layer = new Kinetic.Layer();

  // Loading the tape reels
  var tape = new Image();
  tape.onload = function() {
    var reelL = new Kinetic.Image({
      x: 190,
      y: 210,
      image: tape,
      width: 300,
      height: 300,
      offset: {x:150, y:150},
    rotation: Math.floor(Math.random()*120)
  });
  var reelR = new Kinetic.Image({
    x: 850,
    y: 210,
    image: tape,
    width: 300,
    height: 300,
    offset: {x:150, y:150},
    rotation: Math.floor(Math.random()*120)
  });
  layer.add(reelL);
  layer.add(reelR);
  reelL.setZIndex(1);
  reelR.setZIndex(2);
  layer.draw();

  // Reel animation function
  anim = new Kinetic.Animation(function(frame) {
    var angleDiff = frame.timeDiff * angularSpeed / 1000;
    reelR.rotate(angleDiff);
    reelL.rotate(angleDiff);
  }, layer);
};

// Loading the background
var bg = new Image();
bg.onload = function() {
  var studer = new Kinetic.Image({
    x: 0,
    y: 0,
    image: bg,
    width: 1040,
    height: 720
  });
// Add the shape to the layer
layer.add(studer);
// Move it to the bottom of the layer
studer.setZIndex(0);
layer.draw();
};

// Finally, we load the pins (which shouldn’t rotate
// with the rest of the reels)
var pin = new Image();
pin.onload = function() {
    var pinL = new Kinetic.Image({
      x: 183,
      y: 203,
      image: pin,
      width: 15,
      height: 15
    });
    var pinR = new Kinetic.Image({
      x: 843,
      y: 203,
      image: pin,
      width: 15,
      height: 15
    });
  layer.add(pinL);
  layer.add(pinR);
  // Move them to the top of the layer
  pinL.setZIndex(3);
  pinR.setZIndex(4);
  layer.draw();
  // Add the layer to the stage
  stage.add(layer);
  };

  // Source files
  bg.src = 'canvasbg.jpg';
  tape.src = 'tapemetal.png';
  pin.src = 'pin.png';

}, false);
```

`eqcontrol.js` – Audio management and interaction script

```
// Our <audio> element
var audio = new Audio();
// Hide default controls
audio.controls = false;
audio.autoplay = false;
audio.id = 'track';

// Function for the playback timer update
audio.ontimeupdate = function() {
    var currentSeconds =
        (Math.floor(this.currentTime % 60) < 10 ? '0' : '') +
         Math.floor(this.currentTime % 60);
    var currentMinutes =
        (Math.floor(this.currentTime / 60) < 10 ? '0' : '') +
         Math.floor(this.currentTime / 60);
    // As a side note, native getElementById is faster than
    // jQuery's $ selector
    document.getElementById('timer').innerHTML = currentMinutes +
        " : " + currentSeconds;
};

// Creation of the AudioContext
var context;
if (typeof AudioContext !== 'undefined') {
  context = new AudioContext();
} else if (typeof webkitAudioContext !== 'undefined') {
  context = new webkitAudioContext();
}

// Wait for window.onload to fire
window.addEventListener('load', function(e) {
    // Our <audio> element will be the audio source.
    source = context.createMediaElementSource(audio);

    // Gain to compensate for volume loss after convolution
    gain = context.createGain();
    // It seems to be a problem only in webkit browsers
    if (typeof webkitAudioContext !== 'undefined') {
        gain.gain.value = 25;
    }

    // A convolver for each supported standard
    IEC1_15 = context.createConvolver();
    IEC1_7 = context.createConvolver();
    IEC2_15 = context.createConvolver();
    IEC2_7 = context.createConvolver();
    NAB_15 = context.createConvolver();
    NAB_7 = context.createConvolver();
    NAB_3 = context.createConvolver();

    // Setting the ID for each convolver
    IEC1_15.id = "IEC1_15";
    IEC1_7.id = "IEC1_7";
    IEC2_15.id = "IEC2_15";
    IEC2_7.id = "IEC2_7";
    NAB_15.id = "NAB_15";
    NAB_7.id = "NAB_7";
    NAB_3.id = "NAB_3";
```

