TÉCNICO LISBOA

WikiTime - Collaborative Timelines

Daniel Teles Arrais

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Professor João Nuno de Oliveira e Silva

Examination Committee

Chairperson: Professor António Manuel Raminhos Cordeiro Grilo
Supervisor: Professor João Nuno de Oliveira e Silva
Member of the Committee: Professor José Manuel Emiliano Bidarra de Almeida

May 2017

Acknowledgments

It is with great pride that I am now able to close this chapter of my academic life. It was really an exciting journey, with lots of ups and downs, which would not be possible without all the people that surrounded me through the years and specially during the time I was working on this project.

Foremost, a big thank you to Prof. João Nuno de Oliveira e Silva, for his patience, support, and dedication, that has made this work easier. His insight, knowledge, and objective approach helped me to keep on track and to simplify what sometimes appeared to be difficult, while leaving me with enough room to make my own choices and learn from them.

To my parents, I cannot ever really thank them enough, for their friendship, encouragement, and caring over all these years, and mostly for always being there for me through thick and thin. Also, thanks to Simão, my younger brother, who has the most cynic and intelligent sense of humour and with whom I always share good moments.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there during the good and bad times in my life. In particular, I would like to dedicate a major part of this work to Maria João Leitão. She understood me when I needed to be understood and always tried to push me further. Another huge thanks to my friends, Rui André, who was always present throughout this whole process and had to deal with me in some of my stressful moments, and Filipe Monteiro, for all the good discussions and ideas that we shared. Finally, I also would like to mention Marta Lima and Filipe Teixeira, for being loyal companions in the last few months of this project.

To each and every one of you, my most sincere thanks!

Abstract

The organisation and visualisation of chronological data in Web pages needs to be properly handled to provide the reader with a clear understanding of the examined topic. Despite the existence of modern chronographic tools such as web-timelines, the majority of history-related content on the Web are merely lists, or tables, in which each entry is granted just enough space to display its information. Furthermore, even though the distinct contents are usually related, these links are typically not explored and remain unused, hence limiting the user's access to a fluid experience when studying a particular subject.

Currently, some Web-based technologies for creating visually engaging timelines have already been developed, but these still face some problems regarding the inability to edit or reuse previously created contents, and the non-existence of a mechanism that encourages collaboration among users. The work herein presented defines the architecture of a collaborative infrastructure, entitled *WikiTime*, that addresses these referred issues. The developed system not only allows for an easy creation and editing of timelines and its events but also grants the user with the possibility to clone any existing information. Moreover, a new concept of sub-timelines is introduced, particularly essential to the definition of the relationships between the multiple topics.

When compared with the existing methods for chronological presentation, *WikiTime* showed exceptional results, providing the user with a dynamic and self-consistent experience. In the context of an organisation, *WikiTime* also allows the integration of scattered information.

Keywords

History, Timelines, Web Application, Collaborative Software, ReSTful API

Resumo

A organização e visualização de informação cronológica na Web necessita de maior atenção por parte dos seus utilizadores de forma a permitir uma compreensão adequada dos tópicos apresentados. Ainda que existam ferramentas cronográficas modernas como as web-timelines, a maioria dos conteúdos são ainda apresentados com recurso a listas, ou tabelas, procurando-se sempre utilizar o mínimo espaço necessário. Adicionalmente, embora os vários tópicos estejam tipicamente relacionados, estas ligações não são exploradas e permanecem inutilizadas, privando o utilizador de uma experiência fluída durante o seu estudo.

Apesar de actualmente se encontrarem disponíveis algumas tecnologias Web para a criação de timelines, estas apresentam ainda certas desvantagens no que diz respeito à impossibilidade de editar ou reutilizar conteúdos previamente criados, e à inexistência de um mecanismo de colaboração. O trabalho aqui apresentado define a arquitectura de um sistema colaborativo, intitulado *WikiTime*, que tem como objectivo resolver estes problemas. Além de permitir a fácil gestão de timelines e seus eventos, o sistema também oferece a possibilidade de reutilizar informação existente. Um novo conceito de sub-timelines é também introduzido, essencial na interligação entre diversos tópicos.

Comparando com outras ferramentas, o software desenvolvido apresenta resultados excepcionais, fornecendo ao utilizador uma experiência dinâmica e auto-consistente. No contexto de uma organização, a aplicação *WikiTime* também permite a integração de informação dispersa.

