UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

TESI DI LAUREA

Remix Culture: development of a component based multimedia editing platform

Relatore:
Chiar.ma Prof.ssa Maria Simi

Candidato:
Lukasz Szczygiel

Correlatore:
Chiar.mo Prof. Vincenzo Gervasi

ANNO ACCADEMICO 2019/2020

Contents

List of Figures

Introduction

The Remix Culture is a transformative practice that affects a whole spectrum of creative works. In the digital world, thanks to the Internet pervasiveness, everyone can become a content creator by using and arranging existing elements from various sources. This practice – called "remixing" – can radically change the meaning of the original works during the transformative passage from the "old" into the "new". Nowadays, this is substantially limited by various laws related to the Intellectual Property Rights. On the other hand, whether it is legal or not, this is what commonly happens in the digital environment. People make various forms of remixes, for example, parodies, memes, mash-ups, and other derivative works. The production of these is enabled by software and stimulated by a number of social platforms which offer these new forms of interactivity. On this regard this dissertation also shows the development of a web application that aims to further explore the possibilities of remixing digital items. More generally, it seems that remixing can also be extrapolated from a number of other disciplines and physical works, thus leading to a thesis which correlates this sort of behaviour to the human nature itself.

This dissertation aims at making an in-depth analysis of the Remix Culture phenomenon. The benefits and the potential new applications together with some factors limiting its expansion will be discussed in detail later on. Moreover, a review of the state of the art and some proposals about alternative solutions to copyright issues such as Open Source and Creative Commons are some of the topics that will be covered in the first chapter.

Subsequently, an overview from the management perspective with considerations about the sustainability of the Open Source organisations and some best practices for project development will be demonstrated in order to connect the theory to the practice.

Finally, a practical example of a web application that enables remixing was made. The project consisted in creating a multimedia editor for audio-visual content. Thanks to this solution users are able to explore and choose multiple media types, namely, videos, images, and audio items from a library of elements. These media can then be viewed, arranged between one another, and inserted inside a multi-tracked editor. Analogously to some popular solutions for video-editing, this allows for real-time preview of the combined elements. The peculiarity or the main software value proposition is that this application works directly inside all the modern browsers. In particular, the development process will be documented alongside with the explanation of the technical choices and the current software tendencies. Interestingly, this last chapter also advocates for remixing due to the adopted architecture based on reusable components with encapsulated functionalities. From the software perspective the component-based approach allows for interoperability thanks to the adoption of a standardised Web Components technology. In turn, this opens the possibility of combining single components just as LEGO blocks or puzzles. Indeed, this sort of new combinations can be made to produce new applications.

The theme of this work is also connected to the *PH-Remix* project. The acronym stands for "Public History Remix" and it involves the creation of a platform with the subsequent upload of resources belonging to the "Mediateca-Toscana" regional foundation. This project's technical solution aims at exploring new ways of content dissemination, discovery, use and re-use which may enhance the value and the public utility of the uploaded resources. Furthermore, an important element of innovation consists in the automatic extraction of short clips from the videos by using Artificial Intelligence algorithms. For instance, at the time of writing, the uploaded content consists mainly of documentaries. Once uploaded to the platform, the algorithms are able to perform the object recognition task. This task returns a set of recognised objects, for example, people, furniture, ships, animals and many more together with the video frames containing the recognised objects. Hence, these video frames or video clips become short thematic extracts of the original media. Ultimately, they can be offered to the users for exploration and arranged inside the multimedia editing component.

To summarise, the *PH-Remix* main objectives can be framed into the cultural heritage perspective of digitization of audio-visual resources. It is an attempt to enhance content fruition by encouraging innovative ways of user participation and content discovery. *PH-Remix* is an ongoing project which involves, at the time of writing, the activity of three researchers from three different areas. Namely: archive's analysis and cataloguing, data extraction and artificial intelligence, user experience and user interfaces. Although the development of the web-based multimedia editor originated from the requirements of the *PH-Remix* project, subsequently it was adapted to fit a general-purpose scenario with a solution that can be modified and expanded upon according to the use case. Nevertheless, the project will be considered as a study case across all the chapters of this dissertation.

To review, the dissertation is logically divided into three chapters. The main theoretical concepts with examples and copyright issues are explained in chapter 1 *Remix Culture*. Considerations about project management and Open Source sustainability are discussed in chapter 2 *Project Management and interdisciplinary works*. The application development process with application examples is inserted in chapter 3 *Project development web application for multimedia editing*.

1. Remix Culture

Remix Culture is the overarching theme of this dissertation. This phenomenon encompasses a large number of domains and aspects of everyday life involving creativity, thus shaping the way these works are being created, shared, used, and reused. Indeed, the fruition and the reuse are the most characterising facets of the Remix concept.

As mentioned in the introduction, Remix Culture is a transformative practice. By taking one or more pieces of existing items – both physical and digital – new works can be created. For instance, the replacement of an audio source within a certain video or more complex arrangements like entire movies made by using other audio-visual elements are both valid examples of transformative practices. These kinds of remixes can radically change the meaning and the cultural perception of the original works.

It appears that the Remix Culture can be seen from two high-level perspectives, the first tied to the digital evolution. Indeed, among the inspirations of the Remix Culture is the Free and Open Source-Software ("FOSS"), an initiative which will be introduced in paragraph 1.2 *Open Source* as a precursor to the Open Source movement. In short, the goals and ideas of these movements are mainly oriented towards encouraging a "free" distribution of software. "Free" is a vague term that will have to be discussed later in more details.

A second perspective is not exclusively tied to the digital evolution, rather it can be seen as a sociocultural concept that affects several aspects of human behaviour. From a broader perspective, the Remix Culture can be thought of as a way the society works and evolves through history. The idea can be explained by using the popular metaphor of "standing on the shoulders of giants" in the sense used and intended by Isaac Newton[1]. Therefore, the ability to discover and enhance knowledge – hence progress as humans – can be reached thanks to the contributions of past and present human generations with the knowledge that had been passed to us.

[1] *BBC Moving Words - Sir Isaac Newton.* URL: https://www.bbc.co.uk/worldservice/learningenglish/movingwords/shortlist/newton.shtml. (Accessed May 17, 2021).

This concept is further sustained by Lev Manovich in his article “Remixing and Remixability”[2]. The author argues that:

> “[...] most human cultures developed by borrowing and reworking forms and styles from other cultures; the resulting “remixes” were to be incorporated into other cultures. Ancient Rome remixed Ancient Greece; Renaissance remixed antiquity; nineteenth century European architecture remixed many historical periods including the Renaissance [. . .]”.

This excerpt suggests a definition of remixability as a common trait of human behaviour as deducted from a historical perspective. This can be generalised to humans as communities[3], as well as individuals. The latter view becomes the core point of author’s subsequent analysis when describing the modern times. The core point of his argumentation is the major novelty introduced by the Internet, that is the unprecedented level of participation of individuals with the relative ease of accessibility. Indeed, related to this aspect, he states that “culture has always been about remixability, but now this remixability is available to all participants of Internet culture”[4]. Nowadays, the pervasiveness of Internet connectivity has led to a professionalization of individuals. Thanks to information available on the web combined with software tools that allow to produce new content, the individuals can become active members of the Remix Culture. Finally, acting as creators, members of the public are able to share their work almost instantly, everywhere and at little or no cost.

As a matter of fact, this reflects the tendency of new social interactions enabled by the Internet. Many of the most popular online platforms, like Youtube, TikTok, Instagram, etc. make extensive use of the Remix Culture. In the next section 1.1 *Examples of Remixes*, some practical examples of remixes from various disciplines and time frames, ranging

[2]Lev Manovich. “Remixability and Modularity”. In: (2005), p. 2. URL: http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf.

[3]*Sense of Community or psychological sense of community*. URL: https://en.wikipedia.org/wiki/Sense_of_community. (Accessed May 17, 2021).

[4]Lev Manovich. “Remixability and Modularity”. In: (2005), p. 7. URL: http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf.

from texts, music, films to videogames will be presented to better illustrate the whole phenomenon.

Before expanding further in framing Remix Culture, a first glance on common issues can be taken by reflecting upon this rather radical change of paradigm. Hence, the change from an analogic and physical to a digital environment. It could be argued that the national and international laws have not responded adequately to this change of paradigm. According to Lawrence Lessig, who may be considered among the most prominent figures in favour of this argument, states that:

> "For the first time, the [copyright] law regulates ordinary citizens generally. For the first time, it reaches beyond the professional to control the amateur— to subject the amateur to a control by the law that the law historically reserved to professionals."[5]

This suggests an underlying problem from a legislative perspective, thereby the copyright law. Further evidence will be provided in paragraph 1.2 *Intellectual Property Laws and Reusability*. The overall point is that by analysing the past and current evolution of works of creativity made by "amateurs" i.e., general public, it seems that the law has taken some stringent measures to limit the Remix phenomenon. This is also true for professionals and large businesses. Additionally, some uncoordinated decisions have been made to respond to certain issues with the practical applications of copyright laws. This overall might be considered as an inhibitory factor acting against the Remix Culture ideas and practical applications.

Another interesting point of view related to the concept of Remix Culture is given by the aforementioned Lawrence Lessig, the founder of Creative Commons. In his book "Remix Making Art and Commerce Thrive" he defines Remix Culture as a Read/Write ("RW") culture as opposed to a Read/Only ("RO") culture. The author uses an analogy of how popular computer systems work (for example Linux based systems, Windows etc). Basically, users with read and write permission on a file or directory are authorized to read it and make changes. On the contrary, the read only permission allows only for files and

[5] L. Lessig. "Remix: Making Art and Commerce Thrive in the Hybrid Economy". In: 2008, p. 103.

directories to be read disabling the ability of making any modifications.

Returning to the cultural context, the latter is the situation where the users or the general public passively consume the content made by someone else. This is frequently the case of professionally made content, backed by expensive tools and potentially complex organisational structures. For example, the TV broadcasts, production of high budget movies and their subsequent distribution on CDs, DVDs, etc. It could be argued that the Read/Only culture lasted until the times of Internet pervasiveness, as it is characterised by a top-down consumption of cultural objects. On the other hand, the Read/Write culture allows for a democratized participation on the creative process. Hence, a democratization of the act of creativity where people participate in the creation and the re-creation of culture. These arguments represent a paradigm switch. In the Read/Only culture information flows one-way – unidirectionally –, while in the Read/Write culture the information is multi directional or, speaking in terms of networking: peer to peer like[6].

At this point, it should be clear that the technological progress allowed to minimize the gap between the professionally created content and the works of creativity that can be made without substantial investments or infrastructure. This can be accomplished thanks to software that makes the technical operation of "remixing" relatively easy.

This concept is also exemplified by Lev Manovich in relation to music remixes and music mashups. In his article "Remix and remixability" he states that: "Although precedents of remixing in music can be found earlier, it was the introduction of multi-track mixers that made remixing a standard practice"[7] [8].

Hence, what really made the Read/Write culture come to life was the 21st century digital revolution. Indeed, there are some relevant differences between the way things could be created, re-used, and shared before the age of Internet. A notable example showing these differences is the invention of the mechanical movable-type printing press, that is an efficient machine for printing texts. The main difference between the printing press and

[6] *Peer to peer*. URL: `https://en.wikipedia.org/wiki/Peer-to-peer`. (Accessed May 17, 2021).

[7] Lev Manovich. "Remixability and Modularity". In: (2005), p. 3. URL: `http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf`.

[8] This specific example is somewhat related to this dissertation's practical application. Instead of music as exemplified by Lev Manovich any type of media can be put into a multi-track editor to create new remixes.

the Internet is that the printed output copies were inferior to the originals in terms of quality. The affordability was also an issue, costs related to printing a copy prevented people to fully benefit of that invention. Another aspect of this problem is that people had little choice regarding the production of copies for themselves. The alternative choices – until the introduction of low-cost home prints – would imply an inferior quality to those coming from the professional printing businesses. Thus, the whole process relied on professionals. On the other side, the Internet can be seen as the most efficient, low-cost copying and sharing mechanism accessible from everywhere. Furthermore, the sharing process allows for resources to be distributed globally between connected users, and this is a substantial difference from the previous models. Therefore, users do not often need to rely on professionals to make creative works.

These differences can be further explained from an economical perspective as the distinction between rival goods and non-rival goods[9]. In general, most physical goods are rival goods. For example, a sandwich is a rival good, because the act of eating it clearly diminishes its value. A sandwich loses its value while it is being eaten, so to speak. Sharing rival goods implies the that the benefits of use are decreased or eliminated, so sharing a piece of sandwich produces a tangible loss for the owner of the sandwich. Analogously, sharing a printed copy of a book is an example of rivalry because it prevents the original owner from its unlimited consumption.

On the other side the definition of non-rival goods implies that: "[. . .] one person's enjoyment of a good does not diminish the ability of other people to enjoy the same good"[10]. Hence, ideas and words tend to be non-rival. Sharing them does not make them worse or less valuable. An exception of a physical good that is non-rival could be a public bench. The bench value is not significantly diminished when used by people (at least, hopefully, if it is being used correctly). From this perspective a large majority of digital goods are non-rival. A relevant example is an e-book, no one ever "bought" an e-book, people buy a license to read e-books on their devices of choice. Namely, the use of copies does not

[9] *Rivalry (economics)*. URL: `https://en.wikipedia.org/wiki/Rivalry_(economics)`. (Accessed May 17, 2021).

[10] Matthew Kotchen. "Public Goods". In: (2012). URL: `https://environment.yale.edu/kotchen/pubs/pgchap.pdf`.

preclude its accessibility from others and is not directly connected to tangible losses. A few exceptions like domain names and so on exist in this category as well.

Expanding upon the non-rivalry definition, in his book “The Success of Open Source”[11], Steven Weber argued that some ideas, words, etc. are definable as “anti-rival goods” meaning that they are improved by being shared in a manner similar to the economics idea of the “Network effect”[12]. Some examples could be the case where a social media gets more powerful as more and more people are using it, or an article becomes more valuable as its hyperlink is being shared across the web, etc. In short, the act of sharing increases the benefits for other participants. For this same reason it could be argued that the efforts to combat climate change are non-rival because the benefits of these actions are shared among all the word’s living species, including for example all the nations who refuse to do so. This final point of analysis suggests that the Remix Culture could be in fact non-rival. This argument could be rephrased as a direct question, “are the works of creativity produced as a result of remixing beneficial for the general public?”

Unfortunately, the proposed answer is murky, since from a general point of view it would be very hard to answer with a simple “no” or “yes”. It might be argued that it depends on case to case and on the relative contexts where this question is made. Probably, the answer might not always be positive from the perspective of original creators or de facto owners of works whom creations might be used in a way that contradicts their believes, such as propaganda, commercial profits, etc. Nevertheless, the key point worth considering is that as a matter of fact, everything can be remixed and distributed globally.

In this view, a consideration on the *PH-Remix* project case is pertinent. Among the project’s goals are dissemination, discovery, use and re-use which may enhance the value of created content. From a practical point of view, short clips are extracted from films uploaded to the platform – mainly documentaries at the time of writing – using Artificial Intelligence algorithms. Subsequently, in an idealistic scenario all the clips should become available for remixing i.e., being arranged and combined with other clips to form new

[11] Steven Weber. *The Success of Open Source*. USA: Harvard University Press, 2004, pp. 153–154. ISBN: 0674012925.

[12] *Network effect*. URL: `https://en.wikipedia.org/wiki/Network_effect`. (Accessed May 17, 2021).

audio-visual creations.

However, the tendency of the film industry could be considered sceptical about these ideas or simply not fully aware about the possibilities and new ways of innovating provided by digitisation. Considering the nature of the content initially put on the platform, it could be argued that cultural heritage works and especially documentaries should oblige a moral and ethical motivation of giving back to the communities and not be closed in organisational siloes protected by stringent laws. Indeed, *PH-Remix* for cultural heritage would be a case of non-rivalry because it should be desired to share the messages and stimulate a debate with as many people as possible with as little limits as possible.

Naturally, critiques about the idea of Remix Culture exist[13]. They will be further discussed in the next paragraph in relation to copyright issues. It could be argued that a relevant part of human progress alongside with some of the most iconic inventions were made thanks to the presence of copyright laws. Nowadays, by taking into consideration that the cost of distributing content is as low or almost nonexistent for digital goods it may still be necessary to reward the creators of intellectual property.

This objectively seems to be true. Remix culture can be enabled by openness and Open Source licenses, but these are not a silver bullet for all projects and creations. A relevant example was once provided during the "Open Cultures" course at King's College London by Dr John Lavagnino. He argued that J.K. Rowling could probably not have finished the Harry Potter books if she had decided to start publishing them without a license. A source of income is often needed and necessary. On the other hand, the more people who read Harry Potter the better. Generally, as a creator you probably would want to get as many people as possible to watch/be able to consult your creation. In chapter 2 *Project Management and interdisciplinary works*, arguments advocating for a sustainable economy for content creation – especially for cultural heritage – will be discussed in more detail.

[13]Terry Hart. *Remix Without Romance: What Free Culture Gets Wrong*. URL: `https://www.copyhype.com/2012/04/remix-without-romance-what-free-culture-gets-wrong/`. (Accessed May 17, 2021).

1.1 Examples of Remixes

The practice of modifying both physical and digital items has a long-standing tradition. Remix applies to music, movies, cooking recipes, software, and many other disciplines. After introducing the Remix Culture mainly from a theoretical point of view, it is useful to make some practical examples to better understand the overall concept. For this reason, a selected list of examples from different domains – although mainly focused on the modern age time frame – is presented in this section.

An in-depth study of the creative processes and the way content is being re-used was done by Kirby Ferguson in his four-part video series titled "Everything is a Remix"[14]. Indeed, he states that the acts of copying, transforming, and combining are the basic elements applicable at any level of creativity. Hence, the following assertion: "creativity is not magic, it happens by applying ordinary tools of thought to existing materials"[15].

Music mashups are one of the most common examples of such transformations. Specifically, hip hop music was one of the first musical forms to incorporate samplings in the recordings[16]. Thereby, the results are creative works which typically incorporate fragments of other songs. These fragments are generally re-arranged, thus transformed to produce new sounds and songs. Indeed, in most cases mashups are legal by considering them under the various boundaries of "fair use" law doctrine. These boundaries are usually fuzzy although the tendency of re-using pieces of other songs is very popular among many artists.

One example of a band which crossed boundaries of fair use is Led Zeppelin. They copied significant part of other songs without making fundamental changes. Two documented examples of songs subject to legal claims can be seen on the list below:

- Led Zeppelin song: "Bring it on home" was a copy of Willie Dixon – "Bring It On

[14]*Everything is a remix*. URL: `https://www.everythingisaremix.info/`. (Accessed May 17, 2021).

[15]Kirby Ferguson. *Everything is a remix - Part 3*. URL: `https://www.everythingisaremix.info/blog/everything-is-a-remix-part-3-transcript`. (Accessed May 17, 2021).

[16]*Mashup (culture)*. URL: `https://en.wikipedia.org/wiki/Mashup_(culture)`. (Accessed May 17, 2021).

Home”[17]

- Led Zeppelin – “Stairway to Heaven” most likely copied from the band Spirit with their song “Taurus”[18]

Another discipline which makes extensive use of Remix is the cinema industry. Many movies are inspired by the surrounding works of culture. Nowadays, a common technique consists in transforming the “old” into the “new”. That means taking or re-creating already existing materials from literature, actual events, etc. and producing movies for the current generations. Even existing movies are often a base for new cinema adaptations and frequent prequels and sequels.

One important example of remixing in the film industry are the Disney movies. The Walt Disney Company made extensive use of works from the public domain. Some examples from their repertoire of animated films history are listed below:

- The Little Mermaid is based on the “The Little Mermaid” fairy tale written by Hans Christian Andersen.
- Alice in Wonderland is based on the novel “Alice’s Adventures in Wonderland” and its sequel, “Through the Looking-Glass” both authored by Lewis Carroll.
- Aladdin is based on the folk tale “Aladdin from the Arabian Nights” originated from the Middle Eastern culture and later interpreted by Antoine Galland.
- Mulan is based on the traditional Chinese story of “Hua Mulan”, a legendary folk heroine.

Once transformed into animated movies, Disney likewise any other film producer, were entitled to a period of exclusivity on their productions thanks to copyright. Then after a certain period their work is supposed to enter the public domain to be freely used

[17] *Bring It On Home (song)*. URL: `https://en.wikipedia.org/wiki/Bring_It_on_Home_(Sonny_Boy_Williamson_II_song)`. (Accessed May 17, 2021).

[18] *Spirit copyright infringement lawsuit*. URL: `https://en.wikipedia.org/wiki/Stairway_to_Heaven#Spirit_copyright_infringement_lawsuit`. (Accessed May 17, 2021).

and built upon. This case was regulated by the American legislation in the Copyright Act of 1976[19] which established the duration of copyright for the life of the author plus 50 years, or 75 years for a work of corporate authorship.

Interestingly, some companies and prominently Disney lobbied to have their terms of copyright extended[20] with the Copyright Term Extension Act, also known as "Mickey Mouse Protection Act". The company decided to prevent others from copying its works. Therefore, standing to the current copyright terms, Mickey Mouse, which firstly appeared in 1928, will enter the public domain starting from 1st January 2024[21]. It could be argued that by trying to foresee into the future, Disney will attempt to prevent it from happening by using other laws including trademark protection and using precedents in their favour from other court verdicts.

Returning to the practical cases, the film industry is a rich source of similar examples. For instance, an in-depth analysis of the Star Wars series results in finding multiple references and inspirations from historical events, fiction literature, etc. as well as some copied elements from other film productions. It could be argued that without the influence of past creations Start Wars would not have been created.

The digital evolution alongside with the birth of software for video-editing allowed for a more accessible and complex remixing of new content outside of the professional world. Fan-made trailers, remixes, memes that spread virally are all consequences of the ease of use enabled by a mix of modern technology combined with user's creativity.