```
// Setting the impulse response for each convolver
// (some are shared)
IEC1_15.imp = "IEC1_15";
IEC1_7.imp = "IEC1_7";
IEC2_15.imp = "IEC2_15";
IEC2_7.imp = "IEC2_15";
NAB_15.imp = "IEC2_15";
NAB_7.imp = "IEC2_15";
NAB_3.imp = "NAB_3";

// Chaining the convolver in a round-robin fashion
IEC1_7.next = IEC1_15;
IEC1_15.next = IEC2_7;
IEC2_7.next = IEC2_15;
IEC2_15.next = NAB_15;
NAB_15.next = NAB_7;
NAB_7.next = NAB_3;
NAB_3.next = IEC1_7;

IEC1_7.prev = NAB_3;
IEC1_15.prev = IEC1_7;
IEC2_7.prev = IEC1_15;
IEC2_15.prev = IEC1_7;
NAB_15.prev = IEC2_15;
NAB_7.prev = NAB_15;
NAB_3.prev = NAB_7;

// Setting the (extimated) reel rotation speeds
IEC1_15.speed = IEC2_15.speed = NAB_15.speed = - 360 * 1.5;
IEC1_7.speed = IEC2_7.speed = NAB_7.speed = - 360 * 0.75;
NAB_3.speed = - 360 * 0.375;

// Function to request impulse responses .wav files
function setImpResp(convolver) {
    var request = new XMLHttpRequest();
    request.open("GET", "/" + convolver.imp + ".wav", true);
    request.responseType = "arraybuffer";
    request.onload = function() {
        convolver.buffer = context.createBuffer(request.response,
            false);
        console.log("Impulse response for convolver " +
            convolver.id + " loaded successfully.");
    };
    request.send();
}
```

```
    // Loading impulse responses - TBCompleted
    setImpResp(IEC2_15);
    setImpResp(IEC2_7);
    setImpResp(IEC1_15);
    setImpResp(IEC1_7);
    setImpResp(NAB_15);
    setImpResp(NAB_7);
    setImpResp(NAB_3);

    gain.connect(context.destination);

    // List playable files
    $.getJSON('/list', function(result) {
        $.each(result, function(key, track) {
            $('#selector').append('<option value="'+track.filename+
                '">['+track.ips+'] '+track.title+'</option>');
        });
        audio.src = '/play/'+result[0].filename;
    });

}, false);

// Play - Stop - Prev - Next functions for the buttons
function play(element) {
    element.play();
    anim.start();
}

function pause(element) {
    element.pause();
    anim.stop();
}

// 'next' and 'prev' buttons have been replaced by 'rewind'
// and 'fast forward'.
// We're keeping the code for future reference
function next() {
    var s = document.getElementById('selector');
    s.selectedIndex =
        (document.getElementById('selector').selectedIndex + 1) %
         s.length;
    if (s.selectedIndex === 0)
        s.selectedIndex =
        (document.getElementById('selector').selectedIndex + 1) %
         s.length;
    switchSong(s.value);
```

```
}

function prev() {
    var s = document.getElementById('selector');
    s.selectedIndex = ((s.selectedIndex - 1) % s.length + s.length) %
    s.length;
    if (s.selectedIndex === 0)
        s.selectedIndex = ((s.selectedIndex - 1) % s.length + s.length) %
        s.length;
    switchSong(s.value);
}

// Fast Forward and Rewind functionalities
var intervalRewind;

function fastForward() {
    audio.playbackRate = 4.0;
    changeSpeed(currentEQ.speed * 3);
}

function rewind() {
    changeSpeed(currentSpeed * (-3));
    intervalRewind = setInterval(function() {
       audio.playbackRate = 1.0;
       if (audio.currentTime === 0) {
           clearInterval(intervalRewind);
           pause(audio);
           anim.stop();
       } else {
           audio.currentTime += -0.1;
       }
    },30);
}

function resume() {
    audio.playbackRate = 1.0;
    changeSpeed(currentEQ.speed);
    clearInterval(intervalRewind);
    changeEQ(currentEQ);
}