Palavras Chave

História, Timelines, Aplicação Web, Software Colaborativo, ReSTful API

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface

CDN Content Delivery Network

CERN European Organization for Nuclear Research

CRUD Create, Read, Update, and Delete

CSS Cascading Style Sheets

DBMS Database Management System

DOM Document Object Model

ER Entity-Relationship

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JS JavaScript

JSON JavaScript Object Notation

NPM Node Packaged Modules

OS Operating System

PDF Portable Document Format

RDBMS Relational Database Management System

ReST Representational State Transfer

SQL Structured Query Language

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

UX User Experience

WWW World Wide Web

XML Extensible Markup Language

1

Introduction

Contents

We live in a time where changes are regularly occurring, where evolution is a constant process. People are always so concerned about the present, and even more worried about the future, that anything from the past seems to become irrelevant and outdated. However, history has shown us that the past should not be taken lightly. To better understand the present, sometimes we should look into what has happened and comprehend the factors that have caused change. By analysing past events, we allow ourselves to anticipate what has yet to come, i.e., to understand how society will evolve through time and what consequences should be expected. History, as a discipline, claims to pursue these same objectives and thus, sets its importance in current society's education:

> "*History should be studied because it is essential to individuals and to society, and because it harbors beauty.*"
>
> — *Peter N. Stearns, in [1]*

In an article for the American Historical Association [1], Peter Stearns affirmed that history started being studied at schools as a differentiating factor between educated and uneducated people. However, in his opinion, this fact is quite diminishing regarding the importance of history in education and continues to present a series of reasons for why it should be studied. To summarise, Stearns claims that through the study of history, of how different groups, communities and nations interacted with each other and evolved, we are walking towards a more sensible and responsible society, capable of understanding how different actions or behaviours will affect the lives of its constituting citizens. Moreover, history contributes to building one's moral sense - lessons can be learned not only from those who were responsible for significant achievements but also from the ones who had less suggestive moral values - and for an individual to better acknowledge its place in society, by forming an identity based on the study of how past ethnic groups or institutions settled and evolved.

History was first established as an academic discipline around the year of 1870 [2] but, at that time, education was a privilege only accessible to a small group of people – a wealthier one. Through the years, new technologies have emerged, and the arrival of the World Wide Web (WWW) has opened the doors of history to people who never had access to it before [3]. In a few steps, historians, or simply history enthusiasts, can share their perspectives and experiences with practically anyone, using for example blogs, forums, or a personal Web page. Nonetheless, the organisation and presentation of history and past events in Web pages still face some problems that need to be addressed accurately to allow the proper comprehension of how history evolves and connects different subjects and stories.

1.1 Motivation

When the WWW was invented, in 1989, by Tim Berners-Lee, at the European Organization for Nuclear Research (CERN), the goal was to provide a simple, easy-to-use platform for communicating and sharing information between universities and institutes across the world [4]. At first, the project was only available to a small community, but Tim soon realised the potential behind his idea and, with permission from CERN, he managed to place the software under a public license, allowing people from all over the world to use it freely [5], so long as they had access to the Internet. From this moment on, the dissemination of information on the most varied themes was becoming a trend, and history was also taking its part on this movement.

The WWW is now a huge source of information, one that has challenged historians to rethink not only how to perform research, but mostly how to present and teach about history. In fact, as stated by Cohen and Rosenzweig [3], it displays some clear advantages over the traditional forms for disclosing history. To begin with, historians can not only present their knowledge to a larger audience but also obtain data from different sources around the world, since current online networks are capable of reaching a wide number of people almost instantly, and also more quickly and cheaply. Secondly, by providing the capacity to store large amounts of data of the most varied formats, it is possible to display the desired content in a more enthusiastic way, through the combination of videos, images, or sounds, along with the traditional plain text. Moreover, the Web offers the possibility to search automatically through vast quantities of text or media content, and to link these with other related information, allowing users to move seamlessly through data in multiple different ways.

Over the years, the aforementioned advantages have been put to practice by history researchers. The possibility to create hypermedia documents containing text, media, and references (or links) to other Web pages, has allowed historians to successfully transition from the paper to the digital versions of different documents, such as books or encyclopaedias. However, while these techniques may be sufficient for the display of data included in the referred records, the presentation of chronologies imposes certain requirements that have not been efficiently handled. The majority of history related content on the Web, such as biographies or chronicles, lacks the design and functionality required to provide the readers with a richer and more fluid experience, regarding aspects such as interaction, visualisation, and exploration. The chronological data is usually displayed as simple lists or tables, surrounded by blocks of text, making it difficult for Internet users to get the information they are looking for and to feel involved in the subject. We believe that the reason behind this problem lies in the fact that the producers of such websites may not be competent in Web technologies and, as a result, data is directly transcribed and formatted from plain text into HyperText Markup Language (HTML), which may not be the best solution.