Certainly, a long list of important examples could follow. For example, Machinima[22] productions, videos generated from video games graphic engines are a relevant contribution to re-use. The core point is that the strength of these creations is also traceable to the communities of users that participate in the creative process and the consumption. Indeed,

[19] *Copyright Act of 1976*. URL: `https://en.wikipedia.org/wiki/Copyright_Act_of_1976`. (Accessed May 17, 2021).

[20] *Copyright Term Extension Act*. URL: `https://en.wikipedia.org/wiki/Copyright_Term_Extension_Act`. (Accessed May 17, 2021).

[21] *Mickey Mouse in the public domain*. URL: `http://copyright.nova.edu/mickey-public-domain/`. (Accessed May 17, 2021).

[22] *Machinima*. URL: `https://en.wikipedia.org/wiki/Machinima`. (Accessed May 17, 2021).

there are several examples of Remixes as community efforts.

Perhaps the most well-known and successful community effort is Wikipedia. Wikipedia's main power is a huge community of active volunteers thanks to which it was able to succeed.

Other projects like Free Beer[23] and OpenCola[24] are examples of remixing in the physical world. They respectively consist in creating and improving recipes for beer and (coca) cola and encouraging their production by adopting a permissive license to the recipes. In reality, the latter example is connected to the Open Source community as an attempt to replicate the concepts of free sharing and contributing into the physical world which was already a common practice in the software world.

Naturally, Remix could be viewed as a form of art. Like the latter it is also subject to personal interpretation because, among its goals, is the creation of spaces for debate. Obviously, they often might be a source of tensions and critiques, as the example below.

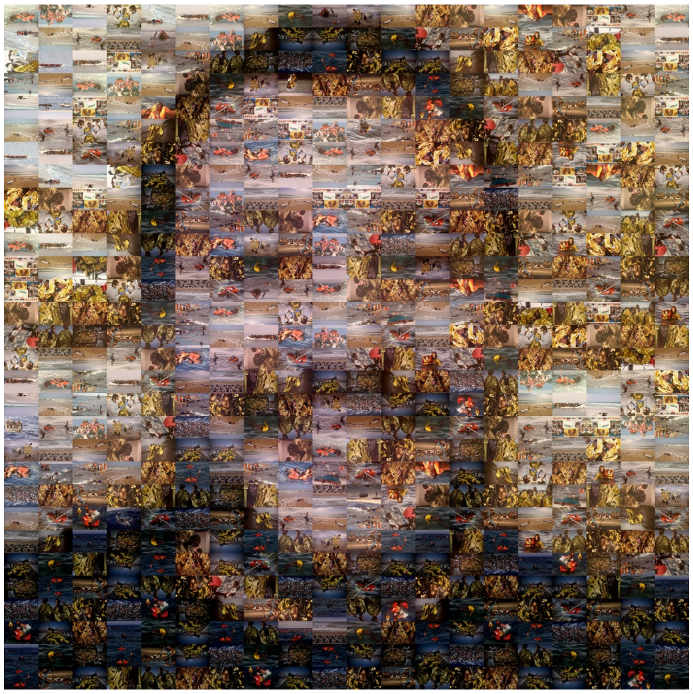


Figure 1.1: "Arte dell'acqua" by students of Russoli High School (Pisa)

[23] *Free Beer (project)*. URL: `http://freebeer.org/blog/`. (Accessed May 17, 2021).

[24] *OpenCola (project)*. URL: `https://en.wikipedia.org/wiki/OpenCola_(drink)`. (Accessed May 17, 2021).

Figure 1.1 ”Arte dell’acqua”[25] is a collage of approximately 400 photos of migrants which depict some tragedies happening at sea while trying to reach the coast. Together the pictures form the face of an Italian politician, Matteo Salvini. He is known for his ideological line against illegal immigration as he advocates for stronger actions and policies to enforce its prevention. Indeed, these forms of remixes can inevitably create controversies. Nevertheless, art encourages reflection and debate and above all it is an expression of free speech.

Lastly, copying is not necessarily equivalent to plagiarism. For instance, this whole dissertation could be seen as a remix. Originated by combining and arranging a set of factual sources with quotations alongside with the author’s academical background, personal experiences, and cultural bias to produce something new. While this last example of remix is perfectly legal – if it complies with some rules about academical quotations, etc. – and thus cannot be considered as a plagiarism, the same cannot be said about other contexts and media types.

1.2 Intellectual Property Laws and Reusability

Understanding the concepts related to intellectual property (“IP”) and how they are regulated, is fundamental in order to acquire a complete picture of the Remix Culture.

Naturally, the activity of Remix is deeply interconnected with a multitude of laws. It is fair to debunk upfront that an extensive analysis of the topics presented in this section is beyond the scope of this work. These are complex topics where legal terms involve the use of legal technicalities which differ from country to country. However, understanding the fundamentals of concepts like copyright and licensing is fundamental to explain and to illustrate some frequent problems occurring when adopting the Remix Culture.

As introduced in the previous paragraph, the re-use of content can be substantially limited and potentially discouraged because of the legal bounds applicable to a multitude of physical and digital goods. These legal bounds are typically expressed in form of author’s

[25] *Studio Gennai - L’arte dell’acqua.* URL: `https://www.studiogennai.it/larte-dellacqua/`. (Accessed May 17, 2021).

rights – or copyright, as these terms are used interchangeably in this dissertation – and can be contained in licences applicable to different kinds of works. In short, by creating new works authors are automatically entitled to exclusive rights on them and, as long as a permissive license is not applied, these works cannot be freely reused by others.

In reality, this might often work as a chilling effect, that is, "a discouraging or deterring effect, especially one resulting from a restrictive law or regulation"[26]. Furthermore, this effect can also be attributed to the quantity and the complexity of regulations affecting the specific items that might be of interest for re-use. A relevant argument in favour of this assertion comes from a phenomenon called proliferation of licenses[27]. This is originated from the creation of a large number of similar agreements that might require some domain specific knowledge for their interpretation. Additionally, licenses – whose main goal is to clearly explain the rights and limitations applied to a specific object – are often incompatible between each other, even if they are similar in their permissiveness or strictness.

The subsequent analysis will be focused on illustrating an overview and developing an understanding of the legislative implications of some legal terms that are relevant for the Remix Culture.

Firstly, "Intellectual Property Rights" ("IPR") is an umbrella term for a category of rights which regulate intangible creations of the human intellect[28].

> "The main purpose of intellectual property law is to encourage the creation of a wide variety of intellectual goods. To achieve this, the law gives people and businesses property rights to the information and intellectual goods they create, usually for a limited period of time."[29]

[26] *Definition of chilling effect from the Collins English Dictionary.* URL: `https://www.collinsdictionary.com/dictionary/english/chilling-effect`. (Accessed May 17, 2021).

[27] *License proliferation.* URL: `https://en.wikipedia.org/wiki/License_proliferation`. (Accessed May 17, 2021).

[28] Patrick Moorhead. *You Will Care About Intellectual Property Sooner Or Later.* URL: `https://www.forbes.com/sites/patrickmoorhead/2017/10/05/you-will-care-about-intellectual-property-sooner-or-later/?sh=4c0ba0471593`. (Accessed May 17, 2021).

[29] *Intellectual Property Rights-uses.* URL: `https://www.longdom.org/peer-reviewed-journals/`

As seen in the above explanation, the reason for restricting the IPR duration to a limited period of time is mainly motivated by the goal of encouraging further creations. Hence, during the time of "exclusiveness", authors can earn profit which in turn acts as an economic incentive for creation in the first place.

Secondly, speaking purely in terms of IPR might be considered as an overgeneralization. Richard M. Stallman argues that including different sets of laws under the IPR term is misleading[30]. Nevertheless, the subject's literature about IPR commonly includes a number of typologies of intellectual property belonging to different branches of law. Namely, rights, patents, copyright, industrial design rights, trademarks, registered designs, trade secrets and so on, also depending on the jurisdiction.

As introduced in the previous paragraph, Remix Culture could potentially be influenced by all of these laws. By taking into consideration the most common use cases a subset of IPR typologies can be taken for a deeper analysis. Specifically:

- Patents
- Trademarks
- Copyrights

Patents are used for protecting inventions. Namely, creativity works that are new, not trivial (for experts in the subject/domain) and are useful, thus capable of being used in some kind of industry. Patents give their owners the right to prevent others from making, using, selling an invention without an explicit permission. For example, scientific or mathematical discoveries cannot be patented, as well as a literary, dramatic, musical, or artistic work and many others. On the other side new plant varieties, medicines, machines, innovative solutions to technological problems are generally valid targets for patents. An ongoing debate regards the creation of software. The European legislation regulated by the European Patent Convention explicitly excludes the possibility of patenting "computer

`intellectual-property-rightsuses-1012.html`. (Accessed May 17, 2021).

[30]Richard M. Stallman. *Did You Say "Intellectual Property"? It's a Seductive Mirage*. URL: `https://www.gnu.org/philosophy/not-ipr.en.html`. (Accessed May 17, 2021).

programs" as stated in Article 52 "Patentable inventions"[31]. Still, it is subject to interpretation by courts. Vice versa the American legislation tend to allow software patents.

Certainly, from the point of view of the Remix Culture, software patents could have long-term negative effects. Namely, as stated by the Free Software Foundation Europe, "they specifically inhibit the development of useful software by blocking compatibility and interoperability"[32].

Secondly, trademarks are some sort of signs that distinguish a company or a service from another. They are also frequently used for marketing purposes like branding, product recognition, etc. Logotypes are a classical example of a trademark. For instance, the Nike "swoosh", the Apple "bitten apple" and McDonalds "golden arches" are among the popular examples of trademarks.

Finally, copyrights regulate that works cannot be copied without the explicit owner's permission. This applies to any medium. For example, nobody can make a movie based on a book without obtaining the permission of the owner of the book copyright. Copyright can protect literary works, including novels, instruction manuals, computer programs, song lyrics, newspaper articles, and some types of databases. Perhaps the most important characteristic of copyright is that it does not have to be applied for. Everyone gets it even if not aware of it.

These three are just some selected examples that creators should be aware of when doing remixes. Interestingly, it seems that the just introduced concepts are similar between the physical word and the digital world. Therefore, it could be argued that they are a straightforward translation of legislative terms originated from the history rather than new concepts adapted for the 21st century. This can be explained by considering that the technological progress happened very fast and especially the Internet was an abrupt revolution. In his article "The fight to keep ideas open to all" James Boyle, professor of Law at Duke University, analyses the current restrictions of IPR in a digital environment. He also states that:

[31] *The European Patent Convention - Article 52*. URL: `https://www.epo.org//law-practice/legal-texts/html/epc/2020/e/ar52.html`. (Accessed May 17, 2021).

[32] *Free Software Foundation Europe - Software Patents in Europe*. URL: `https://fsfe.org/activities/swpat/swpat.en.html`. (Accessed May 17, 2021).

> "The internet has dramatically lowered the cost of copying, including illicit copying. When the web was first weaved in the 1990s, intellectual-property owners found their property had, involuntarily, been turned into a common. Strong new copyright rules and draconian enforcement seemed to be necessary to tame the rebellious digital commoners and reclaim the level of control that had existed in an analogue world."[33]

As seen in the above quotation, there are aspects that may justify some actions and fears of the copyright holders. In the recent years, policymakers, faced by arguments and lobbying, decided to adopt, and extend a copyright model which currently may seem as more and more unfit for the technological progress.

Nowadays, this led to a growing quantity of issues. In the same article James Boyle affirms that our smartphones are covered by between 5,000 to 15,000 patents and up to 250,000 if considering all the related patents. The general idea of scholars and professionals advocating for a more liberal approach to digital right is that intellectual property in the actual form blocks innovation and can be harmful for the society.

An example, relevant especially at the time of writing, might regard life-saving creations like vaccines[34]. As finally stated by Boyle: "In medicine, rafts of patents obstruct research into treatments such as a malaria vaccine; the costs of just identifying the relevant patents is [sic] prohibitive. Such a system is great for patent lawyers, whom I train. It is unlikely to be good for society as a whole."[35] [36]

Returning to the digital objects, the copyright laws evolved in terms of their enforce-

[33]James Boyle. *The Economyst - The fight to keep ideas open to all.* URL: `https://www.economist.com/open-future/2018/12/12/the-fight-to-keep-ideas-open-to-all`. (Accessed May 17, 2021).

[34]*Creative Commons' Response to COVID-19.* URL: `https://creativecommons.org/creative-commons-response-to-covid-19/`. (Accessed May 17, 2021).

[35]James Boyle. *The Economyst - The fight to keep ideas open to all.* URL: `https://www.economist.com/open-future/2018/12/12/the-fight-to-keep-ideas-open-to-all`. (Accessed May 17, 2021).

[36]Monisha Ravisetti. *The Academic Times - First vaccine to fully immunize against malaria.* URL: `https://academictimes.com/first-vaccine-to-fully-immunize-against-malaria-builds-on-pandemic-driven-rna-tech/`. (Accessed May 17, 2021).

ment. Indeed, they can also be efficiently prosecuted – although sometimes wrongly, by using technology. A relevant example is the YouTube Content ID software[37] which automatically checks for the presence of copyrighted materials. This happens when users upload their media to the YouTube platform.

> "The software establishes a link between an existing work and an uploaded work such as a remix. If the content matches, the video may be automatically blocked, or the sound muted, and the user is automatically informed by e-mail that the material has been disabled".[38]

From this perspective, it might be deducted that a potentially dangerous direction would consist in maintaining the current state of IPR and enforcing it through the use of technology. This could be especially harmful for the Remix Culture phenomenon.

1.2.1 Derivative Works and Fair Use

As introduced in the previous section, the Remix Culture often faces problems regarding the legal bounds. Namely, the Intellectual Property Rights. On the other hand, the law also imposes rules and exceptions under which legal works can be created. Specifically, the concepts of derivative works and fair use are key for limiting the application extent of IPR. In the practical applications of remixes, it seems that a common source of problems derives from how contents coming from a variety of sources are used and distributed for the creation of new works. Indeed, this discourse could be framed into the legal perspective by using the term of "derivative work". By considering the definition from the Merriam Webster legal dictionary, a derivative work is defined as a "a piece of intellectual property that substantially derives from an underlying work"[39].

[37] *Google support - How Content ID works*. URL: `https://support.google.com/youtube/answer/2797370?hl=en`. (Accessed May 17, 2021).

[38] Guilda Rostama. *WIPO Magazine - Remix Culture and Amateur Creativity: A Copyright Dilemma*. URL: `https://www.wipo.int/wipo_magazine/en/2015/03/article_0006.html`. (Accessed May 17, 2021).

[39] *"Derivative work." Merriam-Webster.com Legal Dictionary*. URL: `https://www.merriam-webster.com/legal/derivative%20work`. (Accessed May 8, 2021).

From a general perspective, the notion of derivative works seems to be consistent across many national and international legal systems. Albeit the usage of the adverb "substantially" is the key difference leading to different interpretations as explained with further details later in this section. For instance, the United States Copyright Act states that, a "derivative work is a work based upon one or more preexisting works [...]"[40]. This definition is followed by a list of actions and types of works that make a derivative work as such. Along these lines, the 2nd article of the International Berne Convention, entitled "Protected Works", also refers to the derivative works together with the criteria for its protection. "Translations, adaptations, arrangements of music and other alterations of a literary or artistic work shall be protected as original works without prejudice to the copyright in the original work"[41].

Nevertheless, it seems that international and national legal systems are not sufficiently precise when defining derivative works. As a matter of fact, it is very complex to establish which copyrighted materials and above all to what extent, can be used in a new creation without violating the author's laws. In this view the Remix Culture could be defined as a culture or society where the production of derivative works is allowed and therefore encouraged. Overall, at the time of writing it appears that this is not the case.

One last concept regards the fair use. Specifically, it is the part of the copyright law that allows for free speech (at least in the context of the American First Amendment). Thanks to fair use small amounts of copyrighted materials can be used to make an argument without asking the copyright holder for permission. As stated in the section 1.1 *Remix examples*, this dissertation could be thought of as remix. This is the case of text; the same degree of freedom is illegal in, for example, the filmmaking industry. Limitations like those valid for many academic books – stating that around 15% of the total pages can be photocopied – do not exist for other digital objects. This prompts an open question about the possibility of imposing similar limitations to digital objects. This would signifi-

[40]*17 U.S. Code § 101 - Definitions*. URL: `https://www.law.cornell.edu/uscode/text/17/101`. (Accessed May 17, 2021).

[41]*Berne Convention for the Protection of Literary and Artistic Works*. URL: `https://wipolex-res.wipo.int/edocs/lexdocs/treaties/en/berne/trt_berne_001en.html`. (Accessed May 17, 2021).

cantly reduce ambiguities and issues with the interpretation, but from a practical point of view may be unfeasible. According to the European law the concept of fair use is fuzzy and does not provide clear guidelines for those who would want to remix content.

An important contribution expanding upon the problems of copyright is RIP: A Remix Manifesto. It is a documentary about "the changing concept of copyright", and its main point is to illustrate that copyright is not under control[42]. Therefore, advocating for the so called copyleft as opposed to copyright. As a matter of fact, even with tools like derivative work protection and fair use, it is not enough to truly incentivize creativity and producing a rich public domain open to all.

Looking at the state of the art, the introduction of the recent European copyright directive (the directive on copyright and related rights in the Digital Single Market, in short, DSM directive) 2019/790 EU[43] was supposed to give clarity by regulating many of the edge cases by adapting the IPR to the digital environment. At the time of writing, it might still be too soon to understand the implications it will have for the digital market and the prosperity of the Remix Culture. Moreover, some studies argue that the DSM will not provide significant improvements for the digital age. "For sure, legal and practical consequences cannot be properly evaluated at this stage, as it is necessary to wait for the implementation of the Directive by all Member States"[44]. The directive provides some limited options in terms of specific content like those belonging to the cultural heritage. For example, the Art. 6. concerns the acts of reproduction of some cultural heritage institutions' works, but it is only valid for preservation purposes.

As stated before, it is difficult to understand the boundaries until the first interpretations are given in courts. The recent history shows that the interpretation of the directives about copyright can give some practical hints for the interpretative direction that will be taken.

[42]*RiP!: A Remix Manifesto*. URL: `https://en.wikipedia.org/wiki/RiP!:_A_Remix_Manifesto`. (Accessed May 17, 2021).

[43]*Directive (EU) 2019/790 on copyright and related rights in the Digital Single Market*. URL: `https://eur-lex.europa.eu/eli/dir/2019/790/oj`. (Accessed May 17, 2021).

[44]Federico Ferri. "The dark side(s) of the EU Directive on copyright and related rights in the Digital Single Market". In: *China-EU Law Journal* (2020). URL: `https://doi.org/10.1007/s12689-020-00089-5`.

There are multiple examples of existing cases that show what is legal and what tend not to be. Some interesting examples are the "Tom Cabinet case" (regarding the reselling of second-hand e-books)[45] and "Nils Svensson and Others v Retriever Sverige AB" (case on linking/hypertext that redirect Internet users to protected works available on other websites without the authorisation of the copyright holder)[46].

Until this situation is not fully transparent, a valid solution to reduce the risks may come from the use of Open Source and permissive licenses. These will be discussed in the subsequent two sections.

1.3 Open Source

Generally, the topic of "Open Source" tends to be well recognised within the discipline of computer science and similar, arguably less within other fields. This is mainly due to the fact that it was originated in the context of software development, and thus among computer programmers. However, it is relevant for the Remix Culture not only due to the pure software implications, but also from an organisational and legislative point of view. Thanks to the Open Source and the Creative Commons licensing a more transparent, and accessible picture of remixing will be introduced.

First of all, Open Source does not mean "free" as this can be a common misconception due to its cultural meaning. A brief excursus into the Open Source history ought to be made to understand and explain the terminology used in the subject's literature. Historically, around the 1970s with the popularization of personal computers – mainly due to their economic affordability – many programmers started contributing and sharing for "free" their source code with the programming community. However, after some years the progress of the computer industry and the related business models showed a tendency of making proprietary systems, that is closed source programs and systems. To contrast this phenomenon, ways of regulating the freedom of the source code were necessary for

[45] *Tom Kabinet case*. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:62018CJ0263&from=EN`. (Accessed May 17, 2021).

[46] *Nils Svensson and Others v Retriever Sverige AB*. URL: `https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:62012CJ0466`. (Accessed May 17, 2021).

legal reasons. From a practical point of view, a few developers would be interested in contributing to a system that eventually could become proprietary, hence loosing all their voluntary efforts.

For the record, the US Copyright Act was introduced in 1976 with the subsequent result of encouraging the development of proprietary software. For this reason, Richard Stallman founded the Free Software Foundation. In 1983 he defined the term "free" as a combination of four essential freedoms that must exist in order to define software as free[47].

- Freedom 0: The freedom to run the program as you wish, for any purpose.
- Freedom 1: The freedom to study how the program works and change it so it does your computing as you wish.
- Freedom 2: The freedom to redistribute copies so you can help your neighbour.
- Freedom 3: The freedom to distribute copies of your modified versions to others.

Subsequently another shift in terminology was deemed as necessary. According to Bruce Perens, the founder of the Open Source Initiative, on a business level the Free Software Definition was not well received. Apparently, managers and executives were not keen to investing on projects with the world "free" incorporated. "Free" is a difficult term, particularly in the anglophone languages, because it could mean "with no price" or "with no conditions". "The Free Software Definition defines the "free" in free software as being about liberty, not price: it is consistent with the principles of free software to sell copies"[48].

Therefore, speaking in terms of the aforementioned four freedoms, it would mean restricting any of those. Overall, the term "free" is historically overcharged with different connotations and this made the term "open" more adequate as a technical term. Therefore,

[47] *GNU - What is free software*. URL: `https://www.gnu.org/philosophy/free-sw.en.html`. (Accessed May 17, 2021).

[48] Jeffrey Pomerantz and Robin Peek. "Fifty shades of open". In: *First Monday* 21.5 (2016). DOI: `10.5210/fm.v21i5.6360`. URL: `https://journals.uic.edu/ojs/index.php/fm/article/view/6360`.

the shift from the term “free” to “open” began. Nevertheless, in this context the core meaning of both of these terms indicates that the software is the only thing that is gratis. “Just as with proprietary software, there is a “total cost of ownership” of free and Open Source software, which includes such costs as development, customization, maintenance, hardware, support, and many others”[49].