// Adding event listeners

var wasPlaying = true;

$('#play').click( function() {
    play(audio);
    wasPlaying = true;
});

$('#pause').click( function() {
    pause(audio);
    wasPlaying = false;
});

$('#rewind').mousedown( function() {
    if (audio.paused) {
        play(audio);
    }
    rewind();
});

$('#rewind').mouseup( function() {
    resume();
    if (!wasPlaying)
        pause(audio);
});

$('#ffward').mousedown( function() {
    if (audio.paused)
        play(audio);
    fastForward();
});

$('#ffward').mouseup( function() {
    resume();
    if (!wasPlaying)
        pause(audio);
});

var currentEQ;
var currentSpeed;
var originalSpeed;
var newConv;

// Functions to switch EQ and load new tracks

function changeEQ(newEQ) {
    if (newEQ != currentEQ) {
        source.disconnect(currentEQ);
        source.connect(newEQ);
        newEQ.connect(gain);
        changeSpeed(newEQ.speed);
        currentSpeed = newEQ.speed;
        currentEQ = newEQ;
    }
}

function changeSpeed(newSpeed) {
    angularSpeed = newSpeed;
}

// Change the EQ and the playback rate with the IPS knob
function nextEQ(inEQ) {
    if (originalSpeed / inEQ.next.speed == 2) {
        audio.playbackRate = 0.5;
    }
    else if (originalSpeed / inEQ.next.speed == 0.5) {
        audio.playbackRate = 2.0;
    }
    else {
        audio.playbackRate = 1.0;
    }
    $('#kn').attr('class', inEQ.next.id);
    changeEQ(inEQ.next);
}

// Revert to the original EQ
function revertEQ() {
    nextEQ(newConv.prev);
}

function switchSong(newSong) {
    var newEQ = newSong.split('.');
    anim.stop();
    audio.src = '/play/'+newSong;
    audio.addEventListener('canplaythrough', function() {
    if (newEQ[0] == 'IEC1_15')
        newConv = IEC1_15;
    else if (newEQ[0] == 'IEC1_7')
        newConv = IEC1_7;
    else if (newEQ[0] == 'IEC2_15')
        newConv = IEC2_15;
    else if (newEQ[0] == 'IEC2_7')
        newConv = IEC2_7;
    else if (newEQ[0] == 'NAB_15')
        newConv = NAB_15;
    else if (newEQ[0] == 'NAB_7')
```

```
        newConv = NAB_7;
    else if (newEQ[0] == 'NAB_3')
        newConv = NAB_3;
    changeEQ(newConv);
    originalSpeed = newConv.speed;
    $('#kn').attr('class', newConv.id);
    audio.playbackRate.value = 1.0;
        play(audio);
    });
    wasPlaying = true;
}
```

MATLAB SCRIPT FOR OBTAINING THE DIGITAL FILTER FOR EACH EQUALIZATION STANDARD

```
clc;
clear all;
close all;

% t1 and t2 values
% Equalization IEC2 at 15 in/s
t1 = 50*10^-6;
t2 = 3180*10^-6;

% Sampling frequency for the digital filter
Fs = 44100;

% Analog transform function coefficients
num = [t2 1];
den = [t1*t2 t2 0];

% Applying bilinear transformation
[numd,dend] = bilinear(num,den,Fs);

% Plot the frequency response
figure
freqz(numd,dend,1000,Fs);

% Set logarithmic scale on the frequency axis
ax = findall(gcf, 'Type', 'axes');
set(ax, 'XScale', 'log');
set(ax, 'XLim', [0 22050]);
set(ax, 'YLim', [-80 20]);

% Plot the impulse response
```

```
figure
impz(numd,dend);

% Save impulse response
[filtimp,T] = impz(numd,dend,[],Fs);
stereofilt = horzcat(filtimp,filtimp);
audiowrite('IEC2_15.wav',stereofilt,Fs);
```

BIBLIOGRAPHY

[1] *AJAX (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Ajax_(programming) (cit. on p. 22).

[2] Francesco Anderloni. *reel2virtual code repository*. URL: https://github.com/ffander/reel2virtual (cit. on p. 50).

[3] *Audio Units (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Audio_Units (cit. on p. 16).

[4] Federico Avanzini and Sergio Canazza. *Virtual analogue instruments: an approach to active preservation of the Studio di Fonologia Musicale*. Vol. The Studio di Fonologia - A Musical Journey 1954-1983 Update 2008-2012. Hal Leonard MGB, 2009 (cit. on pp. 9, 11, 12, 15).

[5] George Boston. "Safeguarding the Documentary Heritage. A Guide to Standards, Recommended Practices and Reference Literature Related to the Preservation of Documents of All Kinds". In: *UNESCO* (1988) (cit. on pp. 6, 7).

[6] Federica Bressan and Sergio Canazza. "A Systematic Approach to the Preservation of Audio Documents: Methodology and Software Tools". In: *Journal of Electrical and Computer Engineering* 2013 (2013), p. 21 (cit. on pp. 4–8).

[7] Marvin Camras. *Magnetic Recording Handbook*. Van Nostrand Reinhold Company, New York, 1988 (cit. on pp. 38, 40–42).

[8] *Canvas browser support*. URL: http://caniuse.com/canvas (cit. on p. 25).

[9] *Canvas Element (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Canvas_element (cit. on p. 25).

[10] Joel Chadabe. "Le principe du voltage-control et ses implications pour le compositeur". In: *Musique en Jeu* 8 (1972) (cit. on p. 10).

[11] *Chrome Developer Tools*. URL: https://developers.google.com/chrome-developer-tools/ (cit. on p. 23).

[12] *Cloud Computing (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Cloud_computing (cit. on p. 20).

[13] *Connections Counter: The Internet of Everything in Motion*. URL: http://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342 (cit. on p. 20).

[14] World Wide Web Consortium. *Current state of work on the CSS3 Standard*. URL: http://www.w3.org/Style/CSS/current-work (cit. on p. 33).

[15] World Wide Web Consortium. *Web Audio API - W3C Working Draft 10 October 2013*. URL: http://www.w3.org/TR/webaudio/ (cit. on pp. 26–30).

[16] Pete Cooper. *Generator express*. URL: https://github.com/petecoop/generator-express (cit. on p. 48).

[17] Roland Corporation. *AIRA — TR-909 Rhythm Composer*. Jan. 2014. URL: https://www.youtube.com/watch?v=pXsMvTSCkuY (cit. on p. 13).

[18] *CSS (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Cascading_Style_Sheets (cit. on pp. 31–33).

[19] *Cultural Heritage (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Cultural_heritage (cit. on p. 1).

[20] *DREAM Project homepage*. URL: http://dream.dei.unipd.it/ (cit. on p. 9).

[21] L. Duranti. "Interpares3 - team Canada final report"". In: *Tech.Rep., University of British Columbia* (2012) (cit. on p. 8).

[22] *Express.js - node.js web application framework*. URL: http://expressjs.com (cit. on p. 47).

[23] *Facebook Poll: 94 Percent Of Users Don't Like Redesign*. URL: http://techcrunch.com/2009/03/19/facebook-polls-users-on-redesign-94-hate-it/ (cit. on p. 22).

[24] M. Factor, E. Henis, and D. Naor et al. "Authenticity and provenance in long term digital preservation: modeling and implementation in preservation aware storage". In: *Proceedings of the 1st Workshop on the Theory and Practice of Provenance, San Francisco, California, USA* (2009) (cit. on p. 8).