This is an important clarification as it specifies the contextualized nomenclature of these terms. Particularly, the issue with the term “free” is that it does not clearly reflect the costs connected to the effective software fruition. As a matter of fact, software is complex, thus obtaining it without cost could be thought of as a mere beginning.

For example, there are a lot of digital services that are not open. Services offered by companies like Amazon, Facebook, Google, etc. are mostly not open. They are based on terms of use which limit how these platforms can be used. Arguably, they are not even gratis as users pay by giving the ownership control on a set of their personal data. On the other hand, Wikipedia is definitely a success story of the Open Source. It is entirely open in terms of systems and content. It allows for reuse of its structure and data. Everyone can host their own Wikipedia page and modify it without restrictions. As a matter of fact, Wikipedia is based on the MediaWiki platform[50] which is a Content Management Platform (“CMS”) for open contributions and powers multiple community driven wikis, for example fandoms (“Wikias”) focused on various topics. Since thanks to MediaWiki anyone can copy, paste, install, and develop their own solution based on the platform, a straightforward comparison with the Remix Culture can be made. This is an example of an ideal scenario where the Remix Culture can be applied almost without any constrains.

Wikipedia is not the only case of a successful Open Source project. Returning to the aforementioned number of patents in smartphones, described in 1.2 *Intellectual Property Laws and Reusability*, it is safe to affirm that they would not work without the Open Source. One relevant example is the Android operating system which itself is based on

[49]Jeffrey Pomerantz and Robin Peek. “Fifty shades of open”. In: *First Monday* 21.5 (2016). DOI: `10.5210/fm.v21i5.6360`. URL: `https://journals.uic.edu/ojs/index.php/fm/article/view/6360`.

[50]*MediaWiki platform*. URL: `https://www.mediawiki.org/wiki/MediaWiki`. (Accessed May 17, 2021).

Linux. The promiscuity of proprietary solutions and Open Source is possible thanks to the software modularity. A principle whose practical demonstration is discussed in further detail in chapter 3 *Project development: web application for multimedia editing*. Some of the most used commercial systems depend on Open Source components. The Open Source model can also be considered as a valid approach from the managerial perspective. This will be further explained in the next chapter.

As far as this dissertation's project is concerned, the rationale behind making an Open Source multimedia editor could be that there are not many existing in-browser solutions that allows for video-editing. Systems like YouTube, Vimeo, Adobe Spark and similar are closed source and cannot be reused for the purpose of building personal solutions. Although the solutions proposed for this dissertation practical application are related to the front-end architecture, a more complex back-end solution can be connected for creating custom projects. Thanks to the components approach, a contribution to Open Source is made by making the software available to the community.

1.4 Creative Commons

Nowadays the usage of licensing has largely widespread thanks to the Internet. A license or copyright license "is a contract which grants certain rights to use a work or other protected materials"[51].

Back in the day, the aforementioned US Copyright Act from 1976, was perceived as too restrictive by the Open Source supporters[52]. This led to a creation of a non-profit organization called Creative Commons ("CC"). Founded in 2001 by Lawrence Lessig, it defines a set of licenses providing alternatives to the traditional copyright. Most importantly these licences allow for creators to specify upfront how and when their work can be used,

[51] Sarah Dominique Orlandi et al. *FAQs AUTHOR'S RIGHT, COPYRIGHT AND FREE LICENSES FOR CULTURE ON THE WEB*. Zenodo, Mar. 2021. DOI: `10.5281/zenodo.4608430`. URL: `https://doi.org/10.5281/zenodo.4608430`.

[52] Jeffrey Pomerantz and Robin Peek. "Fifty shades of open". In: *First Monday* 21.5 (2016). DOI: `10.5210/fm.v21i5.6360`. URL: `https://journals.uic.edu/ojs/index.php/fm/article/view/6360`.

shared, and remixed by others. As stated in the Creative Commons website, "[the CC] licenses give everyone from individual creators to large institutions a standardized way to grant the public permission to use their creative work under copyright law"[53]. Indeed, the existence of copyright laws is important for Open Source licensing in order to grant usage of something "free of charge" but requiring, for example, that the re-distributing would happen under the same terms. Thus, it enforces rules for something that comes without charging money. Without copyright it would be impossible to do that. Moreover, by defining these agreements using non-legalese, human-readable definitions the CC became a very powerful enabler of openness[54].

Indeed, the Creative Commons Licenses are based on four rights that can be combined to form a set of licenses. These four clauses are:

1. Attribution: All distributions of a work, and derivative works based upon it, must be credited to the creator of the work.

2. Non-commercial: Derivative works cannot be destined for commercial use.

3. Share-alike: Derivative work must be licensed under terms identical to those of the original work.

4. No Derivatives: A work may be redistributed, but only "unchanged and in whole;" no derivative works may be made based on it.

Subsequently the Creative Commons organisation defines six licenses[55] starting from the most open to the most restrictive.

The first two license types allow reusers to distribute, remix, adapt, and build upon the material in any medium or format, also allowing for commercial use. The only requirements are that an attribution must be given to the original creator (CC BY license) and the

[53]*Creative Commons - Licenses.* URL: `https://creativecommons.org/about/cclicenses/`. (Accessed May 17, 2021).

[54]Jeffrey Pomerantz and Robin Peek. "Fifty shades of open". In: *First Monday* 21.5 (2016). DOI: `10.5210/fm.v21i5.6360`. URL: `https://journals.uic.edu/ojs/index.php/fm/article/view/6360`.

[55]*Creative Commons - Licenses.* URL: `https://creativecommons.org/about/cclicenses/`. (Accessed May 17, 2021).

adaptations – thus any modification made to the original work – must be shared on under the same terms (CC BY-SA).

- Attribution - Attribution only (CC BY)

Figure 1.2: Attribution CC BY

- Attribution - Share alike (CC BY-SA)

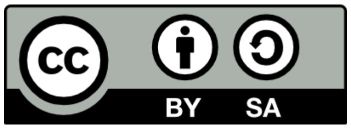


Figure 1.3: Attribution CC BY-SA

The next two license types allow reusers to distribute, remix, adapt, and build upon the material in any medium or format for non-commercial purposes only. Analogously to the first two types a credit must be given to the creator (CC BY-NC) and the adaptations must be shared under the same terms (CC BY-NC-SA).

- Attribution - Non-commercial (CC BY-NC)

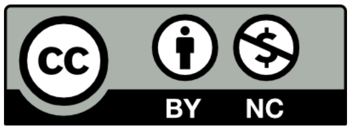


Figure 1.4: Attribution CC BY-NC

- Attribution - Non-commercial - Share alike (CC BY-NC-SA)

Figure 1.5: Attribution CC BY-NC-SA

The following license differs from the aforementioned types since it forbids the production of derivative works. Thus, adaptations are not allowed, but the work can still be used commercially (as long as credit is given to the author).

- Attribution - No derivative works (CC BY-ND)

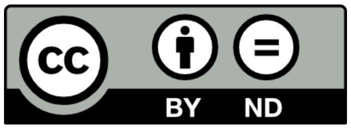


Figure 1.6: Attribution CC BY-ND

Similarly to the above CC BY-ND license, the following one works exactly the same, with the exception that it does not allow for the commercial reuse.

- Attribution - Non-commercial - No derivative works (CC BY-NC-ND)

Figure 1.7: Attribution CC BY-NC-ND

Finally, another tool called “Public Domain Dedication” or “CC0” allows for waiving all the related rights and effectively put the work into the worldwide public domain.

Figure 1.8: Public Domain Dedication

“CC0 allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, with no conditions.”[56]

[56]*Creative Commons - CC0*. URL: `https://creativecommons.org/publicdomain/zero/1.0/`. (Accessed May 17, 2021).

The CC licenses do not contain specific terms about the distribution of source code because they are mainly focused on writings, visual arts, and other creative works. Alternative forms of permissive licensing specifically made for software production also exist. One source containing the range of licenses that are around is: “Various Licenses and Comments About Them”[57] by the Free Software Foundation which provides a list of software licenses with explanations.

By combining the Open Source and Creative Commons the Remix Culture can thrive with new works of creativity which are perfectly legal. An ideal solution would be to propagate the ideas and an understanding of the Creative Commons licenses to encourage their usage among all levels of actors hoping that the openness will create a virtuous cycle that benefits everyone involved.

[57] *GNU - Licenses*. URL: `https://www.gnu.org/licenses/license-list.html.en`. (Accessed May 17, 2021).

2. Project management and interdisciplinary works

This chapter gives an overview of some common issues, methods and best practices that are particularly relevant when developing software and when working on interdisciplinary projects. From the perspective of this dissertation, these considerations are made with the purpose of providing a helpful link between the humanities and the computer sciences field. Before transforming the ideas of the Remix Culture to technical solutions, a quick overview of how and when things can be done will provide additional value to this work. Hence, this chapter can be considered as a bridge that allows to look at the topics discussed in this work from the management level, and thus offer a holistic view of the whole panorama. As a matter of fact, the narrative will connect to the technical chapter about the project development in the section 2.2 *State of the art: web video-editing tools*. Certainly, the journey from the formulation of ideas and theories to their practical application is a complex and articulate topic. Since this dissertation is not focused on management, this will be merely an introduction about some of the techniques and interesting considerations, but their analysis can shed some light into the workflow that guides the production of software from a number of theoretical considerations.

The Remix Culture can be seen and analysed from a business perspective. Nowadays, it can be difficult to imagine a creation that is not somehow a combination of other things. This statement seems to be true also in the case of innovations. They can be defined as creations that introduce something new. "An innovation is something original and more effective and, as a consequence, new, that "breaks into" the market or society [...]"[1]. Moreover, innovations can be divided in two types: sustaining and disruptive innovations. The former category is based on incremental innovation, thus periodic improvements of existing products. For example, from its invention to these days a fridge came through a long process of improvements but the object itself can still be definable and it mainly serves as a fridge. On the other side a disruptive innovation "[...] is an innovation that

[1] *What is innovation?* URL: `https://www.pwarome.org/2019/03/18/what-is-innovation/`. (Accessed May 17, 2021).

helps create a new market and value network, and eventually goes on to disrupt an existing market and value network”[2]. This distinction is helpful to make an argument in favour of re-inventing instead of making disruptive innovations. Indeed, apart from some highly technical disciplines and fields that produce some significant technological advances, a large number of inventions are things that are re-invented, thus enhanced but not disruptively changed. They follow the cycle of incremental innovation rather than disruptive changes. Namely, two prominent examples can be considered as revolutionary but are to be framed as sustaining innovations.

- Ford Model T was the first mass-produced vehicle which made its first appearance in 1908[3].

Henry Ford democratized travel because he lowered the price of automobiles that became affordable to the general public. He simplified the production of cars which allowed for its industrialized production and in turn for a substantial internal cost reduction. Hence, a considerable price reduction for the final consumer. But cars already existed, they were expensive thereby destined to a wealthy audience. Therefore, it was not a disruptive innovation.

- IKEA optimized the large-scale process of distributing with the idea of flat-packed furniture[4].

Ingvar Kamprad, the founder of IKEA, optimized the large-scale process of distributing and selling the furniture with the idea of flat-packed furniture as “he realized that half the sale price of a table was in the cost of transporting it”. He decided to leave the effort on assembling smaller pieces together to the final customer. Providing instructions that everyone could understand. This led to unbeatable prices in comparison to the competitors in the furniture sector. From a practical point of view, since he reduced the transportation costs this also cannot be considered as a disruptive innovation.

[2]Clayton M. Christensen. *Disruptive Innovation*. URL: `https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/disruptive-innovation`. (Accessed May 17, 2021).

[3]Greg Lockwood Richard Koch. *Simplify*. Piatkus, 2016, pp. 3–10. ISBN: 9780349411859.

[4]Greg Lockwood Richard Koch. *Simplify*. Piatkus, 2016, pp. 11–21. ISBN: 9780349411859.

These examples demonstrate why the majority of business innovations belong to the category of sustaining innovations. Consequently, the Remix Culture concepts are relevant in such case but are probably slightly less pertinent in case of disruptive innovations, where the degree of originality is supposed to be greater. Nevertheless, the defining elements of remix, thus copying, transforming, and combining can be found in both these categories of innovations. This leads to the assumption that "everything is a remix" which by itself can be considered as misleading due to the broadness of the remix definition. Owen Gallagher is an active researcher in the remix field, he has published several book chapters, journal articles and conference papers on the topic. He advocates for a stricter definition of "remix" in his book "Reclaiming Critical Remix Video" stating that:

> "Every creative work is arguably inspired in some way by something else, but this does not mean that it is a remix. If every creative act is a remix, then nothing is a remix, and this is simply not the case."[5]

As said in the chapter 1 *Remix Culture*, this dissertation considers two high-level perspectives of the Remix Culture. The first is tied to the digital evolution and the second to a broader sociocultural concept. Hence, in this work both views are considered as equally important for characterising the whole concept.

Speaking of software, the software management is a topic that is worth mentioning before developing most projects. A good starting point is the book "The Mythical Man-Month: Essays on Software Engineering" written by Frederick P. Brooks Jr. The book is focused on software engineering and project management and describes the author's adventures when working at IBM as a project manager. In particular, it analyses a large project of the IBM – the IBM 360 computer family and then their operating system called OS/360 – that exceeded all the estimates in terms of costs and time delivery.

> "[...] the product was late, it took more memory than planned, the costs were several times the estimate, and it did not perform very well until several releases after the first."

[5]Owen Gallagher, Eduardo Navas, and xtine burrough xtine. *The Routledge Companion to Remix Studies*. Jan. 2015, p. 11. ISBN: 978-0415716253.

He compares the situation of a well-established company with a large availability of professionals to a scene from tar pits.

Figure 2.1: La Brea Tar Pits mural by C.R. Knight. Adapted as the cover illustration for the book "The Mythical Man-Month".

> “No scene from prehistory is quite so vivid as that of the mortal struggles of great beasts in the tar pits. In the mind’s eyes one sees dinosaurs, mammoths, and saber-toothed tigers struggling against the grip of the tar. The fiercer the struggle, the more entangling the tar, and no beast is so strong or so skilful but that he ultimately sinks.”[6]

Subsequently the author argues that many software teams have failed in meeting their goals, schedules, and budgets, therefore they have become entangled into the tar pits. It is a common scenario in all fields starting from the public government projects to private companies. The idea that something is getting slowly entrenched and completely unable to move or progress is how the author sees many software projects. This also led to the formulation of the famous Brook’s law which states that, “adding manpower to a late software project makes it later”. This leads to a conclusion that the work cannot be broken down into “man-months” or other simple units of effort. It seems that the core lesson to be learned is that poorly organised projects cannot be fixed using straightforward intuitive solutions like incrementing personnel because this does not tackle the root problems. Certainly, this highlights an important point in favour of the importance of good

[6]Frederick P. Brooks. *The Mythical Man-Month (Anniversary Ed.)* USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201835959.

management. Analogously, the scenes from tar pits could also be relevant when working on interdisciplinary projects. The latter are connected to another factor which increases the development complexities. Namely, the communication barriers. Founding a common language about some terms and concepts that are discipline-specific between people from different backgrounds and specializations and at the same time making everyone contribute to the development of the project may be regarded as the most important goal of any management and the imperative reason to ensure that a project succeeds. Thus, the importance of knowing and adopting the right methodology for managing projects is the very reason for which this is being mentioned in an interdisciplinary work.

Brooks revised his book ("The Mythical Man-Month") in 1995 and he added an assertion about the management techniques in software development. He stated that, "there will be no silver bullet [for managing complex software projects] within ten years". By looking at the state of the art of management, arguably he was right. On the other hand, things have improved into the field in the last years. Nowadays, the Agile Methods seems to be universally recognised as the foundation of good management for the majority of new products. "Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment"[7]. In short, the methodology consists in dividing work into small teams with cross-functional skills which develop the software in an iterative manner. It seems that almost every application that eventually will be given to the final users require certain organisational and mental approaches to be successful. This assertion could be further expanded by making a bold claim that every software project that eventually will be used by the large public is interdisciplinary.

The reason for making this claim lies in the peculiarity of the work done by software programmers. They possess a fundamental skill for their job, that is the Computational Thinking. Jeanette Wing defines Computational Thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent"[8]. This skill

[7] *Agile Alliance - What is Agile?* URL: `https://www.agilealliance.org/agile101/`. (Accessed May 17, 2021).

[8] Jeanette Wing. *Computational Thinking: What and Why?* URL: `https://www.cs.cmu.edu/`

comes with great responsibility as humans by nature are not logical nor rational. People are often guided by emotions, curiosity, and the cognitive principles of human thought processes. Therefore, it is imperative for the software solutions produced in a logical manner to be also user friendly. It could be argued that developing the best software, is an ability to provide User Interfaces that require the lowest energy consumption, thus the lowest effort from the user while still remaining effectively functional. For these reasons it seems that combining the Agile Methodology together with a Human-centered approach to development is the best solution available at the time of writing. Adopting a good methodology and techniques obviously does not guarantee success but largely increases the probabilities of it.

It is less and less a question of personal opinions on whether a Human-centered design is a valid approach. As a matter of fact, it is an ISO standard (ISO 9241-210:2019 Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems[9]). As defined in the standard, "Human-centred design is an approach to interactive systems development that aims to make systems usable and useful by focusing on the users, their needs and requirements, and by applying human factors/ergonomics, and usability knowledge and techniques."

There are various ways of adopting these kinds of iterative and human-centered approaches, Design Thinking by the Hasso Plattner Institute of Design at Stanford, and the Human Centered Design Process by IDEO[10] are among the widely adopted design methods for reaching the aforementioned goals.

~CompThink/resources/TheLinkWing.pdf. (Accessed May 17, 2021).

[9]*Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems.* URL: https://www.iso.org/standard/77520.html. (Accessed May 17, 2021).

[10]Cole Hoover. *Human-Centered Design vs. Design-Thinking: How They're Different and How to Use Them Together to Create Lasting Change.* URL: https://blog.movingworlds.org/human-centered-design-vs-design-thinking-how-theyre-different-and-how-to-use-them-together-to-create-lasting-change/. (Accessed May 17, 2021).

2.1 Open Source management

Open Source can be seen as a different way of organising work and can be analysed from the perspective of management. Notably, the mentioned example of the IBM management demonstrated how even large software organisations who are specialized in the field and do not lack resources and personnel fail in delivering their products. This prompts the question, how the success of large and technically complex Open Source projects like the Linux operating system could possibly be explained?

A positive response to this question might imply that the Open Source is a viable management approach. For that matter, software that works made by people who participate voluntarily and without some sort of hierarchical top-down management should probably fail standing to the analysis made in the last section. In case of Linux, this does not mean that the management was absent at all, instead it was more flexible and directed towards the inclusion, thereby making people feel part of the community rather than directly managing their efforts. Although every participant could contribute, it did not necessarily mean that all the contributions were automatically included into the system. Certainly, this workflow became more efficient with the system controls and versioning tools like Git[11]. It might be argued that the Open Source could be considered as a management revolution thanks to the Internet pervasiveness.

> "A global network lowers the costs of "commons collaboration" close to zero. Some people will participate to showcase their credentials, or to build a user-base for consulting services, or just because sharing is fun and costs nothing. Examples include Creative Commons [. . .] Wikipedia; Linux, an open-source operating system; and the massive amount of useful digital content created by volunteers."[12]

The economic incentives are at the base of a classical management model but are

[11] *Git - Version Control System*. URL: `https://git-scm.com/`. (Accessed May 17, 2021).

[12] James Boyle. *The Economyst - The fight to keep ideas open to all*. URL: `https://www.economist.com/open-future/2018/12/12/the-fight-to-keep-ideas-open-to-all`. (Accessed May 17, 2021).

not the only effective way of rewarding and encouraging people to work on projects. It appears that other motivations like feeling part of a community, passion for new technologies and the possibility of sharing universally accessible solutions can be an effective way of incentivising the collaboration in an Open Source environment. Another aspect worth considering regards the software developer's reputation. Commonly, participating into Open Source projects proves to be a valid working experience valued among many companies and software houses. By participating in this kind of projects developers can combine different motivations together. For example, it is possible to pursue the passion of programming while helping the community and at the same time improving one's own working career portfolio.

There is another facet of the Open Source world that was only slightly mentioned in last chapter, large corporations are also funding and paying their employees to work on Open Source projects[13]. The explanation for this phenomenon is simple since their own software makes extensive usage of the Open Source software. For instance, systems like Android, iOS etc. use thousands of open-sourced licensed solutions[14]. These solutions coexist and are mixed together, obviously respecting the licenses, to form products subsequently sold worldwide. Without the proper maintenance of these systems, their own proprietary solutions would collapse. Generally speaking, maintaining software is a delicate topic. Especially volunteering developers tend to be unwilling to do so, thus economic incentives might be needed. This is also the reason why Open Source is not "free". Indeed, it is connected with costs, like those necessary for its maintenance.

The final aspect regards the bureaucracy since many Open Source projects rely on donations. From a legal point of view the legislation imposes some sort of a formal structure in order to legally operate, make transparent financial reports, etc. Namely, these non-profit organisations are called foundations. When an Open Source projects grows, the necessity of creating a foundation becomes compelling. The Linux Foundation, Mozilla Foundation, Wikimedia Foundation are all examples of it. Bureaucracy and rules govern

[13]James Turner. *Open source has a funding problem.* URL: `https://stackoverflow.blog/2021/01/07/open-source-has-a-funding-problem/`. (Accessed May 17, 2021).

[14]For example, Apple provides a list of Open Source software used in their iOS based products. https://opensource.apple.com/

the work even if it is voluntary work. Every project that grows needs some sort of structure and eventually a legal organisation to ask for money, spend money and to demonstrate how it is spent. Certainly, other functions are also executed by these entities like enforcing community guidelines, but they are out of the scope of this argumentation.

Interestingly, large companies thrive thanks to the Open Source and frequently they contribute back whether it is for their own interest or for the one of the commons. Lawrence Lessig states that "[...] many businesses are now hybrids, with some elements that are open and some elements that are commercial"[15]. It is a scenario where private, and public coexist. It would be interesting to see this coexistence expanded to all the creativity works protected by IPR as mentioned in the first chapter of this work.

To summarize, in the past years the Open Source has proven to be a valid and sustainable management and business model although its adoption depends on the nature of the projects. Furthermore, not everyone may be interested in working in a flexible or almost structureless manner. The open world presents opportunities. From the software point of view the rationale behind making an Open Source video editor could be that there are not many existing in-browser solutions that allows for video-editing. Systems like YouTube, Vimeo are closed source and cannot be reused to build personal solutions. A contribution to Open Source can be made by building software that has not yet been offered in such a way to the community.

2.2 State of the art: web video-editing tools

Many solutions that allow to modify audio-visual content for "free" exist. Nevertheless, at the time of writing, Open Source web-based solutions are absent. As the difference between "free" and "open" should be clear by now, this section aims to list the most popular features offered by these kinds of tools before implementing some of them into the practical project development.

Before continuing with the state of the art analysis, any topic focused on management would probably be incomplete without mentioning the "Pareto principle", also referred

[15]L. Lessig. "Remix: Making Art and Commerce Thrive in the Hybrid Economy". In: 2008.

to as the “80/20” rule. The rule was made by the economist Vilfredo Pareto and it states that 80% of effects comes from 20% of the causes. It is also applicable to the software world and turns out to be particularly useful for targeting the final users for whom to develop the solutions. As a matter of fact, it is impossible to satisfy everyone’s needs and requirements. For example, if a person “A” asks for a certain feature “b” and a person “C” asks for another feature “d”, the manager should be able to provide the reason why one person’s idea gets accepted instead of the other one. Not doing so may lead to frictions and misunderstandings. Especially if the discarded feature intuitively made more sense. The reason for choosing the other one is justified by targeting the specific audience or the “20%”.

By keeping in mind the fact that users are all different it will be possible to enrich the state of the art analysis of video-editing tools and hopefully make better choices at a later stage of this project’s development. A variety of software and tools were considered in order to “copy” – speaking in the remix terms — ergo, get inspired and analyse the current tendencies of similar software. In particular, the focus was mainly posed on the web based solutions to better understand their current limitations. Notably the following applications were considered: Vimeo movie-maker[16], YouTube studio[17] (reserved for YouTube content creators), Canva Video[18] and Magisto[19]. An attempt to prioritise certain features was made by taking the point of view of a general user. In particular, “MoSCoW is a prioritisation technique for helping to understand and manage priorities”[20]. It consists of four categories, that is, “Must have”, “Should have”, “Could have” and “Won’t have”. While the first three are ordered by importance and are rather self-explanatory, the latter indicates the least important feature for the current development time constraint. It does not mean that it is undesirable but just that these features produce the lowest benefits for the project.

[16] *Vimeo - Video maker*. URL: `https://vimeo.com/create`. (Accessed May 17, 2021).

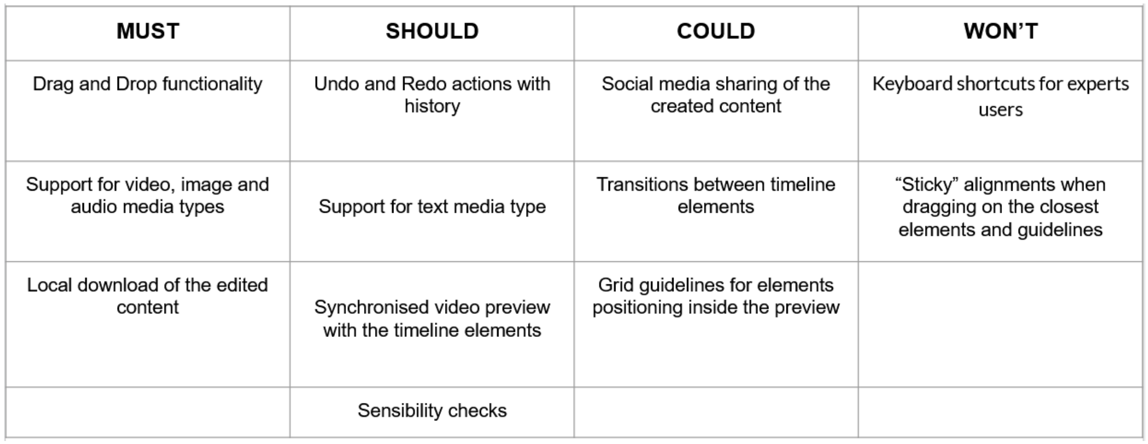
[17] *YouTube studio*. URL: `https://studio.youtube.com`. (Accessed May 17, 2021).

[18] *Canva - Video maker*. URL: `https://www.canva.com/create/videos/`. (Accessed May 17, 2021).

[19] *Magisto - Video Editor*. URL: `https://www.magisto.com/`. (Accessed May 17, 2021).

[20] *MoSCoW Prioritisation technique*. URL: `https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation`. (Accessed May 17, 2021).

The results reassuming some popular video-editing features can be seen in the figure below.



MUST	SHOULD	COULD	WON'T
Drag and Drop functionality	Undo and Redo actions with history	Social media sharing of the created content	Keyboard shortcuts for experts users
Support for video, image and audio media types	Support for text media type	Transitions between timeline elements	"Sticky" alignments when dragging on the closest elements and guidelines
Local download of the edited content	Synchronised video preview with the timeline elements	Grid guidelines for elements positioning inside the preview	
	Sensibility checks		

Figure 2.2: MoSCoW prioritisation of some common video-editing features

At this point a selected number of these functionalities that are of particular interest or may simply result unclear can be shortly explained.

- Drag and drop may be implemented for selecting and dragging media from the application library of content to the editing timeline. It should respect the Fitts' usability law[21], so the distance between the starting point and the target should be as short as possible while maintaining the right dimension of the target to reduce the risk of errors.

- Undo and Redo actions with history may be important for the User Experience as the editing may frequently be done in a trial and error manner. Without the possibility of going back and forth between the new and old modifications the whole editing process may result tedious for the user.

- Sensibility checks may be useful to prevent users from accidentally quitting the web page without saving their progress or in case it happens the current work could be saved in the cache memory.

[21] *Interaction Design Foundation - Fitts' Law*. URL: `https://www.interaction-design.org/literature/topics/fitts-law`. (Accessed May 17, 2021).

Another point of interest would be in making an application with no barriers for entry, for example without the necessity of creating an account. Looking at the state of the art all the considered tools included a certain form of lock-out. Namely, the need for registering an account and authenticating the user. Some ideas that emerged from this analysis were kept in mind when developing the next chapter's solution. This dissertation was concerned with making a working prototype, so it might be argued that these analyses were not as impactful as they are supposed to be when making final solutions. Indeed, the scope of this activity strongly depends on the study of the final users of the application.

3. Project development: web application for multimedia editing

This chapter is meant to illustrate the main thesis about the Remix Culture by combining most of the topics presented in the previous parts into a practical example of project development. The project consists in developing a web application for multimedia editing that can run in every modern browser. Hence, from a general point of view this can be seen as an implementation of a set of features that are common in many offline and online tools for video-editing, as analysed in section 2.2 *State of the art: web video-editing tools*. In more detail, the application main value proposition is to offer the possibility of "remixing" different types of multimedia content, in particular videos, images, and sound media. Thanks to this universally accessible and Open Source technical solution the Remix Culture can come to life and can be experienced by anyone with an internet connection.

More details about the requirements, technical decisions and considerations can be found in the paragraph 3.4 *Project idea and Front End Software Architecture*. At this point, the main argument is addressed from the perspective of software development. Specifically, the results presented in this this chapter follow a two-pronged objective.

Firstly, the development of an application for remixing multimedia enables users to experience the concepts of the aforementioned Remix Culture by enhancing content discoverability and stimulating user's creativity. On the other side, content holders, i.e., authors, institutions, etc. can gain in recognition, thus receive added value from this kind of application. Therefore, this can be seen as an attempt to promote the Remix Culture among institutions which may be encouraged by the benefits and decide to upload content on the platform. A particularly suitable target might consist of cultural heritage organisations with their frequent digitalization programmes. These institutions' digitization strategies often set their priority goals for enhancing user engagement and participation.

A second objective comes from the choices made for the technical architecture. The video-editing tool is open for modifications in terms of software development. By publishing and applying the “MIT” license[1] to the source code, a contribution to the Open Source is made. Furthermore, by adopting a component driven development with a solution based on Web Components technology – a standard and framework agnostic solution as explained in paragraph 3.2 *Web Components* – the technical solutions are offered to the community as a library from which to choose from, expand upon, and create brand new projects.

In the next paragraph, 3.1 *Reusability and maintainability, through abstractions to a component-based approach*, an analysis of the modern software development is presented together with arguments in favour and against the adoption of a component-based approach.

Finally, at the end of this chapter, two examples of application instances will be made: the first example was developed in the context of the *PH-Remix* project. In this case, as mentioned in the introduction, the content belongs to the “Mediateca Toscana”[2] and the development was guided by the requirements expressed in the regional project grant. The project involves several researchers, and the development is still in progress as the project deadline is set to 2022. Therefore, a front-end component (or set of components) was implemented, in anticipation of the back end architecture, while work on the latter is still in progress at the time of writing. The second application integrates a variety of Open Source materials. It is an example of reusability of the components architecture that can be used for different purposes. The implementation is based on the public Pexels API[3] which offers a large number of videos and images. All the content can be used thanks to a permissive license. This solution acts as an aggregator of Open Source licensed materials to create new general purpose productions. It is not connected to any specific target audience. An example could be the discovery of content to be used as commercial advertisement for any kind of industrial product.

[1] *The MIT License*. URL: `https://opensource.org/licenses/MIT`. (Accessed May 17, 2021).

[2] *Mediateca Toscana*. URL: `https://www.mediatecatoscana.it/`. (Accessed May 17, 2021).

[3] *Pexels - Photos and Videos*. URL: `https://www.pexels.com/`. (Accessed May 17, 2021).

3.1 Reusability and maintainability, through abstractions to a component-based approach

Modern software development techniques and tendencies are mainly concerned with the objective of simplifying the development of technical solutions, as well as easing up code reusability and maintainability.

Nowadays, programming – the activity of writing a code – involves the use of a growing number of abstractions as a way of responding to the challenges deriving from the complexity of modern software applications. R.C. Martin in the introduction of Clean Code[4] affirms that:

> "[...] the level of abstraction of our languages will continue to increase. I also expect that the number of domain-specific languages will continue to grow. This will be a good thing. But it will not eliminate code [...]".

As the author confutes the "no-code development approach", he also states that the code will be evolving as an expression of the software specifications. These specifications or requirements need to be translated into a form that is understandable and executable by a machine. Hence, the concept of abstractions will be analysed as one of the guiding principles of software evolution.

Focus will be put on the front-end development ecosystem, as it reflects the majority of work done for this dissertation practical application. Despite this choice, most of the arguments presented apply to the software world in general. The final aim of this paragraph is to provide the rationale behind the choice of Web Components as the technology for project development. As a matter of fact, before introducing Web Components, to fully understand the component-based approach, an introspection into the web development stack ought to be made.

The front-end development is commonly considered as a fast-paced environment among

[4]Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st ed. USA: Prentice Hall PTR, 2008. ISBN: 0132350882.

developers[5]. In short, the lifecycle of the technologies used for the web application layer is shorter compared to the rest of the software world. Consequently, programmers usually need to make a significant effort in order to stay up-to-date on the recent technological progress. In other words, front-end frameworks and libraries tend to become less maintained, and thus deprecated in a short span of time.

Although the fundamental programming skills required for creating web applications – ranging from simple static websites to complex online platforms – are still based on HTML, CSS, and JavaScript, several solutions were created for more efficient and effective way of reaching programming goals. A relevant example of this evolution is provided by JavaScript frameworks and libraries. It is worth emphasizing that JavaScript started off as a scripting language, generally used for small tasks like adding interactivity to the web pages. As of today, it has become a mature and widely used language both on the front-end and the back-end side (mainly thanks to the development of V8 JavaScript engine used for runtimes environments of Node.js, Deno etc.).

On the front-end side, the progress might be viewed as a development of proprietary and Open Source systems that implement features with the goal of easing up and speeding up the development of web projects. Starting from the jQuery library in 2006, which offered a cross-browser compatible set of features with an API that simplified the most frequent actions like DOM traversal, event handling etc., to the first frameworks (Backbone, Ember, Angular just to name a few) with more sophisticated features like data binding, dependency injection, routing and many more. Another popular, more recent example is the React library.

One of the core innovations of the React library is its design principle of "composition of components". This principle consists in building "encapsulated components that manage their own state, then compose them to make complex UIs"[6]. This innovation, as seen

[5]It is worth noting how this statement can be sustained by real world data. Stack Overflow makes yearly surveys among its community of developers. The Stack Overflow Annual Developer Survey (https://insights.stackoverflow.com/survey/) contains a section called "trending techs" where the rise and decrease in popularity of web development technologies over the years can be analysed in further detail

[6]*React JS - Design Principles.* URL: `https://reactjs.org/docs/design-principles.html`. (Accessed May 17, 2021).

in React, reflects the tendency of many other modern front-end frameworks, such as: Vue, Angular, Svelte, etc. They are based on the idea of splitting codebases into small reusable pieces of code, called components. This approach is particularly useful for medium and large software projects as it enables modularization, by splitting software logic into separate modules, thereby encouraging reusability and maintainability. Another benefit of this approach is that the development process can also be better organised between multiple teams working on different features.

As more and more solutions were introduced into the web development ecosystem, a phenomenon commonly called "proliferation of frameworks"[7] became an issue. An issue that might have consequences beyond the vastity of the offer to choose from. An immediate consequence of using frameworks is the framework-specific way of developing code or components. The developers are required to learn a new framework and dedicate to it in order to gain the benefits that are being offered by those solutions. A flaw of this approach logically follows as the development becomes limited to the specific framework or technology. It seems that more arguments on the downside of using frameworks can be made. The possible common issues involve their lifespan and changes due to new releases that may lead to issues with compatibility between the old and the new versions. Also, the size of the framework codebase, i.e., the functions installed and required by default – often not necessary in every project – can slow down the application performances. For some projects this might be a viable solution, as for companies currently developing solutions that might benefit from specific features and differences offered by frameworks or for those who can avail of having an internal expertise to justify the usage of a specific framework.

Many of these new features offered by the above tools are connected to the standardization process itself. It seems that the creation of new solutions that gain popularity within the community stimulate the implementation of these features into web standards. Notably, in this context, W3C recommendations standards[8] and JavaScript ECMAScript

[7]Kevin Ball. *The increasing nature of frontend complexity*. URL: `https://blog.logrocket.com/the-increasing-nature-of-frontend-complexity-b73c784c09ae/`. (Accessed May 17, 2021).

[8]*W3C - Standards and Drafts*. URL: `https://www.w3.org/TR/?status=rec`. (Accessed May 17, 2021).

specification[9]. Arguments in favour of this thesis can be made. For example, the standard Selectors API specification[10], offering "querySelector" "querySelectorAll" interfaces for finding elements inside the DOM tree structure, was inspired by the jQuery selectors helper methods[11]. From the perspective of frameworks or libraries adopters the benefits of adopting the libraries over the native standard – which is faster – are inversely proportional to the number of features that exist in the standard. Although nothing prevents developers from adopting jQuery and similar solutions, their adoption may make less and less sense leading to their natural decline.

This prompts the question, if the component approach seems to be a shared direction for developing more complex front-end systems, how to tackle the issues regarding supposedly one of the main features of components, namely the benefit of writing a reusable code while being locked-in into specific technology stacks?

A possible answer might be in the Web Components technology which enforces the property of reuse among applications. In short, having a standardized way of building components would mean greater reusability, configurability, and consistency – avoiding technical lock-ins – for many years to come. As a general rule of thumb, Web Standards, made by W3C and similar, are more reliable than the proprietary Long Time Support ("LTS") solutions. Web components are a set of Web APIs that allow to create custom, reusable HTML elements with logic and styling encapsulated into their definition. They can tackle the problems of developing interoperable components. By inserting the components within the frameworks, they are expected to work in a plug and play manner thanks to the native APIs compatibility[12].

Nevertheless, Web Components are not to be considered a silver bullet for solving issues presented in this section. Currently, it appears that some problems could also be

[9] *ECMAScript® 2020 language specification.* URL: `https://www.ecma-international.org/publications-and-standards/standards/ecma-262/`. (Accessed May 17, 2021).

[10] *W3C - Selectors API.* URL: `https://www.w3.org/TR/2020/SPSD-selectors-api-20201103/`. (Accessed May 17, 2021).

[11] *jQuery - Selectors API.* URL: `https://api.jquery.com/category/selectors/`. (Accessed May 17, 2021).

[12] Rob Dodson. *Custom Elements Everywhere.* URL: `https://custom-elements-everywhere.com/`. (Accessed May 17, 2021).

mitigated by adopting a set of techniques and strategies such as “Micro-Frontends”[13]. The latter combined with “Webpack Module Federation” seems a particularly promising solution. A detailed analysis is beyond the scope of this dissertation. However, for a better perspective of the current state of art, the idea of Micro-Frontends could be summarized as the application of microservices architecture to the front-end world. As it happens with the back-end, the aim is to decouple the single pieces of an application in a way that they can be developed independently and using different technologies. From a business point of view, this would make sense in large projects and is a promising approach to Agile methodology as small teams could work on developing autonomous features. The key difference between the Web Components approach and the Micro-Frontends is that the development of interoperable components can be done by using different frameworks. A quick example could be a coherent application composed by components made with the Vue, React and Svelte frameworks. Certainly, this leads to complexities. Therefore, the Webpack Module Federation was introduced. This Webpack feature allows a JavaScript application to dynamically load code from another application without compromising security[14].

In this panorama, the Web Components technology seems like an additional asset to the current way of developing front-end projects. Its aim does not involve replacing the actual front-end frameworks, it should rather be considered as one of the possible ways for adopting a component-based approach on the web.

Returning to the *PH-Remix* case study, the Web Components were adopted as a front-end technology in line with the project architecture. As a matter of fact, the project architecture is based on a microservice model that allows for a degree of flexibility and enables a parallel development of the back-end infrastructure. Also, from the project development point of view, relying on the standard allows for future developers to modify and enrich the code more easily. Consequently, the necessity of learning new frameworks is absent. This suggests that the features that Web Components offer could solve some of these problems.

[13]*Micro-Frontends*. URL: `https://micro-frontends.org/`. (Accessed May 17, 2021).

[14]Zack Jackson. *Webpack 5 Module Federation*. URL: `https://medium.com/swlh/webpack-5-module-federation-a-game-changer-to-javascript-architecture-bcdd30e02669`. (Accessed May 17, 2021).

Indeed, the main promise of making a flexible architecture seemed to be easily solved thanks to this front-end technology of choice. In the next paragraph the main standard's features will be introduced.

3.2 Web Components

To reiterate the definition from the last paragraph, Web components are a set of Web APIs that allow to create custom, reusable HTML elements with logic and styling encapsulated into their definition. From a practical point of view, modern browsers can compute Web Components in the form of custom HTML tags that, in turn, can be used and re-used across multiple projects and technology stacks. The described behaviour is equivalent to the one of standard HTML elements as defined in the HTML documentation.

Web Components offer a native platform and a technology agnostic component model for developing a code that runs in browsers. As a matter of fact, this follows the tendency of many modern front-end frameworks, which are based on the idea of splitting codebases into components, as mentioned in this chapter's introduction. The main difference is the interoperability provided by adopting the solution introduced in this chapter.

Web Components technology can be considered as an umbrella term that encompasses three distinct Web APIs: Custom Elements, Shadow DOM, and HTML Templates. By defining and referring to Web Components as a native technology, the standardization by W3C is intended. The specifications for each of these Web APIs are now incorporated as a living standard[15]. Each of these specifications will be explored with more detail in the next sections to explain their features and the current limits of this technology. The purpose of these explanations is to provide some basic knowledge about the internal mechanisms and a few points of interest relevant to the subject that might be of interest for readers.

The formal documentations and technical definitions hardly give the developer's perspective and community feelings about this topic. Therefore, another aspect worth investigating regards the evolution and the adoption of Web Components.

[15] *W3C - Evergreen Standards*. URL: `https://www.w3.org/wiki/Evergreen_Standards`. (Accessed May 17, 2021).

The proposal of Web Components is not new, speaking in terms of software development dynamics. Firstly introduced in 2011 by Alex Russell[16], it went through years of standardization process. The latter might be also viewed as search for consensus between browsers producers, namely the owners of: Safari, Firefox, Chrome – therefore, all the Chromium based browsers – and Microsoft browsers. While the Microsoft browsers consist of Internet Explorer and Microsoft Edge, the former has reached the end of its lifecycle maintaining only the technical support and security updates for its latest version (Internet Explorer 11). The browser does not support the Web Components standard[17], therefore its implementation and usage require the adoption of polyfills and transpilers. Generally, transpilers are used to compile the source code written in one language to another. Although in this case their main usage is to transform the code written using some – usually recent – standard into a backward compatible code that can work with the old browsers not supporting that standard. One example of the most popular transpiler is Babel[18], which allows developers to use the most recent JavaScript features among all the browsers by translating them into a code compliant with the old language standards.

Returning to the issues of browser compatibility, the deprecation of Internet Explorer and the subsequent evolution of its successor, Microsoft Edge, gave Web Components a boost in terms of compatibility. The main reason can be linked back to the Microsoft Edge passage to the Chromium Open Source project on 8th April 2019[19]. This fundamental change consisted in a passage from the combination of EdgeHTML browser engine and Chakra JavaScript engine to Blink and V8, respectively a browser engine and a JavaScript engine.

This led to a substantial improvement and larger adoption by the community. How-

[16]Alex Russell. *Web Components and Model Driven Views*. URL: `https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell`. (Accessed May 17, 2021).