[25] Jarod Ferguson. *Solving the upload progress bar problem – The History of Node.js*. Feb. 2012. URL: http://elegantcode.com/2012/02/06/solving-the-upload-progress-bar-problemthe-history-of-node-js/ (cit. on pp. 34, 36).

[26] Cloud Fondry. *Future-proofing Your Apps: Cloud Foundry and Node.js*. URL: http://blog.cloudfoundry.com/2012/06/27/future-proofing-your-apps-cloud-foundry-and-node-js/ (cit. on p. 35).

[27] Felix Geisendörfer. *Felix's Node.js Convincing the boss guide*. 2011. URL: http://nodeguide.com/convincing_the_boss.html (cit. on p. 36).

[28] *HTML Wikipedia Entry*. URL: https://en.wikipedia.org/wiki/HTML (cit. on p. 24).

[29] *HTML5 Audio (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/HTML5_Audio (cit. on p. 26).

[30] *HTML5 Introduction - Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5 (cit. on p. 24).

[31] *HTML5 Please*. URL: http://html5please.com/ (cit. on p. 33).

[32] *HTML5 Wikipedia Entry*. URL: https://en.wikipedia.org/wiki/HTML5 (cit. on pp. 24, 25).

[33] *Introducing UNESCO*. URL: http://en.unesco.org/about-us/introducing-unesco (cit. on p. 1).

[34] Anders Janmyr. *A Not Very Short Introduction To Node.js*. May 2011. URL: http://www.jayway.com/2011/05/15/a-not-very-short-introduction-to-node-js/ (cit. on p. 34).

[35] *jQuery Homepage*. URL: http://jquery.com/ (cit. on p. 22).

[36] *JSON (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/JSON (cit. on p. 50).

[37] Jeff Kunkle. *Node.js Explained*. July 2012. URL: http://kunkle.org/nodejs-explained-pres/#/dead-lock-free (cit. on p. 34).

[38] *Magnetic tape (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Magnetic_tape (cit. on p. 3).

[39] Josh Marinacci. *HTML Canvas Deep Dive*. URL: http://projects.joshy.org/books/canvasdeepdive/toc.html (cit. on p. 25).

[40] *Media formats supported by the HTML audio and video elements*. URL: https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats (cit. on p. 26).

[41] *Microphone (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Microphone (cit. on p. 10).

[42] *Minimoog (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Minimoog (cit. on p. 16).

[43] *MiniV Plugin product page*. URL: http://www.arturia.com/evolution/en/products/minimoogv/intro.html (cit. on p. 17).

[44] Mozilla Developer Network. *<audio>*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio (cit. on p. 25).

[45] Mozilla Developer Network. *Canvas*. URL: https://developer.mozilla.org/en-US/docs/HTML/Canvas (cit. on p. 25).

[46] Mozilla Developer Network. *CSS*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (cit. on pp. 31–33).

[47] *Node Packaged Modules*. URL: https://npmjs.org/ (cit. on p. 34).

[48] *node.js Official Website*. URL: https://en.wikipedia.org/wiki/V8_(JavaScript_engine) (cit. on pp. 33, 34).

[49] *Node.js Wikipedia Entry*. URL: http://en.wikipedia.org/wiki/Nodejs (cit. on pp. 33, 35).

[50] Jolie O'Dell. *Why Everyone Is Talking About Node*. Mar. 2011. URL: http://mashable.com/2011/03/09/node-js/ (cit. on p. 35).

[51] OpenShift. *CPU performance of small gear*. URL: https://www.openshift.com/forums/openshift/cpu-performance-of-small-gear (cit. on p. 60).

[52] OpenShift. *FAQs*. URL: https://www.openshift.com/faq#t6n11272.

[53] OpenShift. *Homepage*. URL: https://www.openshift.com/ (cit. on p. 60).

[54] Nicola Orio et al. "Methodologies and tools for audio digital archives". In: *International Journal on Digital Libraries* 10.4 (Dec. 2009), pp. 201–220 (cit. on p. 6).

[55] *Phonograph cylinder (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Phonograph_cylinder (cit. on p. 2).

[56] *Phonograph (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Phonograph (cit. on pp. 1, 11).