[17]*Can I use... - Web Components*. URL: `https://caniuse.com/?search=components`. (Accessed May 17, 2021).

[18]*Babel - JavaScript compiler*. URL: `https://babeljs.io/`. (Accessed May 17, 2021).

[19]Microsoft Edge Team. *What to expect in the new Microsoft Edge Insider Channels*. URL: `https://blogs.windows.com/msedgedev/2019/04/08/microsoft-edge-preview-channel-details/`. (Accessed May 17, 2021).

ever, this overview draws attention to the standardization process. As seen, almost ten years were necessary until one of the last major browsers adopted the initial proposal. In the meantime, the standard itself came through a long process of maturation and stabilisation[20].

The longevity of the process exemplified above might imply and justify the existence of critiques against the Web Components standard. By investigating the most common reasons, it appears that they were mainly concerned with cross-browser discrepancies, partial standard adoption, and frequent API changes. Serhii Kulykov, a developer currently working at Vaadin – a company which uses Web Components with an Open Sourced web application platform – critically reviews in his series of blogs[21] [22] about Web Components and gives an insight view regarding most of the problems previously mentioned.

To summarize, Web Components became standardised and supported by all the major browsers only recently. This suggest that their popularity and adoption rate could lead to a more substantial increment within the web technology market share in the future. Consequently, attracting more developers, who develop more tools leading to possible improvements of the standard itself, as argued above when talking about the connection of standardization and external solutions development. Ultimately this could lead to a substantial front-end innovation.

Looking at the state of art adopters, ING[23], AXA[24] and IBM[25] are three examples of

[20] An interesting lecture by Jan Miksovsky on how the standardization of HTML slot element may give a better view on the complexities of standardization process: https://component.kitchen/blog/posts/a-history-of-the-html-slot-element

[21] Serhii Kulykov. *Beyond the polyfills: how Web Components affect us today?* URL: `https://dev.to/webpadawan/beyond-the-polyfills-how-web-components-affect-us-today-3j0a`. (Accessed May 17, 2021).

[22] Serhii Kulykov. *The journey of Web Components: wrong ways, lacking parts and promising paths.* URL: `https://dev.to/webpadawan/the-journey-of-web-components-wrong-ways-lacking-parts-and-promising-paths-1d5a`. (Accessed May 17, 2021).

[23] *ING - Lion Components.* URL: `https://github.com/ing-bank/lion`. (Accessed May 17, 2021).

[24] *AXA - Components Design System.* URL: `https://design.axa.com/web-guidelines/design-system`. (Accessed May 17, 2021).

[25] *IBM - Carbon Design System.* URL: `https://github.com/carbon-design-system/carbon-web-components`. (Accessed May 17, 2021).

companies that implement and Open Source their design systems based on Web Components. Various other components implementations are also present in some of the biggest websites. For example, just to name a few: GitHub, YouTube, Google Earth are among the adopters of Web Components. Especially, Google with its Web Components library, called LitElement, seems committed to the technology. In paragraph 3.3 *Web Components libraries* a more extensive analysis of LitElement will be presented.

As seen up to this point, the overview of state of the art highlights some of the challenges and opportunities for future Web Components development. Before delving into an overview of the WebAPIs technicalities, an introduction to JavaScript Modules will be presented in the following paragraph.

3.2.1 JavaScript Modules

JavaScript modules (also known as "ES Modules" or "ECMAScript Modules") are core to the modern web development and can be considered as a prerequisite for the Web Components technology. The definition of a module seems to be quite consistent across the subject literature. A module can be defined as: "[...] a piece of program that specifies which other pieces it relies on and which functionality it provides for other modules to use (its interface)"[26].

From a general point of view, modules act as a glue connecting components or code dependencies and therefore allowing for code splitting and composition. This implies the advantage of writing and maintaining a code structure that can be both understandable and maintainable. This can be particularly beneficial for developers (usually people who did not create the original code). Most likely, without modularization, code would be destined to become exponentially complex to organise in relation to the growth of project features. As written in the "Modules" chapter of "Eloquent JavaScript" book:

> "The phrase "big ball of mud" is often used for such large, structureless programs. Everything sticks together, and when you try to pick out a piece, the

[26]Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd. USA: No Starch Press, 2018, pp. 173–184. ISBN: 1593279507.

whole thing comes apart, and your hands get dirty."[27]

This quote resembles the "Tar Pits" metaphor as seen in chapter 2 *Project Management and interdisciplinary works*, when talking about some of the most common causes leading to failures of software projects from the managerial perspective. Here, the author makes arguments in favour of spending additional time for planning and coding structuring phases. A relevant example might come from the experiences and feelings of many developers who worked or are working with structureless or badly structured code. It may often appear that rewriting code from scratch could be easier than trying to extract it from its original context. By joining these two evocative concepts, "big ball of mud" and "tar pits", a step forward in the direction of project quality improvement can be made. This would mean recognising and preventing the root causes leading to the negative symptoms described above. A possible solution may come from an appropriate combination of good programming practices and management skills.

In 2015 JavaScript introduced its own standardized module system, called ES Modules[28] with its own import-export syntax. Since 2018 the JavaScript Modules standard is supported by all the major browsers natively[29]. Nowadays, this is standard but initially the Web Components technology used to have a fourth element, HTML imports[30]. The latter was intended as a packaging mechanism for including HTML documents inside other HTML documents. However, the standardization attempt failed since the browser vendors decided not to support it.

A simple example of ES Modules usage logic can be seen in the next code snippet. Assuming the following file structure as proposed in Code 3.1.

```
index.html
```

[27]Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd. USA: No Starch Press, 2018, pp. 173–184. ISBN: 1593279507.

[28]*Mozilla Developer Network - JS Modules*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules. (Accessed May 17, 2021).

[29]*Can I use... - ES6 Modules*. URL: https://caniuse.com/es6-module. (Accessed May 17, 2021).

[30]*Mozilla Developer Network - HTML Imports*. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components/HTML_Imports. (Accessed May 17, 2021).

```
main.js
modules/ // directory
    utils.js
```

Code 3.1: File structure with a module

In this example one of the functions defined in the file “utils.js” is used inside “main.js” script. This is done by exporting functionality from the module and importing it inside the script thanks to “import” and “export” statements:

```
// main.js
import { createAndAppendParagraph } from "./modules/utils.js";

const element = createAndAppendParagraph(document.body, "article-
    segment");

// utils.js

const MEDIA_TYPES = {
    VIDEO: "video",
    AUDIO: "audio",
    IMAGE: "image"
}

function createAndAppendParagraph(parent, className) {
    let paragraphElem = document.createElement("p");
    paragraphElem.classList.add("${className}");
    parent.appendChild(paragraphElem);
}

export { MEDIA_TYPES, createAndAppendParagraph }
```

Code 3.2: Import, export directives

The code above exports two types of variables, a constant “MEDIA_TYPES” and a function “createAndAppendParagraph”. In this way any JavaScript construct can be exported if it is not a top-level item, that is: an item not nested inside objects or functions etc. The main script imports one of the exported functionalities, namely the "createAndAppendParagraph" function. The latter is called within "main.js" and appends a DOM HTML paragraph to the parent (passed through the "parent" parameter") with the class as specified in the "className" parameter. By analogy, the constant "MEDIA_TYPES" could be imported by an indefinite number of JavaScript scripts. For example, since JavaScript does not implement an "enum" data type it could be used for the purpose of simulating immutable data through code. In this perspective, modules are crucial for Web Components. Starting from how custom elements are instantiated into an HTML document via the import directive to the inner dependencies between components. Modules allow to avoid conflicts in the global JavaScript namespace, where components could interfere with one another (a concept that will be further extended in section 3.2.3 *Shadow DOM*).

By expanding upon the idea of reusability, Web Components should enable developers to pick and use components created from anyone and to incorporate them into their own projects. Indeed, components in general should work like LEGO blocks. Since a Web Component may rely on other Components to work, the modules system provides the mechanism to connect or "glue" everything together in an organised manner. This resolves the "big ball of mud" software problem.

Finally, it might be argued that modularization is crucial for most good practices of software development. Principles like those introduced by Robert C. Martin, commonly called: "SOLID Design Principles"[31] but also other principles with rather fascinating acronyms like: DRY[32], KISS[33] and YAGNI[34] can be thought as a developer's Swiss Army knife (as a positive metaphor for usefulness and adaptability) for solving present and fu-

[31] *Wikipedia - Solid (principles).* URL: `https://en.wikipedia.org/wiki/SOLID`. (Accessed May 17, 2021).

[32] *Wikipedia - DRY (principle).* URL: `https://en.wikipedia.org/wiki/Don%27t_repeat_yourself`. (Accessed May 17, 2021).

[33] *Wikipedia - KISS (principle).* URL: `https://en.wikipedia.org/wiki/KISS_principle`. (Accessed May 17, 2021).

[34] *Wikipedia - YAGNI (principle).* URL: `https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it`. (Accessed May 17, 2021).

ture software problems. From the web developer's perspective this may suggests a more efficient and productive way of working. Continuing this tendency of beneficial features in the next paragraphs Web Components APIs will be presented.

3.2.2 Custom Elements

Custom Elements, as defined in the W3C living standard[35], can be introduced as: "A set of APIs that allow you to define custom elements and their behaviour, which can then be used as desired in your user interface"[36].

From a practical perspective, Custom Elements enable to programmatically extend the base HTMLElement[37] (an interface that represents any HTML element) via a Custom Registry API[38]. Hence, calling the "CustomElementRegistry" method "define" allows to define a new custom element as demonstrated in the Code 3.3 *Custom Element definition*. This turns out to be a powerful combination for enriching the HTML native vocabulary of elements with custom tags containing encapsulated functionality.

```
class VideoEditor extends HTMLElement {
    constructor() {
        super();
        // Some element functionality here
        ...
    }
}
window.customElements.define("video-editor", VideoEditor);
```

Code 3.3: Custom Element definition

[35] *W3C - Custom Elements*. URL: https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements. (Accessed May 17, 2021).

[36] *Mozilla Developer Network - Web Components*. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components. (Accessed May 17, 2021).

[37] *Mozilla Developer Network - HTMLElement*. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement. (Accessed May 17, 2021).

[38] *Mozilla Developer Network - CustomElementRegistry*. URL: https://developer.mozilla.org/en-US/docs/Web/API/CustomElementRegistry. (Accessed May 17, 2021).

Custom Elements are created by using the JavaScript class syntax which enables inheritance through the "*prototypal chain*" from HTMLElement interface. In the above example, the code in line 8 registers a Custom Element named "video-editor" (a dash is required by the syntax) whose definition is contained in the class "VideoEditor", that in turn defines the element's behaviour. From now on, the "video-editor" Custom Element can be used just as any other native HTML tag.

```
<video-editor></video-editor>
```

Code 3.4: Custom Element usage

As mentioned, the class syntax is used to extend the native HTMLElement interface through inheritance. Figure 3.1 *HTML Element Interface* depicts the native hierarchy of Browser APIs.

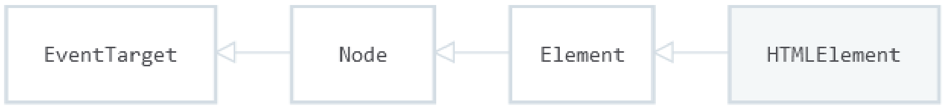


Figure 3.1: HTML Element Interface (from Mozilla Developers Network, licensed under CC-BY-SA 2.5)

In this context, the result of registering the custom element could be viewed as in Figure 3.2 *DOM Inheritance*.

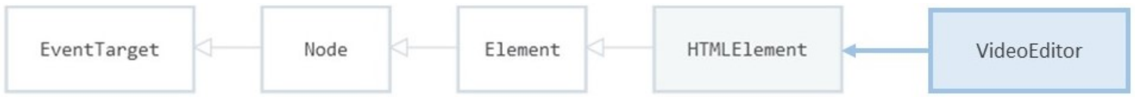


Figure 3.2: DOM Inheritance

In this way a new *"Autonomous Custom Element"* was created. As seen in the examples, these are elements that do not directly inherit from other elements in the HTML vocabulary (apart from the HTMLElement root). As a matter of fact, the API allows to create two types of custom elements. The second type is called *"Customized built-in element"*, meaning that the custom element extends a built-in HTML element. For example a HTML ”p” tag, standing for a paragraph element or the HTML “img” tag standing for image element, and so on. At the time of writing, the Customized built-in elements have a slightly more limited browser support[39]. Furthermore, in this work only Autonomous Custom Elements were used.

Another important feature of Custom Elements are lifecycle hooks. These hooks – or more precisely, callback functions – can be defined in the class constructor and are automatically executed when elements enter one of the following states[40]:

- connectedCallback: when the element is appended for the first time into a document’s DOM.
- disconnectedCallback: when the element is removed from a document’s DOM.
- adoptedCallback: when the element’s is moved to a new document.
- attributeChangedCallback: when the element’s attributes are added, removed, or changed.

A frequent usage example can be made by considering Custom Element initialization and the update phase. The former triggers a connectedCallback method. Within this method, an initial state of the Component can be set: namely, API calls to populate the User Interface, registration of event handlers etc. Later an update phase can occur when one or more Component attributes change thanks to the attributeChangedCallback method. This is the case of reacting to user actions such as form actions, resolution changes, various interactions on interactive elements and so on. These changes can be automatically captured and reflected dynamically thanks to Component attributes. For example, if users interact with a server, the application state could change. A common scenario is generating a loading state. While the application is waiting for the server response, it can generate the appropriate feedback (a spinner element, helpful text messages etc.).

[39] *Can I use... - Custom Elements.* URL: `https://caniuse.com/custom-elementsv1`. (Accessed May 17, 2021).

[40] Eric Bidelman. *Custom Elements v1: Reusable Web Components.* URL: `https://developers.google.com/web/fundamentals/web-components/customelements#reactions`. (Accessed May 17, 2021).

3.2.3 Shadow DOM

Shadow DOM API as referred in the standard, can be defined as:

> "A set of JavaScript APIs for attaching an encapsulated "shadow DOM" tree to an element — which is rendered separately from the main DOM document — and controlling associated functionality"[41].

In other words, it enables to attach a separate, encapsulated DOM which is isolated from the rest of the page. This is particularly useful for Custom Elements. It implies that the functionality defined inside a component, i.e., styles, markups and behaviour is limited (encapsulated) within the Shadow DOM boundaries. In other terms, anything happening inside a Shadow DOM does not have any effect outside of it.

The Shadow DOM API method, "attachShadow"[42] is used for attaching a Shadow DOM to an HTML Element (therefore any[43] autonomous custom elements defined in the previous section).

The consequences can be seen with an example. Given two components: "Component A" and "Component B", supposedly both containing an HTML input element with an attribute, "id" set to value "input-1". Standing to the HTML best practices "ids" should be unique in the entire document. This is not the case of this example as "Component A" DOM is fully separated from "Component B" DOM. Therefore, elements referred by the id "input-1" can be styled and referenced separately for the two components even if they were instantiated inside the same application.

Code 3.5 shows the shadow DOM attachment to a standard HTML Element and a Custom Element.

[41] *WHATWG - Shadow Trees*. URL: `https://dom.spec.whatwg.org/#shadow-trees`. (Accessed May 17, 2021).

[42] *Mozilla Developer Network - Element.attachShadow*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Element/attachShadow`. (Accessed May 17, 2021).

[43] Shadow DOM cannot be attached to every HTML element, a list of HTML element supporting this functionality can be found in the documentation.

```javascript
// Shadow Root attachment to HTML <article> element
const htmlInput = document.createElement("article")
        .attachShadow({ mode: "open" });

// Shadow Root attachment to Custom Element
class VideoEditor extends HTMLElement {
    constructor() {
        super();
        // Some element functionality here
        ...
        this.attachShadow({ mode: "close" });
    }
}
```

Code 3.5: Shadow DOM attachment

The Shadow DOM API allows to attach two types of Shadow DOM. An open shadow root allows accessing the DOM outside of the HTML element definition while a closed shadow root does not expose the DOM outside of the element. In the latter case the DOM cannot be referenced and accessed outside of its context. In theory, it exists in the flow of the document since it is rendered but its functionality is hidden from the rest of the page. However multiple ways of "hacking through" the limitations of closed shadow root exist[44], therefore the usage of open shadow DOMs is recommended.

Finally, an overview to summarize the concepts introduced can be seen in Figure 3.3: *DOM Tree and Shadow Tree*.

[44]Leon Revill. *Open vs. Closed Shadow DOM*. URL: `https://blog.revillweb.com/open-vs-closed-shadow-dom-9f3d7427d1af`. (Accessed May 17, 2021).

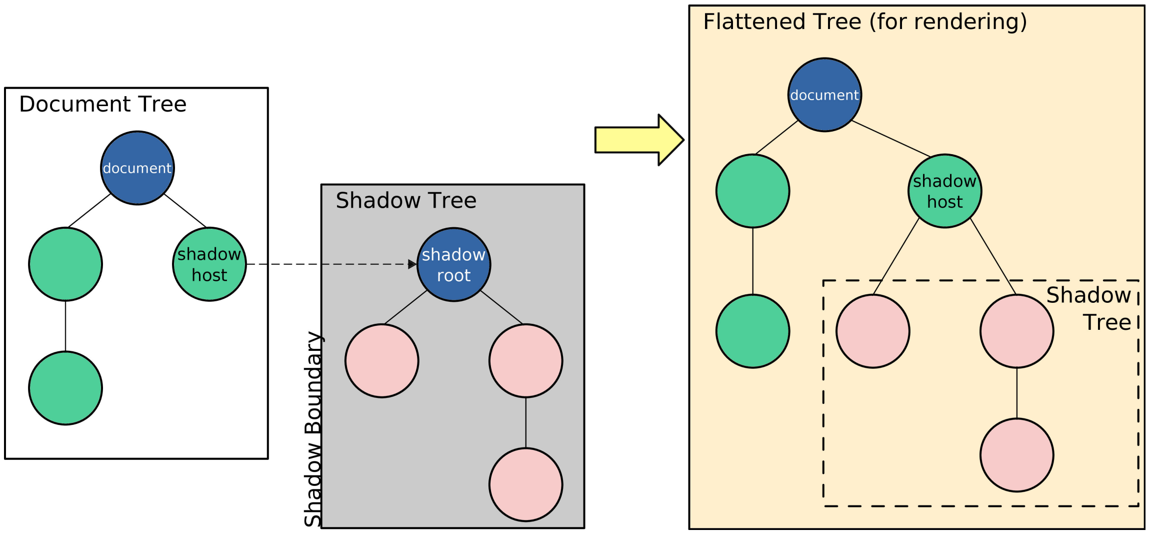


Figure 3.3: DOM Tree and Shadow Tree (from Mozilla Developers Network, licensed under CC-BY-SA 2.5)

Some basic terminology is also explained in the documentation[45]:

- Shadow host: The regular DOM node that the shadow DOM is attached to.
- Shadow tree: The DOM tree inside the shadow DOM.
- Shadow root: The root node of the shadow tree.
- Shadow boundary: the place where the shadow DOM ends, and the regular DOM begins.

3.2.4 HTML Templates

The HTML Templates technology consists of HTML template[46] and HTML slot[47] elements, specified by the W3C living standards. Among the main characteristics, they allow

[45] *Mozilla Developer Network - Using shadow DOM*. URL: `https://developer.mozilla.org/en-US/docs/Web/Events/Creating_and_triggering_events`. (Accessed May 17, 2021).

[46] *WHATWG HTML Standard - Template*. URL: `https://html.spec.whatwg.org/multipage/scripting.html#the-template-element`. (Accessed May 17, 2021).

[47] *WHATWG HTML Standard - Slot*. URL: `https://html.spec.whatwg.org/multipage/scripting.html#the-slot-element`. (Accessed May 17, 2021).

to produce reusable content that can be defined, referenced by JavaScript, and later instantiated inside the DOM Document. This can be useful for creating content with redundant structure only once.

The Template element "is a mechanism for holding HTML that is not to be rendered immediately when a page is loaded but may be instantiated subsequently during runtime using JavaScript"[48]. This definition suggests that HTML structures of arbitrary complexity can be created and dynamically appended to the DOM Document.

```html
<template class="image-gallery">
    <ul class="gallery-list">
        <img src="firstImage.png" alt="first image description">
        <img src="secondImage.png" alt="second image description">
    </ul>
</template>
```

Code 3.6: HTML Template creation

Notably, this approach can be used with and within Web Components to produce templates of components. Therefore, components can be put inside templates and instantiated when and where necessary. While templates are a more general purpose mechanism, the slot element was introduced specifically for the Web Components technology context. Thanks to the "slot" tag, developers can create placeholders inside Web Components that allow to instantiate additional children with their own separate DOM trees. Compared to templates, slots offer greater flexibility and configurability (intended as the possibility of instantiating dynamically generated markup). By expanding upon the last example, it should be possible to visualize how slots enable greater flexibility.

```html
<template class="image-gallery">
    <ul class="gallery-list">
        <slot name="image-description"></slot>
        <img src="firstImage.png" alt="first image description">
```

[48] *Mozilla Developer Network - The Content Template element*. URL: `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template`. (Accessed May 17, 2021).

```
        <slot name="image-footer"></slot>
        ...
    </ul>
</template>
```

Code 3.7: Slotted templates

From now on, any valid HTML element can replace the slot placeholders by referring to their name attribute (indeed, they can also be called “named slots”). A paragraph with an image description can be expected on top of the ”img” element, also a footer containing the author’s credits and a hyperlink to other similar work can be imagined.

This procedure allows for greater reuse of specific parts of markup, the latter can be instantiated differently depending on the context. Another example comes from the work done for this dissertation project. The Query-Text Web Component – in practice a search bar – can be optionally instantiated with a “slottable” button coming from an external Web Components library with its own Shadow DOM and encapsulated behaviours (figures: 3.7 *Query-Text base class* and 3.8 *Query-Text with slotted search button* demonstrate these differences).

Overall, templates and slots in conjunction allow for a powerful composition of markup, thus preventing redundancy and shortening the code.

3.3 Web Components libraries

Web Components libraries were created with the main intent of simplifying and speeding up the development process. Libraries usually provide some sort of abstractions on top of the native Web Components for efficiency and in order to enable new features.

During the planning phase, some of the most popular and active Web Components libraries were considered for adoption: LitElement[49], FAST[50], hybrids[51], Stencil[52] and

[49] *Lit (Web Component library)*. URL: https://lit.dev/. (Accessed May 17, 2021).
[50] *FAST (Design System)*. URL: https://www.fast.design. (Accessed May 17, 2021).
[51] *Hybrids (Web Component library)*. URL: https://hybrids.js.org/. (Accessed May 17, 2021).
[52] *Stencil (Design System)*. URL: https://stenciljs.com/. (Accessed May 17, 2021).

Adobe Spectrum[53]. By looking at these proposals and the companies behind each library, it emerges that some important organisations are investing in and actually using the Web Components technology. Namely, FAST is authored by Microsoft, LitElement by Google, Adobe by Adobe, and Stencil is made by the Ionic Framework team[54]. Hybrids is an exception since it is a community based effort.

As anticipated, LitElement was the choice for the dissertation's project. It originated from Google's Polymer library. This library was mainly chosen due to its popularity, the presence of a large and thriving community involved in the project and a high quality exhaustive documentation[55]. Certainly, the features offered and the fact that LitElement is fully compatible with all the browsers which adopt the native Web Components technology was also crucial during the decision making process. Indeed, LitElement is an abstraction over the native HTMLElement. By using LitElement class-based components, developers are extending the LitElement class which in turn extends HTMLElement behind the scenes.

LitElement provides features like reactive state, scoped styles, and a declarative template system that are extensively used during this project's development. The former can also be defined as reactive properties, i.e., properties that hold the state of the component. This implies that changing one or more of the components' reactive properties automatically triggers an update cycle, re-rendering the component. From the developer point of view this is a big simplifier. Indeed, by combining another LitElement feature, declarative templates, the User Interface can be efficiently updated and almost without writing any code.

Lit templates allow to render dynamic contents thanks to many helper operators. As stated in the official documentation[56]:

- Expressions: Templates include dynamic values called expressions that can be used to render attributes, text, properties, event handlers, and even other templates.

[53]*Adobe Spectrum (Design System)*. URL: `https://spectrum.adobe.com/`. (Accessed May 17, 2021).

[54]*Ionic (framework*. URL: `https://ionicframework.com/`. (Accessed May 17, 2021).

[55]*Lit - Documentation*. URL: `https://lit.dev/docs/`. (Accessed May 17, 2021).

[56]*Lit - Templates*. URL: `https://lit.dev/docs/templates/overview/`. (Accessed May 17, 2021).

- Conditionals: Expressions can render conditional content using standard JavaScript flow control.
- Lists: Render lists by transforming data into arrays of templates using standard JavaScript looping and array techniques.

Finally, thanks to the scoped styles feature, LitElement automatically attaches a Shadow DOM to the component. This allows to define encapsulated styles that affect only the components context.

3.4 Project idea and Front End Software Architecture

In this section, a brief discussion about the project idea and its evolution will be introduced followed by a description of the adopted front-end software architecture. As mentioned in this work's introduction, the application idea was originated from the activities related to the *PH-Remix* interdisciplinary project.

The practical project work of this dissertation is mainly concerned with the development of an in-browser multimedia editing tool. This tool – called "Remix environment" or "Remix platform" in the *PH-Remix* context – allows for discovery and reuse of multimedia content.

Two phases of development can be extrapolated from a temporal perspective. In the first phase, the application was subject to goals and requirements defined for the *PH-Remix* project. In the second phase, the application was expanded upon to fit a more general scenario. These two phases ought not to be considered as mutually exclusive. On the contrary, thanks to modularity and configurability, the application can accommodate the requirements of the *PH-Remix* project as well as alternative project ideas and implementations. The latter point might be viewed as particularly interesting in the context of this work's narrative. By demonstrating that multiple applications can be made using the same components it seems that a remix can be applied to the software world. The components can be assembled like LEGO blocks alongside with other components developed from the community, an argument best exemplified in paragraph 3.7 *Examples of applications*.

The first step of this work involved defining an appropriate software architecture and was done thanks to the collaboration with the *PH-Remix* project members. Consequently, the project was initialized with the subsequent goal of producing an application prototype. The prototype can be seen as an evolutionary prototype[57] due to the adopted architecture that can be easily refined and expanded.

The front-end architecture was driven by the principles of microservices architecture[58]. Namely, the Web Components technology is used as foundation for adopting the separation of concerns design principle[59] between application components. The main choices for the adoption of Web Components adoption were made by keeping in mind the benefits of flexibility, maintainability, and scalability. Also, for the sake of efficiency and productivity LitElement was adopted as the Web Component library.

The core front-end structure is composed of the following components:

- Video-Editor-App
  - Video-Preview
  - Track-Editor
- Query-Ui
  - Query-Text
  - Result-Media

As seen in the list above, two components' levels can be distinguished. On the top-level, the Video-Editor-App and Query-Ui act as wrappers. They can instantiate the components below. They are designed for managing the communication between the client side and the server side through API calls. Therefore, they are more project specific –

[57] *Wikipedia - Evolutionary prototyping.* URL: `https://en.wikipedia.org/wiki/Software_prototyping#Evolutionary_prototyping`. (Accessed May 17, 2021).

[58] *RedHat - What are microservices?* URL: `https://www.redhat.com/en/topics/microservices/what-are-microservices`. (Accessed May 17, 2021).

[59] *Wikipedia - Separation of concerns.* URL: `https://en.wikipedia.org/wiki/Separation_of_concerns`. (Accessed May 17, 2021).

than the components at one level below – and must be configured in order to request and retrieve application specific data. Similarly, the Query-Ui acts as wrapper enabling communication between Query-Text and Result-Media components. This suggests another architectural choice. The rest of the components are isolated and are not concerned with the state of other components. Strictly, the communication with the external world is not allowed. They can be used independently in other application contexts and for different purposes. In other words, they should be operational and testable in isolation. However, communication is possible, and necessary to compose the components into a coherent application. Therefore, the following planned communication rules were established. Data can be passed top-down through properties and can be lift-up through Custom Events[60]. For example, Query-Ui receives a search string from the Query-Text component, sends it through an API to a search engine and returns the results to the media viewer, also known as Result-Media.

This architecture can be seen in Figure 3.4 *Application Mock-up*.

[60]*Mozilla Developer Network - Creating and triggering events*. URL: `https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM`. (Accessed May 17, 2021).

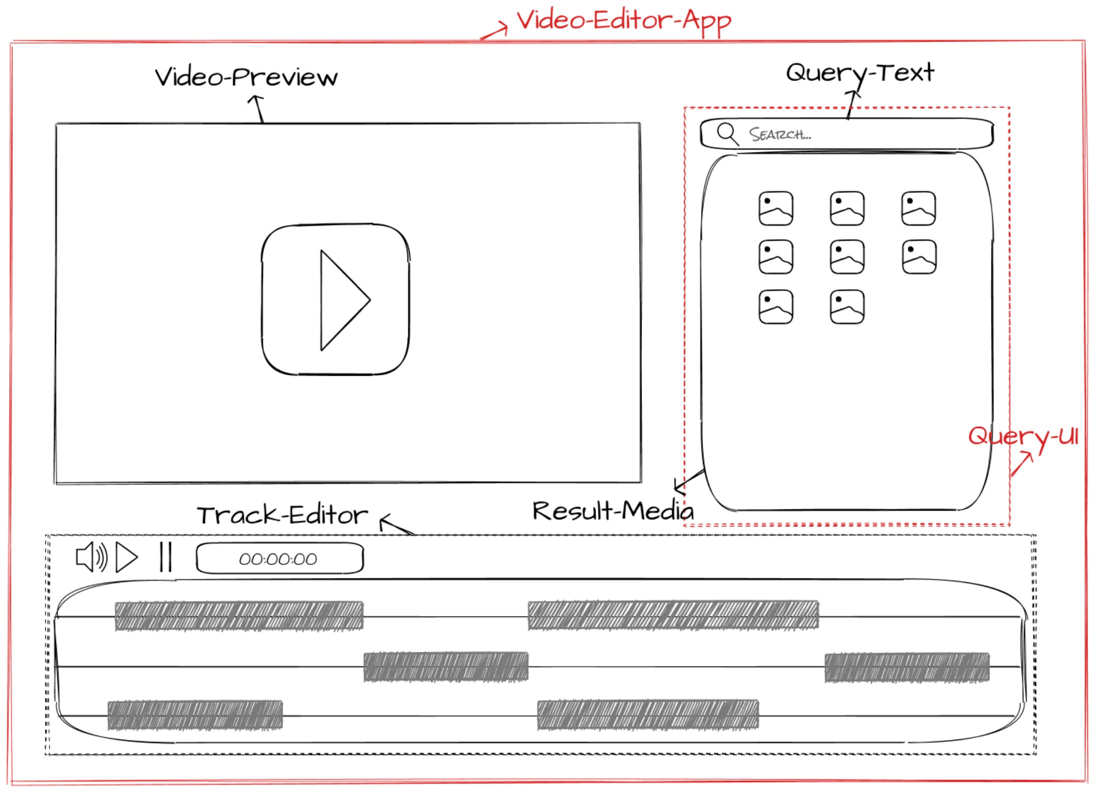


Figure 3.4: Application Mock-up

Each of these components will be explained in more details in the next sections.

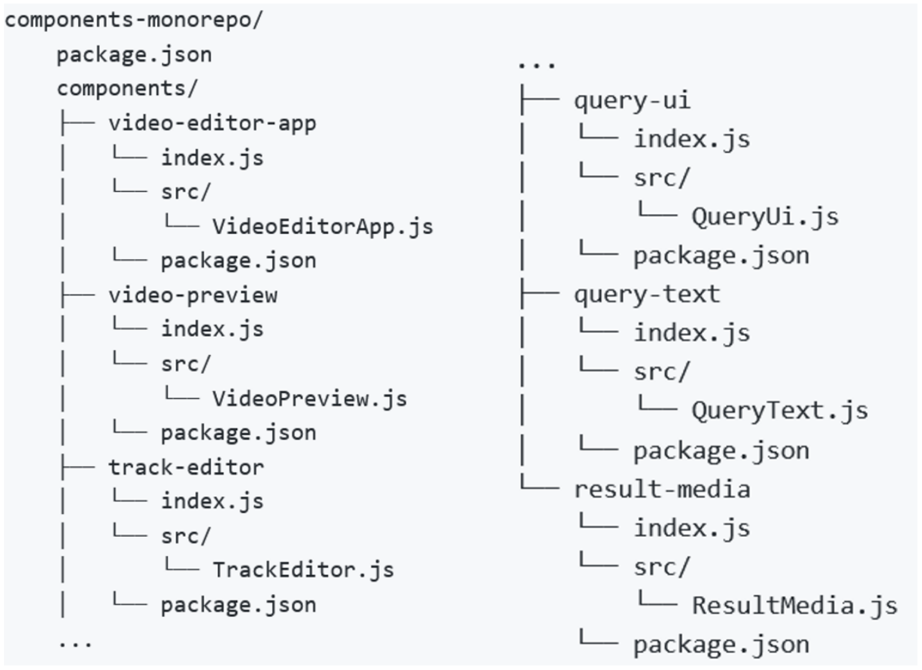
Another important aspect regards the organizational choices. That is, the way the repository is organised, how the code structure is made, how the build and deploy phases are designed.

Firstly, components were designed to be independent and therefore, they can reside within separate online repositories. From a practical point of view, after some development iterations, an alternative approach was adopted. A monorepo[61] setup was undertaken with the help of the Lerna[62] library. Monorepo is a repository which contains a number of projects inside the same centralized container. Since the components are used as dependencies which are injected or arranged into an application, this helps to organise and speed up the development. Nevertheless, nothing prevents from extracting components

[61]*Atlassian - Monorepos in Git*. URL: `https://www.atlassian.com/git/tutorials/monorepos`. (Accessed May 17, 2021).

[62]*Lerna*. URL: `https://github.com/lerna/lerna`. (Accessed May 17, 2021).

into separate repositories anytime.



```
components-monorepo/
    package.json
    components/
    ├── video-editor-app
    |    └── index.js
    |    └── src/
    |         └── VideoEditorApp.js
    |    └── package.json
    ├── video-preview
    |    └── index.js
    |    └── src/
    |         └── VideoPreview.js
    |    └── package.json
    ├── track-editor
    |    └── index.js
    |    └── src/
    |         └── TrackEditor.js
    |    └── package.json
    ...
    ├── query-ui
    |    └── index.js
    |    └── src/
    |         └── QueryUi.js
    |    └── package.json
    ├── query-text
    |    └── index.js
    |    └── src/
    |         └── QueryText.js
    |    └── package.json
    └── result-media
         └── index.js
         └── src/
              └── ResultMedia.js
         └── package.json
```

Figure 3.5: Project repository structure

The figure 3.5 shows a flat structure of components, although they can be dependent on one other. For example, Query-Ui can depend upon Query-Text specifying the latter as a project dependency inside its “package.json”. By adopting a monorepo structure the dependencies are always at hand, ready to be modified and at the same time they maintain their own commit history. Lerna is a tool that further optimizes this workflow. Among the most prominent features, it allows for dependency hoisting, versioning with the possibility of publishing components into registries like npm, and many others.

In addition, the repository structure is well suited for automation. In particular, the “index.js” file provides an entry point for every component. In this context JavaScript module bundlers can be easily implemented.

```javascript
import { QueryText } from "./src/QueryText.js";
window.customElements.define("query-text", QueryText);
```

Code 3.8: Component entry point example

The figure above depicts the Query-Text component although its structure is identical for every other component. Again, the objective is to offer a robust way of distributing the application as a whole and a future-proof structure for project evolution. Indeed, the last step of a development iteration in the software world is usually informally referred to by the colloquial expression “going to production”. The practical meaning is about publishing the code to the server and therefore offering it to the final users. At this point the common wish is to have the most optimized code possible. In the web development scenario this can be done with the help of module bundlers. Among prominent features, they allow to resolve dependencies, produce module maps for efficiency, and subsequently minify all the code.

Finally, several dependencies providing additional features are shared by all the components, therefore they will not be included in the descriptions contained in the next paragraphs. The dependencies were chosen and adopted following the Open Web Components initiative[63] indications. Their mission is to provide a list of best practices and recommendations for developing and sharing Web Components.

Therefore, a brief list of development dependencies is included in the project. Firstly, the purpose is indicated followed by a tool or a list of tools names and their brief description.

- Linting and Code checking: ESLint[64], Prettier[65].
  - These tools allow to keep consistent code formatting styles and avoid the most common problems by performing a statical code analysis.
- Testing: Karma[66], Mocha[67].
  - Testing environment for unit testing, integration testing and mocking.

[63] *Open Web Components*. URL: `https://open-wc.org/`. (Accessed May 17, 2021).
[64] *ESLint*. URL: `https://eslint.org/`. (Accessed May 17, 2021).
[65] *Prettier*. URL: `https://prettier.io/`. (Accessed May 17, 2021).
[66] *Karma - Test Runner*. URL: `https://karma-runner.github.io/latest/index.html`. (Accessed May 17, 2021).
[67] *Mocha - Test Framework*. URL: `https://mochajs.org/`. (Accessed May 17, 2021).

- Interactive documentation: Storybook[68]
  - It allows to visually describe the components with all the states they can assume.
- Building and deployment: Rollup[69]
  - JavaScript module bundler used for library or application deployment.

The next paragraphs introduce and briefly explain each component. The goal is to provide the core information about their structure and present some of the most interesting challenges encountered during development. Therefore, the next sections are not intended as a software documentation guiding through all the processes from installation to deployment. The documentation can be found by following each component's links or by exploring the project's root repository[70].

3.5 Searching and displaying results

Searching and displaying the retrieved results can be considered pivotal for applications whose success can be largely measured in terms of the quantity and the quality of offered resources. Consequently, the main benefits that come from a good implementation might involve improved content discoverability and better user engagement.

These objectives were pursued by developing three components. First, a component called Query-Ui[71] will be introduced. It is a connector between the client and server side and can be seen as a top level wrapper governing the functionalities below, of its children's components. Indeed, it instantiates two components: Query-Text, which manages the search and autocomplete feature while Result-Media is concerned with the rendering of results found. Notably, Query-Ui manages all the logic and information flow coming from

[68] *Storybook*. URL: `https://storybook.js.org/`. (Accessed May 17, 2021).

[69] *Rollup*. URL: `https://rollupjs.org/guide/en/`. (Accessed May 17, 2021).

[70] Lukasz Szczygiel. *Project's components repository*. URL: `https://github.com/lukasd2/phrmx-components`. (Accessed May 17, 2021).

[71] Lukasz Szczygiel. *Project Component - QueryUi*. URL: `https://github.com/lukasd2/phrmx-components/tree/main/components/query-ui`. (Accessed May 17, 2021).

the instantiated components. It also sends and receives data from external APIs, thereby passing data “downwards”, to the wrapped components.

An example of this relationship can be seen in the Code below.

```
<query-ui>
  <query-text
  .dictionaries=${this.dictionaries} placeholder=${this.placeholder}
   >
  </query-text>
  <result-media
  .answerSet=${this.searchResults} headerTitle=${this.headerTitle}>
  </result-media>
</query-ui>
```

Code 3.9: Query-Ui component instantiation

In this simplified example taken from one of the possible application configurations, properties and attributes are passed down to the components. Continuing with this practical example, Query-Ui internally makes an API call to the server side. In return, the endpoint responds with a dictionary – a list of words that can be autocompleted – which can be then passed to the search component. Namely, Query-Text. The principles guiding the de facto search feature are similar. The difference in implementation regards the presence of an event listener which is triggered every time users make a search. Ultimately, the results generated from the server are passed to the result viewer component, via the “answerSet” property, which renders them.

3.5.1 Search component

The Search component, named “Query-Text”[72] provides search and autocomplete functionalities within multi-keyed dictionaries of strings.

The idea behind Query-Text was to create a “google-like” search functionality based

[72]Lukasz Szczygiel. *Project Component - QueryText*. URL: `https://github.com/lukasd2/phrmx-components/tree/main/components/query-text`. (Accessed May 17, 2021).

on lists of keywords. This is done by providing a list of prefixed dictionaries during the component instantiation. The component extracts keys from the dictionary and uses them as prefixes for autocompletion purposes. An example of valid dictionaries – not to be confused with dictionary data structures present in many programming languages – is exemplified the Code below.

```
{
    "filmmaker:":
        ["Martin Scorsese", "Ridley Scott", "Woody Allen", ...]
    "#:":
        ["goat", "chair", "mountain", ...]
    ...
}
```

Code 3.10: Dictionaries examples

In this example, two prefixes are present: "filmmaker:" and ”#". When users start typing one of the recognised prefixes, autocompletion based on that prefixed list of strings will be proposed. For example, by typing "#:mou" a suggestion for the word "mountain" will be made.

Finally, by pressing the "ENTER" key or alternatively by clicking on the search button (if present), the component emits an event (through Custom Event bubble[73]) with the actual value of the query string. An important feature is implemented to provide "fuzzy search" with autocompletion when user types into the search bar. Fuzzy search allows to tackle the possibility of typing errors made by users. A configurable options list is implemented with a scoring system offered by ”Fuse.js" library[74]. Indeed, for example, if the word "montan" is typed, the user will be offered the "mountain" suggestion.

[73]*Mozilla Developer Network - Event.bubbles*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Event/bubbles`. (Accessed May 17, 2021).

[74]Kiro Risk. *Fuse.js*. URL: `https://fusejs.io/`. (Accessed May 17, 2021).

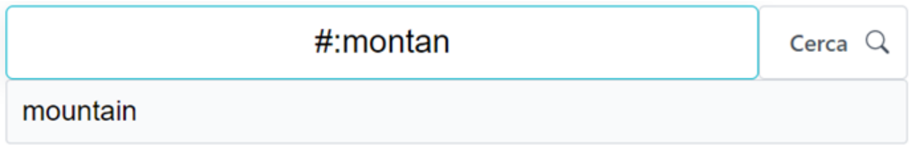


Figure 3.6: Query-Text autosuggestion feature

One of the possible issues that could arise regards the component performances. Especially when included within a more complex application, a performance-related issue emerges when users start to type fast onto the search bar. This implies the generation of a large number of DOM events and API calls if users decide to search the typed text. This issue was tackled by implementing a "debounce"[75] technique which limits the quantity of processed events. With configurable parameters, this instructs the autocomplete function to wait some arbitrary period of time in the order of milliseconds before taking any further action. Namely, it suggests words from the dictionary or sends the searched text to the search engine via API.

Finally, two examples of Query Text are exemplified in the figures below.

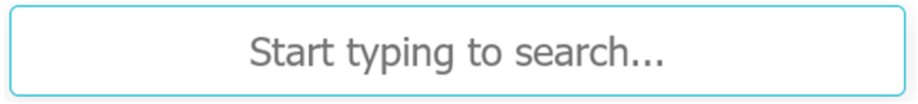


Figure 3.7: Query-Text base class

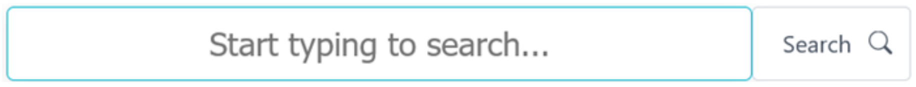


Figure 3.8: Query-Text with slotted search button

The Figure 3.7 shows the bare minimum component. Secondly, the figure 3.8 shows an example of a button instantiated within the Query-Text slot.

[75]David Corbacho. *Debouncing and Throttling Explained Through Examples*. URL: `https://css-tricks.com/debouncing-throttling-explained-examples/`. (Accessed May 17, 2021).