[57] Raja Rao. *Future-proofing Your Apps*. June 2012. URL: http://blog.cloudfoundry.com/2012/06/27/future-proofing-your-apps-cloud-foundry-and-node-js/.

[58] *Roland TR-909 (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Roland_TR-909 (cit. on pp. 13, 14).

[59] Dietrich Schuller. *Curriculum vitae*. URL: http://www.phonogrammarchiv.at/wwwnew/DS_CV_e.htm (cit. on p. 6).

[60] Dietrich Schüller. "The ethics of preservation, restoration, and reissues of historical sound recordings". In: *Journal of Audio Engineering Society* 39.12 (1991), pp. 1014–1016 (cit. on p. 7).

[61] Stefania Serafin and Steven Gelineck. *Novel interfaces for controlling musical expression: historical overview and current directions*. Vol. The Studio di Fonologia - A Musical Journey 1954-1983 Update 2008-2012. Hal Leonard MGB, 2009 (cit. on p. 15).

[62] *Slides from Ryan Dahl's presentation at JSConf '09*. 2009. URL: http://s3.amazonaws.com/four.livejournal/20091117/jsconf.pdf (cit. on p. 33).

[63] Boris Smus. *Web Audio API: Advanced Sound for Games and Interactive Apps*. O'Reilly Media, Mar. 2013 (cit. on pp. 26, 27).

[64] *Sound recording and reproduction (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Sound_recording_and_reproduction (cit. on p. 1).

[65] *Speed Index - WebPageTest Documentation*. URL: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index (cit. on p. 61).

[66] Aaron Stannard. *Intro to Node.JS for .NET Developers*. Dec. 2011. URL: http://www.aaronstannard.com/post/2011/12/14/Intro-to-NodeJS-for-NET-Developers.aspx (cit. on p. 34).

[67] William Storm. "The establishment of international re-recording standards". In: *Phonographic Bulletin* 27 (1980), pp. 5–12 (cit. on p. 6).

[68] *Synthesizer (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Synthesizer (cit. on p. 12).

[69] Mikito Takada. *Understanding the Node.js Event Loop*. Feb. 2011. URL: http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/ (cit. on p. 34).

[70] Jessica Thornsby. *Node.js Moves to Joyent*. Nov. 2010. URL: http://jaxenter.com/node-js-moves-to-joyent-32530.html (cit. on p. 34).

[71] *Twitter Bootstrap*. URL: http://getbootstrap.com/ (cit. on p. 23).

[72] *V8 (JavaScript Engine) Wikipedia Entry*. URL: http://nodejs..org (cit. on p. 33).

[73] *Vinyl records (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Vinyl_records (cit. on p. 2).

[74] *Virtual Studio Technology (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Virtual_Studio_Technology (cit. on p. 16).

[75] *W3C: HTML Canvas 2D Contex Candidate Recommendation*. URL: http://www.w3.org/TR/2013/CR-2dcontext-20130806/#drawing-images-to-the-canvas (cit. on p. 25).

[76] *W3C: HTML5 Candidate Recommendation*. URL: http://www.w3.org/TR/2014/CR-html5-20140204.

[77] *W3C Technical Report Development Process*. URL: http://www.w3.org/2005/10/Process-20051014/tr#q73 (cit. on pp. 25, 27).

[78] *W3C Wikipedia Entry*. URL: https://en.wikipedia.org/wiki/World_Wide_Web_Consortium (cit. on p. 22).

[79] *Web Application (Wikipedia Entry)*. URL: https://en.wikipedia.org/wiki/Web_application (cit. on pp. 19, 20, 23).

[80] *Web Audio API browser support*. URL: http://caniuse.com/#feat=audio-api (cit. on p. 30).

[81] *WebPageTest*. URL: http://www.webpagetest.org/ (cit. on p. 61).

[82] *WebRTC Project Homepage*. URL: http://www.webrtc.org/ (cit. on p. 27).

[83] *"Why is Node.js becoming so popular?" - Quora Question*. May 2011. URL: http://www.quora.com/Node-js/Why-is-Node-js-becoming-so-popular (cit. on p. 35).

[84] *Yeoman - Modern workflows for modern webapps*. URL: http://yeoman.io/ (cit. on p. 48).