3.5.2 Results viewer component

As the name suggests, the Result-Media component renders some results onto the interface. For this project, search results are displayed as a gallery of thumbnails allowing for content selection and discovery. Importantly, the figures – or cards, as they are commonly called across UI libraries – can be dragged away from the component. This feature enables many possibilities as explored more in detail when introducing the Track-Editor component. Specifically, with the help of the native Drag and Drop API[76] some information about the object can be transferred through the drop event.

The adopted preview style can be seen in the Figure 3.9 *Result-Media example*.

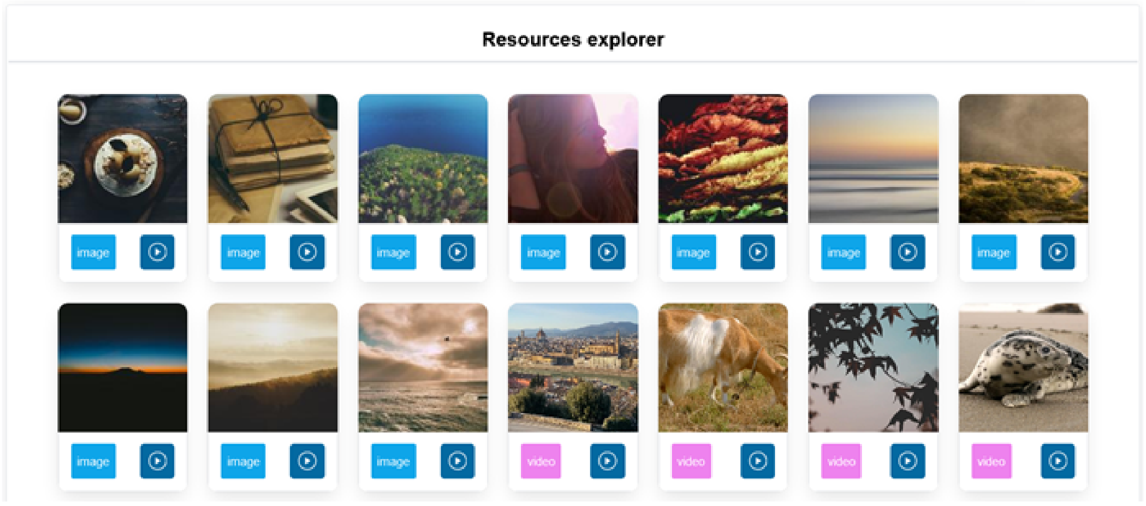


Figure 3.9: Result-Media example

As mentioned before, these figures can be dragged and dropped to any valid drop target. Indeed, when users start dragging an object from the gallery,an event is triggered. This event is connected to the creation of a DataTransfer object[77] containing all the necessary information to be transferred. For example, dragging a thumbnail that represents a video implies the insertion of the video duration onto the transferred data object. The latter can be read once the dragged item is dropped.

[76] *Mozilla Developer Network - Drag and Drop*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API`. (Accessed May 17, 2021).

[77] *Mozilla Developer Network - Data Transfer*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer`. (Accessed May 17, 2021).

Additional features can also be optionally enabled. Since image thumbnails might not always be self-explanatory, the tooltip feature is also implemented in this project. Users can find further details about the resource by "hovering" on top of each media result. Continuing with the video result example, a tooltip may contain information about its title, duration, etc. Lastly, users might also be interested in reproducing the media directly inside the application. This is done by pressing the "play" button in combination with the Video-Player component which will be discussed in 3.6.2 *Media Preview component*.

3.6 Multimedia Editor

The most characterizing piece of this project is the Multimedia Editor. It is a top level wrapper that orchestrates the application specific behaviour. The Multimedia Editor is called "Video-Editor-App" in the project repository but by design it should be re-named and configured in relation to the application purpose. In this case, it instantiates the Track-Editor and the Video-Preview components.

As a result, the application allows to combine and arrange image, sound, and video resources by dragging them from the Result-Media component into an interactive track. Subsequently, the elements arranged in the track can be viewed inside a player and potentially exported with the help of a back end infrastructure. Although combine and preview operations are the concern of the respective components, the Multimedia Editor has a key role in making them work together. Similarly to Query-Ui, it can communicate with the outside world through APIs.

The following markup reflects the relationship between components.

```
<video-editor-app>
      <video-preview
        .resources=${this.resources}
        .singleMediaPreview=${this.singleMediaPreview}
        .executeSegmentsPreview=${this.playSegments}
        .terminateSegmentsPreview=${this.endSegments}
      ></video-preview>
      <query-ui></query-ui>
      <track-editor
        draggedElementType=${this.draggedElementType}
      ></track-editor>
</video-editor-app>
```

Code 3.11: Video-Editor-App component instantiation

The figure shows some of the properties that are used for synchronising the Track-Editor with its preview. As a matter of fact, the Video-Editor-App also manages the data flow originated from the Track-Editor. This cannot be seen in the markup because of the already mentioned design decision: "data can be passed top-down through properties and can be lift-up through Custom Events". The latter explains the logic behind the interaction. This can be further demonstrated with the help of a practical example.

A frequent scenario involves users that compose multimedia resources inside the editing track by dropping them from the content library of choice. At any moment it is possible to reproduce the arrangements made within Track-Editor component. At this point, the latter component sends an event upwards. This event is composed of a list of media elements – alongside information about their start and end dates – that were present inside the track at the exact time the "play" button was pressed. This information is captured by the Video-Editor-App whose job is to request media from external APIs. Lastly, when a successful response is available, the data is passed through to the Video-Preview, ready to be played. This behaviour will be discussed in further detail in the next sections.

3.6.1 Track Editor component

"Track-Editor"[78] is the most complex component created for this project application. It offers the core feature of arranging media content similarly to the way it is implemented by some of the alternative software for video-editing. Its peculiarity is that it offers this functionality directly from the browser level.

[78]Lukasz Szczygiel. *Project Component - TrackEditor*. URL: `https://github.com/lukasd2/phrmx-components/tree/main/components/track-editor`. (Accessed May 17, 2021).

In detail, Track-Editor can be composed of multiple tracks. Depending on the configuration, an arbitrary number of tracks can be used. Tracks refer to specific media types, for example: an audio, video, and image track. Every track has a temporal dimension which allows to perform some basic operations that are key for any editing tool. Indeed, a temporal marker is also implemented for keeping track of the currently reproduced media. This can be seen in the example below:

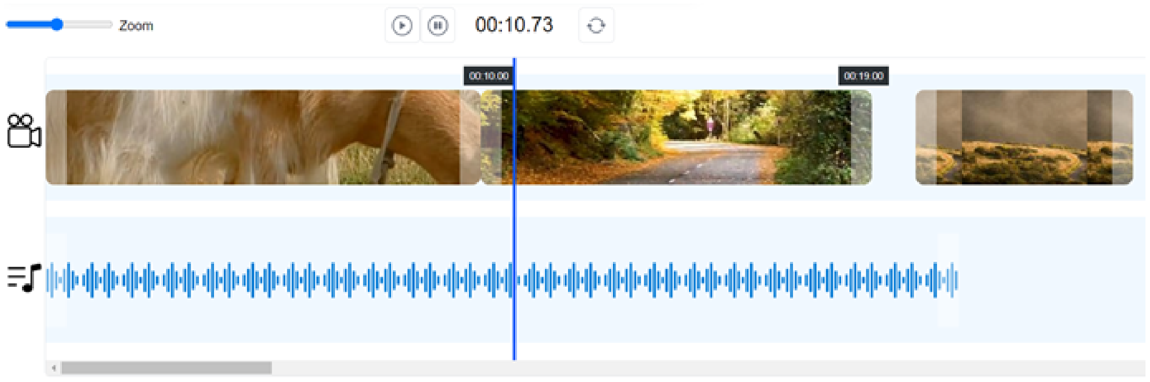


Figure 3.10: Track-Editor example

Currently, the Track-Editor also allows to:

- Receive dragged media from the application library.
- Arrange media elements inside the tracks by dragging them back and forth.
- Resize or trim media in relation to their type.
- Dynamically zoom in and out the track and re-dimension all the content inside of it.
- Check whether a specific track can contain the dragged media type.
- Play, pause, and reset the media reproduction.
- Open a context menu for removing undesired media.

Some interesting points can be made from this list. First of all, operating with a content length of a few seconds is different from content whose duration might be in order of several minutes. This potential problem is mitigated by the zooming functionality. Also, some basic form of error prevention has been made to preserve the preview functionality. This is done by generating an informative message on the UI stating that a specific media type cannot be put inside a track.

Finally, the ability of playing, stopping, and resetting the current tracks state is synchronized with the temporal marker – namely, the blue line in 3.10 *Track-Editor example*.

From the implementation perspective, when users press play, a Custom Event is dispatched. This event can be captured thanks to its bubble property, as showed in the Code below.

```
function dispatchEventForPreview() {
    const event = new CustomEvent('track-elements', {
      detail: {
        trackElements: this.segmentsOnTracks,
      },
      bubbles: true,
      composed: true,
    });
    this.dispatchEvent(event);
  }
```

Code 3.12: Track-Editor event dispatch

This event transfers the current state of the Track-Editor. At a later stage this event can be captured and used for generating API requests.

3.6.2 Media Preview component

The final part of this application is the media preview. The component that manages the preview is called "Video-Preview"[79]. As discussed in the previous paragraph, its main purpose consists in reproducing a synchronised preview of the media from the Track-Editor. Generally speaking, it can preview any number of media content, starting from a single preview to a large number of media with different media types.

[79]Lukasz Szczygiel. *Project Component - VideoPreview*. URL: https://github.com/lukasd2/phrmx-components/tree/main/components/video-preview. (Accessed May 17, 2021).

An interesting implementation choice can be derived from the synchronization feature. It might be argued that the component has intuitively some sort of an internal clock or at least some way of keeping track of the elapsed time. As a matter of fact, in this case this is not entirely true. Contrary to the Track-Editor, which implements an internal timer that guides the component state. It turns out that Video-Preview does not necessarily need one. And yet it works. Walking through an example may effectively demonstrate how things work internally. Before jumping into the example, a possible way of instantiating the component is presented in the Code below.

```
<video-preview
        ?displayLoadingScreen=${this.displayLoadingScreen}
        ?stopPlayer=${this.stopPlayer}
        ?resumePlayer=${this.resumePlayer}
        .resources=${this.resources}
        .executeSegmentsPreview=${this.playSegments}
        .terminateSegmentsPreview=${this.endSegments}
></video-preview>
```

Code 3.13: Video-Preview component instantiation

In particular, the “resources” property contains all the media items to be previewed. Hence, a video would be inserted into a HTML video source tag, an image would reside within an HTML img source and so on.

Assuming the scenario of Track-Editor interaction with the Video-Preview component, after putting an arbitrary number of media elements inside the tracks, the user presses the “play” button. Subsequently the following steps occur:

1. All the media references are sent through a Custom Event.
2. The Event is captured by the top level Video-Editor-App component. It requests all the items using API calls.
3. The responses arrive asynchronously and are sent to the Video-Preview component for preview.

4. Track-Editor starts its internal timer and checks if at the time 00:00:00 (in the minutes/seconds/milliseconds format) any media element has been positioned into the track. If yes, the next step is executed, otherwise the timer runs until it encounters an element on the track.
5. Track-Editor emits the "executeSegmentsPreview" event containing one or more references to media elements that must be previewed.
6. The Event is captured by the top level Video-Editor-App component and passed immediately to the Video Preview component.
7. Video Preview selects the right resource(s) – from the resource list received in the step 3 – and starts the preview.
8. Once the media element duration ends, the Track-Editor sends a "terminateSegmentsPreview" event.
9. The Event is captured by the top level Video-Editor-App component and passed immediately to the Video Preview component.
10. Video Preview selects the right resource(s) – from the resource list received in the step 3 – and stops the preview.

This workflow – apart from the asynchronous API operations – happens with a near real time precision and continues as long as the user does not stop the reproduction or there are no more elements to be played on tracks.

As exemplified, the above mechanism makes the whole synchronized preview system work without the necessity of implementing additional timers. The Track-Editor governs the current items being played. By adopting this implementation, the Video-Preview component is simpler and more performant.

3.7 Application examples

The main application example is a prototype made for the *PH-Remix* project. This application uses resources currently protected by copyright which belong to the Mediateca

Toscana. At the time of writing, they are mainly documentaries. In this case the resources are short videos extracted from the original works. This is done by Artificial Intelligence algorithms as explained in this dissertation's introduction. Figure 3.11 shows the application interface which reflects the components architecture as determined in section 3.4 *Project idea and Front End Software Architecture*.

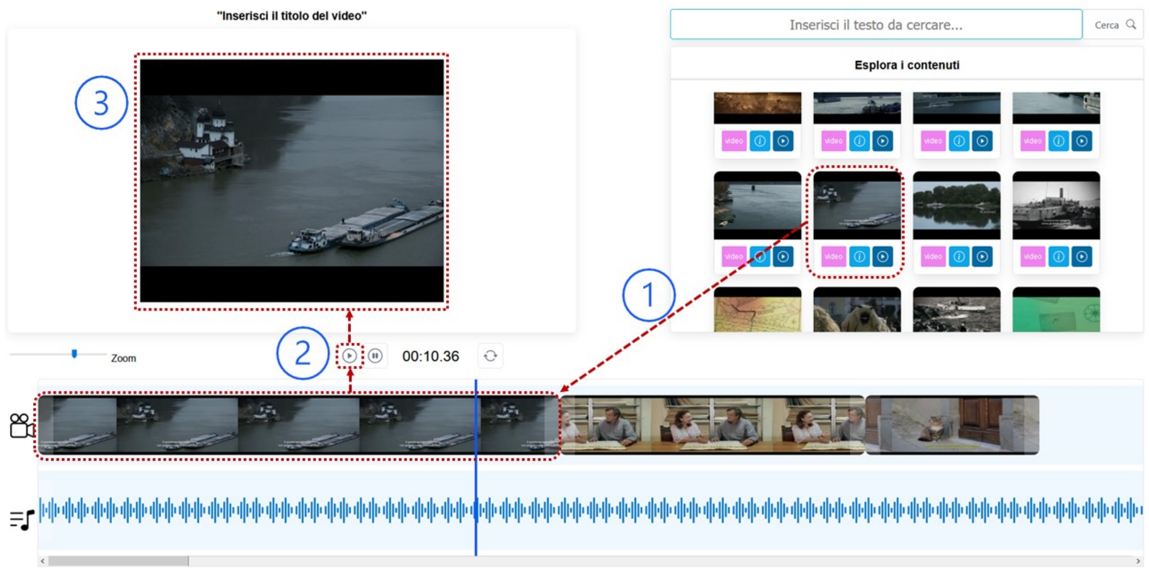
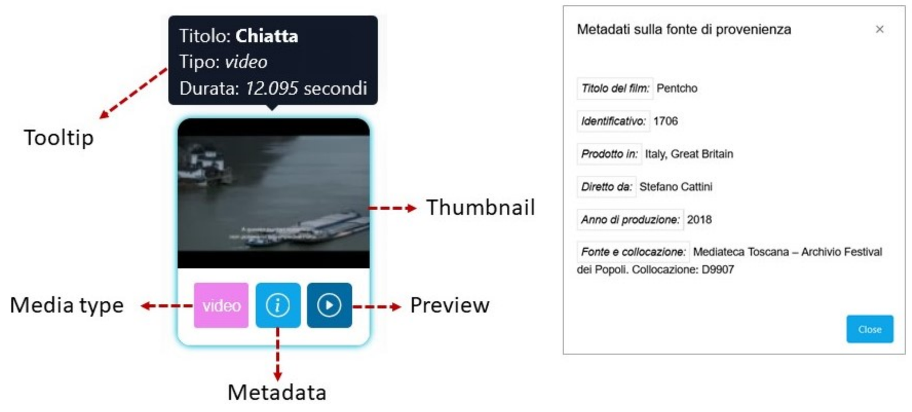

"Inserisci il titolo del video"
Inserisci il testo da cercare...
Cerca
Esplora i contenuti
video
1
2
3
Zoom
00:10.36

Figure 3.11: *PH-Remix* annotated User Interface

Firstly, four components can be seen directly in this example. The search bar or Query-Text component is positioned in the top-right corner. Then the middle-right side shows the content library. It can be explored thanks to the thumbnails, that is representative images extracted from each video clip. This is the part managed by the Result-Media component. At the bottom there is a track editor with two different types of media tracks. A video track on top and an audio track on the bottom. This part is managed by the Track-Editor component and it allows for positioning and arranging the media from the content library. Finally, on the top-left side a preview of the content can be reproduced. The Video-Preview component is used for displaying the elements from the editing track, as well as single video clips directly from the content library.

The Figure 3.11 has also been annotated to show some common actions that were explained earlier in this section. There are three steps necessary to select, insert and preview the content from the library. Step 1 consists in selecting the video clip to be arranged or edited inside the track. This is done by dragging the thumbnail directly into the corresponding media track. In this case a video clip is dragged to the video track. As seen in the interface, the latter proportionally expands the thumbnail length horizontally according to the duration of the video clip. This action can be repeated for every resource in the library. Subsequently, step 2 triggers the preview of the content that was positioned on the track. This happens by pressing the "play" button above the Track-Editor. Finally, step 3 allows to preview the sequence of the arranged content. This is the most common scenario for multimedia editing and can be repeated until the desired solution is reached.

Consequently, a more fine-grained look at the interface is shown by the figure below.



Metadati sulla fonte di provenienza ×

Titolo del film: Pentcho

Identificativo: 1706

Prodotto in: Italy, Great Britain

Diretto da: Stefano Cattini

Anno di produzione: 2018

Fonte e collocazione: Mediateca Toscana – Archivio Festival dei Popoli. Collocazione: D9907

Close

Figure 3.12: *PH-Remix*: Single resource entry (on the left) and modal window with details about the resource (on the right)

This shows an annotated example (from the Italian version) of a single resource entry. The left-hand side displays the element state while the cursor is "hovering" over it. When users place their cursor on top of a library element, a tooltip with some basic information appears to disambiguate the nature of the resource. In this case the thumbnail seems to depict some sort of a barge floating on water. Indeed, the tooltip describes it as a "Barge" ("Chiatta" in Italian) alongside with the media type (video) and its duration in seconds. Below the thumbnail three elements of the interface allows to perform some actions with the element. These are always visible independently from the item's state (like the aforementioned positioning of the cursor on top of it). Going from left to right, the media type is also specified within a coloured box, with a different colour for each media type. Then in the centre some more information about the object can be shown as exemplified on the right side of Figure 3.12. This is particularly useful for content with cultural value. Indeed, in this case some metadata about the extracted clip and its original source are shown. For instance, the image displays the name of the documentary, its director, year of production, physical location, etc.

The last button allows to directly preview the video clip inside the Video-Preview component. This is useful to evaluate whether the visual content of the selected items is relevant for the editing action. Then it can be dragged to the track editor, as explained above.

On the other hand, the second example is a general-purpose application which integrates the resource from the Pexels API. Although at the time of writing the User Interface is almost exactly similar to the one of the *PH-Remix*, its peculiarity is that – thanks to the permissive license of the Pexels platform – it allows for legal remixing of content by everyone. As a matter of fact, this consents unconstrained browsing of a rich Open Source catalogue of content. In particular, it consists of thousands of videos and images that can be freely built upon. Apart from the nature of the content itself, some features that are present in the *PH-Remix* example are not present in the general-purpose application and vice versa. For instance the ability of viewing object metadata is relevant to cultural heritage objects, but it may not be necessary for simple pictures made for fun.

Conclusions

This dissertation shows the interdisciplinary nature of the Remix Culture phenomenon, highlighting its importance in the modern society. The definitions taken from the topic's literature and examples from the real world were aimed at showing a holistic view of the argument. The purpose was to present an original interpretation of the narrative starting from the theory through organisational and business topics to end with the development of a practical application. The latter enables users to directly experience the Remix Culture with as few technological limits as possible.

The application developed for this dissertation is a prototype, but the development process already provided some valuable skills and ideas that led to considerations about some best practices and arguments in favour of a standardised technology for creating components in the front-end ecosystem. As a matter of fact, one of the discussed ideas regarded the possibility of remixing software thanks to a component-based approach. Although this is already possible, a substantial margin for improvement is also possible and desirable. Namely, the Web Components that are shared and published by the community do not often exactly work in a plug and play manner. At the time of writing, building an application that uses existing external components – built using different libraries and for different purposes – is not as easy as it would seem in theory. Notably, some weak points are to be reconducted to the lack of a comprehensive documentation guiding through all the installation steps, but some difficulties with interoperability are also common. In general, it seems that many existing components are not very developer-friendly. Nevertheless, this approach seems promising and can be considered as a valuable asset for the current and future software developers with the hope that remixing components into new applications will proliferate. These considerations also lead towards some ideas for future works and studies that may further explore the criteria and effective ways for sharing and re-using software code.

The other critical point that emerged during the investigations was connected to the Intellectual Property Rights and their relation to the activity of re-use. Standing to current legislation many scholars and researchers advocate in favour of adapting the laws to the digital era. At the same time, the analysis of the Open Source and Creative Commons was presented to demonstrate viable alternatives to the current legislative issues and uncertainties. However, it was also demonstrated that these open approaches should not be considered as an ideal solution for every project and creative work. They are rather to be considered as an additional asset that can be used for creating sustainable hybrid economies where Open Source solutions are mixed with the proprietary ones guided by copyright rules.

Finally, one important goal of this works consisted in spreading awareness about the possibilities that the open world and permissive licensing offer. This is particularly important for some disciplines involved with the creation of art and culture. At the time of writing the GLAM (Galleries, Libraries, Archives and Museums) institutions are frequently involved in digitisation strategies that also include finding new ways to encourage content dissemination and user participation. The recent events, specifically the pandemic crisis, further showed that these innovations strategies are crucial for the culture sector. From this perspective the *PH-Remix* project seems like an ideal candidate to explore the advantages of the Remix Culture when adopted by the cultural heritage organisations. While it is too early to decide whether the impact of the project will encourage and transform the traditional ways of content fruition, the result of the work seems promising and worth investigating once the project becomes fully operative.

To conclude, this dissertation showed some points of interest and possible topics for future research. The narrative was guided by an interdisciplinary approach. Hopefully, the latter may provide answers to some pressing questions that have not yet been solved by using more generic discipline-oriented approaches. It is also to be hoped that it will ultimately help in solving some of the current issues, thereby have a positive impact in transforming the way the society sees and uses the Remix Culture.

Bibliography

[1] *“Derivative work.” Merriam-Webster.com Legal Dictionary*. URL: `https://www.merriam-webster.com/legal/derivative%20work`. (Accessed May 8, 2021).

[2] *17 U.S. Code § 101 - Definitions*. URL: `https://www.law.cornell.edu/uscode/text/17/101`. (Accessed May 17, 2021).

[3] *Adobe Spectrum (Design System)*. URL: `https://spectrum.adobe.com/`. (Accessed May 17, 2021).

[4] *Agile Alliance - What is Agile?* URL: `https://www.agilealliance.org/agile101/`. (Accessed May 17, 2021).

[5] *Atlassian - Monorepos in Git*. URL: `https://www.atlassian.com/git/tutorials/monorepos`. (Accessed May 17, 2021).

[6] *AXA - Components Design System*. URL: `https://design.axa.com/web-guidelines/design-system`. (Accessed May 17, 2021).

[7] *Babel - JavaScript compiler*. URL: `https://babeljs.io/`. (Accessed May 17, 2021).

[8] Kevin Ball. *The increasing nature of frontend complexity*. URL: `https://blog.logrocket.com/the-increasing-nature-of-frontend-complexity-b73c784c09ae/`. (Accessed May 17, 2021).

[9] *BBC Moving Words - Sir Isaac Newton*. URL: `https://www.bbc.co.uk/worldservice/learningenglish/movingwords/shortlist/newton.shtml`. (Accessed May 17, 2021).

[10] *Berne Convention for the Protection of Literary and Artistic Works*. URL: https://wipolex-res.wipo.int/edocs/lexdocs/treaties/en/berne/trt_berne_001en.html. (Accessed May 17, 2021).

[11] Eric Bidelman. *Custom Elements v1: Reusable Web Components*. URL: https://developers.google.com/web/fundamentals/web-components/customelements#reactions. (Accessed May 17, 2021).

[12] James Boyle. *The Economyst - The fight to keep ideas open to all*. URL: https://www.economist.com/open-future/2018/12/12/the-fight-to-keep-ideas-open-to-all. (Accessed May 17, 2021).

[13] *Bring It On Home (song)*. URL: https://en.wikipedia.org/wiki/Bring_It_on_Home_(Sonny_Boy_Williamson_II_song). (Accessed May 17, 2021).

[14] Frederick P. Brooks. *The Mythical Man-Month (Anniversary Ed.)* USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201835959.

[15] *Can I use... - Custom Elements*. URL: https://caniuse.com/custom-elementsv1. (Accessed May 17, 2021).

[16] *Can I use... - ES6 Modules*. URL: https://caniuse.com/es6-module. (Accessed May 17, 2021).

[17] *Can I use... - Web Components*. URL: https://caniuse.com/?search=components. (Accessed May 17, 2021).

[18] *Canva - Video maker*. URL: https://www.canva.com/create/videos/. (Accessed May 17, 2021).

[19] Clayton M. Christensen. *Disruptive Innovation*. URL: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/disruptive-innovation. (Accessed May 17, 2021).

[20] *Copyright Act of 1976*. URL: https://en.wikipedia.org/wiki/Copyright_Act_of_1976. (Accessed May 17, 2021).

[21] *Copyright Term Extension Act*. URL: `https://en.wikipedia.org/wiki/Copyright_Term_Extension_Act`. (Accessed May 17, 2021).

[22] David Corbacho. *Debouncing and Throttling Explained Through Examples*. URL: `https://css-tricks.com/debouncing-throttling-explained-examples/`. (Accessed May 17, 2021).

[23] *Creative Commons - CC0*. URL: `https://creativecommons.org/publicdomain/zero/1.0/`. (Accessed May 17, 2021).

[24] *Creative Commons - Licenses*. URL: `https://creativecommons.org/about/cclicenses/`. (Accessed May 17, 2021).

[25] *Creative Commons' Response to COVID-19*. URL: `https://creativecommons.org/creative-commons-response-to-covid-19/`. (Accessed May 17, 2021).

[26] *Definition of chilling effect from the Collins English Dictionary*. URL: `https://www.collinsdictionary.com/dictionary/english/chilling-effect`. (Accessed May 17, 2021).

[27] *Directive (EU) 2019/790 on copyright and related rights in the Digital Single Market*. URL: `https://eur-lex.europa.eu/eli/dir/2019/790/oj`. (Accessed May 17, 2021).

[28] Rob Dodson. *Custom Elements Everywhere*. URL: `https://custom-elements-everywhere.com/`. (Accessed May 17, 2021).

[29] *ECMAScript® 2020 language specification*. URL: `https://www.ecma-international.org/publications-and-standards/standards/ecma-262/`. (Accessed May 17, 2021).

[30] *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*. URL: `https://www.iso.org/standard/77520.html`. (Accessed May 17, 2021).

[31] *ESLint*. URL: `https://eslint.org/`. (Accessed May 17, 2021).

[32] *Everything is a remix*. URL: `https://www.everythingisaremix.info/`. (Accessed May 17, 2021).

[33] *FAST (Design System)*. URL: `https://www.fast.design`. (Accessed May 17, 2021).

[34] Kirby Ferguson. *Everything is a remix - Part 3*. URL: `https://www.everythingisaremix.info/blog/everything-is-a-remix-part-3-transcript`. (Accessed May 17, 2021).

[35] Federico Ferri. "The dark side(s) of the EU Directive on copyright and related rights in the Digital Single Market". In: *China-EU Law Journal* (2020). URL: `https://doi.org/10.1007/s12689-020-00089-5`.

[36] *Free Beer (project)*. URL: `http://freebeer.org/blog/`. (Accessed May 17, 2021).

[37] *Free Software Foundation Europe - Software Patents in Europe*. URL: `https://fsfe.org/activities/swpat/swpat.en.html`. (Accessed May 17, 2021).

[38] Owen Gallagher, Eduardo Navas, and xtine burrough xtine. *The Routledge Companion to Remix Studies*. Jan. 2015, p. 11. ISBN: 978-0415716253.

[39] *Git - Version Control System*. URL: `https://git-scm.com/`. (Accessed May 17, 2021).

[40] *GNU - Licenses*. URL: `https://www.gnu.org/licenses/license-list.html.en`. (Accessed May 17, 2021).

[41] *GNU - What is free software*. URL: https://www.gnu.org/philosophy/free-sw.en.html. (Accessed May 17, 2021).

[42] *Google support - How Content ID works*. URL: https://support.google.com/youtube/answer/2797370?hl=en. (Accessed May 17, 2021).

[43] Terry Hart. *Remix Without Romance: What Free Culture Gets Wrong*. URL: https://www.copyhype.com/2012/04/remix-without-romance-what-free-culture-gets-wrong/. (Accessed May 17, 2021).

[44] Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd. USA: No Starch Press, 2018, pp. 173–184. ISBN: 1593279507.

[45] Cole Hoover. *Human-Centered Design vs. Design-Thinking: How They're Different and How to Use Them Together to Create Lasting Change*. URL: https://blog.movingworlds.org/human-centered-design-vs-design-thinking-how-theyre-different-and-how-to-use-them-together-to-create-lasting-change/. (Accessed May 17, 2021).

[46] *Hybrids (Web Component library)*. URL: https://hybrids.js.org/. (Accessed May 17, 2021).

[47] *IBM - Carbon Design System*. URL: https://github.com/carbon-design-system/carbon-web-components. (Accessed May 17, 2021).

[48] *ING - Lion Components*. URL: https://github.com/ing-bank/lion. (Accessed May 17, 2021).

[49] *Intellectual Property Rights-uses*. URL: https://www.longdom.org/peer-reviewed-journals/intellectual-property-rightsuses-1012.html. (Accessed May 17, 2021).

[50] *Interaction Design Foundation - Fitts' Law*. URL: `https://www.interaction-design.org/literature/topics/fitts-law`. (Accessed May 17, 2021).

[51] *Ionic (framework*. URL: `https://ionicframework.com/`. (Accessed May 17, 2021).

[52] Zack Jackson. *Webpack 5 Module Federation*. URL: `https://medium.com/swlh/webpack-5-module-federation-a-game-changer-to-javascript-architecture-bcdd30e02669`. (Accessed May 17, 2021).

[53] *jQuery - Selectors API*. URL: `https://api.jquery.com/category/selectors/`. (Accessed May 17, 2021).

[54] *Karma - Test Runner*. URL: `https://karma-runner.github.io/latest/index.html`. (Accessed May 17, 2021).

[55] Matthew Kotchen. "Public Goods". In: (2012). URL: `https://environment.yale.edu/kotchen/pubs/pgchap.pdf`.

[56] Serhii Kulykov. *Beyond the polyfills: how Web Components affect us today?* URL: `https://dev.to/webpadawan/beyond-the-polyfills-how-web-components-affect-us-today-3j0a`. (Accessed May 17, 2021).

[57] Serhii Kulykov. *The journey of Web Components: wrong ways, lacking parts and promising paths*. URL: `https://dev.to/webpadawan/the-journey-of-web-components-wrong-ways-lacking-parts-and-promising-paths-1d5a`. (Accessed May 17, 2021).

[58] *Lerna*. URL: `https://github.com/lerna/lerna`. (Accessed May 17, 2021).

[59] L. Lessig. "Remix: Making Art and Commerce Thrive in the Hybrid Economy". In: 2008, p. 103.

[60] L. Lessig. "Remix: Making Art and Commerce Thrive in the Hybrid Economy". In: 2008.

[61] *License proliferation*. URL: https://en.wikipedia.org/wiki/License_proliferation. (Accessed May 17, 2021).

[62] *Lit - Documentation*. URL: https://lit.dev/docs/. (Accessed May 17, 2021).

[63] *Lit - Templates*. URL: https://lit.dev/docs/templates/overview/. (Accessed May 17, 2021).

[64] *Lit (Web Component library)*. URL: https://lit.dev/. (Accessed May 17, 2021).

[65] *Machinima*. URL: https://en.wikipedia.org/wiki/Machinima. (Accessed May 17, 2021).

[66] *Magisto - Video Editor*. URL: https://www.magisto.com/. (Accessed May 17, 2021).

[67] Lev Manovich. “Remixability and Modularity”. In: (2005), p. 2. URL: http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf.

[68] Lev Manovich. “Remixability and Modularity”. In: (2005), p. 7. URL: http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf.

[69] Lev Manovich. “Remixability and Modularity”. In: (2005), p. 3. URL: http://manovich.net/content/04-projects/046-remixability-and-modularity/43_article_2005.pdf.

[70] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st ed. USA: Prentice Hall PTR, 2008. ISBN: 0132350882.

[71] *Mashup (culture)*. URL: https://en.wikipedia.org/wiki/Mashup_(culture). (Accessed May 17, 2021).

[72] *Mediateca Toscana*. URL: https://www.mediatecatoscana.it/. (Accessed May 17, 2021).

[73] *MediaWiki platform*. URL: https://www.mediawiki.org/wiki/MediaWiki. (Accessed May 17, 2021).

[74] *Mickey Mouse in the public domain*. URL: http://copyright.nova.edu/mickey-public-domain/. (Accessed May 17, 2021).

[75] *Micro-Frontends*. URL: https://micro-frontends.org/. (Accessed May 17, 2021).

[76] *Mocha - Test Framework*. URL: https://mochajs.org/. (Accessed May 17, 2021).

[77] Patrick Moorhead. *You Will Care About Intellectual Property Sooner Or Later*. URL: https://www.forbes.com/sites/patrickmoorhead/2017/10/05/you-will-care-about-intellectual-property-sooner-or-later/?sh=4c0ba0471593. (Accessed May 17, 2021).

[78] *MoSCoW Prioritisation technique*. URL: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation. (Accessed May 17, 2021).

[79] *Mozilla Developer Network - Creating and triggering events*. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM. (Accessed May 17, 2021).

[80] *Mozilla Developer Network - CustomElementRegistry*. URL: https://developer.mozilla.org/en-US/docs/Web/API/CustomElementRegistry. (Accessed May 17, 2021).

[81] *Mozilla Developer Network - Data Transfer*. URL: https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer. (Accessed May 17, 2021).

[82] *Mozilla Developer Network - Drag and Drop*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API`. (Accessed May 17, 2021).

[83] *Mozilla Developer Network - Element.attachShadow*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Element/attachShadow`. (Accessed May 17, 2021).

[84] *Mozilla Developer Network - Event.bubbles*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Event/bubbles`. (Accessed May 17, 2021).

[85] *Mozilla Developer Network - HTML Imports*. URL: `https://developer.mozilla.org/en-US/docs/Web/Web_Components/HTML_Imports`. (Accessed May 17, 2021).

[86] *Mozilla Developer Network - HTMLElement*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement`. (Accessed May 17, 2021).

[87] *Mozilla Developer Network - JS Modules*. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules`. (Accessed May 17, 2021).

[88] *Mozilla Developer Network - The Content Template element*. URL: `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template`. (Accessed May 17, 2021).

[89] *Mozilla Developer Network - Using shadow DOM*. URL: `https://developer.mozilla.org/en-US/docs/Web/Events/Creating_and_triggering_events`. (Accessed May 17, 2021).

[90] *Mozilla Developer Network - Web Components*. URL: `https://developer.mozilla.org/en-US/docs/Web/Web_Components`. (Accessed May 17, 2021).

[91] *Network effect*. URL: https://en.wikipedia.org/wiki/Network_effect. (Accessed May 17, 2021).

[92] *Nils Svensson and Others v Retriever Sverige AB*. URL: https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:62012CJ0466. (Accessed May 17, 2021).

[93] *Open Web Components*. URL: https://open-wc.org/. (Accessed May 17, 2021).

[94] *OpenCola (project)*. URL: https://en.wikipedia.org/wiki/OpenCola_(drink). (Accessed May 17, 2021).

[95] Sarah Dominique Orlandi et al. *FAQs AUTHOR'S RIGHT, COPYRIGHT AND FREE LICENSES FOR CULTURE ON THE WEB*. Zenodo, Mar. 2021. DOI: 10.5281/zenodo.4608430. URL: https://doi.org/10.5281/zenodo.4608430.

[96] *Peer to peer*. URL: https://en.wikipedia.org/wiki/Peer-to-peer. (Accessed May 17, 2021).

[97] *Pexels - Photos and Videos*. URL: https://www.pexels.com/. (Accessed May 17, 2021).

[98] Jeffrey Pomerantz and Robin Peek. "Fifty shades of open". In: *First Monday* 21.5 (2016). DOI: 10.5210/fm.v21i5.6360. URL: https://journals.uic.edu/ojs/index.php/fm/article/view/6360.

[99] *Prettier*. URL: https://prettier.io/. (Accessed May 17, 2021).

[100] Monisha Ravisetti. *The Academic Times - First vaccine to fully immunize against malaria*. URL: https://academictimes.com/first-vaccine-to-fully-immunize-against-malaria-builds-on-pandemic-driven-rna-tech/. (Accessed May 17, 2021).

[101] *React JS - Design Principles*. URL: https://reactjs.org/docs/design-principles.html. (Accessed May 17, 2021).

[102] *RedHat - What are microservices?* URL: https://www.redhat.com/en/topics/microservices/what-are-microservices. (Accessed May 17, 2021).

[103] Leon Revill. *Open vs. Closed Shadow DOM*. URL: https://blog.revillweb.com/open-vs-closed-shadow-dom-9f3d7427d1af. (Accessed May 17, 2021).

[104] Greg Lockwood Richard Koch. *Simplify*. Piatkus, 2016, pp. 3–10. ISBN: 9780349411859.

[105] Greg Lockwood Richard Koch. *Simplify*. Piatkus, 2016, pp. 11–21. ISBN: 9780349411859.

[106] *RiP!: A Remix Manifesto*. URL: https://en.wikipedia.org/wiki/RiP!:_A_Remix_Manifesto. (Accessed May 17, 2021).

[107] Kiro Risk. *Fuse.js*. URL: https://fusejs.io/. (Accessed May 17, 2021).

[108] *Rivalry (economics)*. URL: https://en.wikipedia.org/wiki/Rivalry_(economics). (Accessed May 17, 2021).

[109] *Rollup*. URL: https://rollupjs.org/guide/en/. (Accessed May 17, 2021).

[110] Guilda Rostama. *WIPO Magazine - Remix Culture and Amateur Creativity: A Copyright Dilemma*. URL: https://www.wipo.int/wipo_magazine/en/2015/03/article_0006.html. (Accessed May 17, 2021).

[111] Alex Russell. *Web Components and Model Driven Views*. URL: https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell. (Accessed May 17, 2021).

[112] *Sense of Community or psychological sense of community*. URL: https://en.wikipedia.org/wiki/Sense_of_community. (Accessed May 17, 2021).

[113] *Spirit copyright infringement lawsuit*. URL: https://en.wikipedia.org/wiki/Stairway_to_Heaven#Spirit_copyright_infringement_lawsuit. (Accessed May 17, 2021).

[114] Richard M. Stallman. *Did You Say "Intellectual Property"? It's a Seductive Mirage*. URL: https://www.gnu.org/philosophy/not-ipr.en.html. (Accessed May 17, 2021).

[115] *Stencil (Design System)*. URL: https://stenciljs.com/. (Accessed May 17, 2021).

[116] *Storybook*. URL: https://storybook.js.org/. (Accessed May 17, 2021).

[117] *Studio Gennai - L'arte dell'acqua*. URL: https://www.studiogennai.it/larte-dellacqua/. (Accessed May 17, 2021).

[118] Lukasz Szczygiel. *Project Component - QueryText*. URL: https://github.com/lukasd2/phrmx-components/tree/main/components/query-text. (Accessed May 17, 2021).

[119] Lukasz Szczygiel. *Project Component - QueryUi*. URL: https://github.com/lukasd2/phrmx-components/tree/main/components/query-ui. (Accessed May 17, 2021).

[120] Lukasz Szczygiel. *Project Component - TrackEditor*. URL: https://github.com/lukasd2/phrmx-components/tree/main/components/track-editor. (Accessed May 17, 2021).

[121] Lukasz Szczygiel. *Project Component - VideoPreview*. URL: https://github.com/lukasd2/phrmx-

components/tree/main/components/video-preview. (Accessed May 17, 2021).

[122] Lukasz Szczygiel. *Project's components repository*. URL: https://github.com/lukasd2/phrmx-components. (Accessed May 17, 2021).

[123] Microsoft Edge Team. *What to expect in the new Microsoft Edge Insider Channels*. URL: https://blogs.windows.com/msedgedev/2019/04/08/microsoft-edge-preview-channel-details/. (Accessed May 17, 2021).

[124] *The European Patent Convention - Article 52*. URL: https://www.epo.org//law-practice/legal-texts/html/epc/2020/e/ar52.html. (Accessed May 17, 2021).

[125] *The MIT License*. URL: https://opensource.org/licenses/MIT. (Accessed May 17, 2021).

[126] *Tom Kabinet case*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:62018CJ0263&from=EN. (Accessed May 17, 2021).

[127] James Turner. *Open source has a funding problem*. URL: https://stackoverflow.blog/2021/01/07/open-source-has-a-funding-problem/. (Accessed May 17, 2021).

[128] *Vimeo - Video maker*. URL: https://vimeo.com/create. (Accessed May 17, 2021).

[129] *W3C - Custom Elements*. URL: https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements. (Accessed May 17, 2021).

[130] *W3C - Evergreen Standards*. URL: https://www.w3.org/wiki/Evergreen_Standards. (Accessed May 17, 2021).

[131] *W3C - Selectors API*. URL: `https://www.w3.org/TR/2020/SPSD-selectors-api-20201103/`. (Accessed May 17, 2021).

[132] *W3C - Standards and Drafts*. URL: `https://www.w3.org/TR/?status=rec`. (Accessed May 17, 2021).

[133] Steven Weber. *The Success of Open Source*. USA: Harvard University Press, 2004, pp. 153–154. ISBN: 0674012925.

[134] *What is innovation?* URL: `https://www.pwarome.org/2019/03/18/what-is-innovation/`. (Accessed May 17, 2021).

[135] *WHATWG - Shadow Trees*. URL: `https://dom.spec.whatwg.org/#shadow-trees`. (Accessed May 17, 2021).

[136] *WHATWG HTML Standard - Slot*. URL: `https://html.spec.whatwg.org/multipage/scripting.html#the-slot-element`. (Accessed May 17, 2021).

[137] *WHATWG HTML Standard - Template*. URL: `https://html.spec.whatwg.org/multipage/scripting.html#the-template-element`. (Accessed May 17, 2021).

[138] *Wikipedia - DRY (principle)*. URL: `https://en.wikipedia.org/wiki/Don%27t_repeat_yourself`. (Accessed May 17, 2021).

[139] *Wikipedia - Evolutionary prototyping*. URL: `https://en.wikipedia.org/wiki/Software_prototyping#Evolutionary_prototyping`. (Accessed May 17, 2021).

[140] *Wikipedia - KISS (principle)*. URL: `https://en.wikipedia.org/wiki/KISS_principle`. (Accessed May 17, 2021).

[141] *Wikipedia - Separation of concerns*. URL: `https://en.wikipedia.org/wiki/Separation_of_concerns`. (Accessed May 17, 2021).

[142] *Wikipedia - Solid (principles)*. URL: `https://en.wikipedia.org/wiki/SOLID`. (Accessed May 17, 2021).

[143] *Wikipedia - YAGNI (principle)*. URL: `https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it`. (Accessed May 17, 2021).

[144] Jeanette Wing. *Computational Thinking: What and Why?* URL: `https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf`. (Accessed May 17, 2021).

[145] *YouTube studio*. URL: `https://studio.youtube.com`. (Accessed May 17, 2021).