The Web is, by nature, disorganised, and while it may seem like an advantage to have multiple sources and formats of information, it is confusing to read or follow a particular topic if the contents are split over several locations. Furthermore, the multiple current data is usually connected and only at a later stage are these links found, exposed, and used, which defeats one of the purposes of studying

history, that is, to not only look for when and where an event has happened, but also which other events are related to it. As an example, consider the biographies of Alfredo Bensaúde [6], Duarte Pacheco [7], and Abreu Faro [8], which despite sharing a common background related to the Instituto Superior Técnico, it is not clearly perceptible, hence restricting the readers in obtaining a complete information on the topic. The relationships between distinct subjects are an important part of representing history, allowing to properly understand the existence and purpose of the multiple events.

The proper visualisation of data is of utmost importance, and an appropriate display tool is one that facilitates the readers in understanding the presented contents, communicating information to the users clearly and efficiently, while also simplifying the process of relating content and recognising patterns. There is a wide variety of tools for data visualisation [9], and each serves a different purpose according to the type of data one wishes to exhibit. Chronology, in its most basic definition, is the arrangement of a series of events over a definite time period, and time itself can be visualised in multiple different ways. Nonetheless, the idea of portraying time as a line is so embedded into peoples' conceptions and notions that using the *timeline* for representing historical data prevails as the correct fitting [10, 11]. Stephen Davis, a Professor of Design Research at the Royal College of Art, supports this same view and states that: "a graphical timeline is assumed to be more informative than a list of dates, however poorly it is designed" [12]. Digital timelines offer a large deal of interactivity and engagement, supporting visually attractive displays of information, and allowing users to move quickly through time, to efficiently analyse and compare different historical periods, and to easily change between closer or wider perspectives.

1.2 Problem Statement

The areas of study in which timeline tools can be applied are countless. Not only can timelines be used to represent historical data, but they can also be used to explain a certain developmental process, as in the fields of biology or psychology, or simply to organise a team project by defining concise milestones and setting the correspondent deadlines. Despite the fact that it exists already software capable of creating rich and elegant timelines [13–16], these are still incomplete and lack some features that we consider to be necessary:

1. <u>Existence of a collaboration mechanism</u>
   Most existing tools do not have support for active sharing of information, or if they do, it is limited to a small number of people, which obliges users to use multiple different tools. Online collaboration allows users to produce content faster (more people working equals to less time spent) and with higher quality (there is an active and continual evaluation from every collaborator, making the final product less prone to errors);

2. <u>Possibility to embed previously created timelines into larger ones</u>
   The history of institutions, nations, or communities, is built based on the people that belonged to it. By combining the various biographies of an organisation's workers, we should be able to represent its timeline while describing any existing connections between the different people.

3. Simple editing of contents
   It is usually easier to review and enhance the created content than it is to produce it in the first place. By allowing people to visualise and edit data with little effort, we are offering them the possibility to make sure that what they have created has a clear focus or idea, while avoiding the existence of any irrelevant material and thus increasing the user's confidence and motivation for continuous improvement;

4. Reusing existing timelines or events
   Users should be able to directly clone and modify what others have created, stimulating them to share different perspectives and knowledge without the burden of having to gather all the information and build everything from the start. Additionally, the reused content is inherently linked with the original one or with other existing copies, which may help us defining complex relationships between different topics;

Since the source code supporting the majority of these applications is typically distributed under public licenses, it would be expected, at this moment, that a system capable of suppressing the referred problems would already be available. However, considering that to perform such task a sizable amount of programming knowledge is required, the development of these applications is hindered by another issue: interest usually lies within people with professional backgrounds in areas of study unrelated to software development, hence people incapable of designing a product that could contain all the required characteristics and features.

Although some advances have been made in creating software that can transform chronological data into vibrant displays of properly organised information, these are still limited and do not allow to interpret the presented historical contents correctly. History needs to be seen as a field of great value, from which lessons can be taken to help us comprehend the present and to be more aware of the future. With the constant evolution of digital technologies, it is necessary that we take the time to think about how we can use these to serve the purposes of history best. By extending the existing software, a tool worthy of a historical presentation will be provided, capable of organising events in time, displaying the complex relationships that compose history, reusing or encapsulating timelines and events, and supporting collaboration mechanisms.

1.3 Objectives

The primary objective of this work is to define and implement a Web-based collaborative infrastructure, named *WikiTime*, for the easy creation and editing of timelines, without users being required to have prior knowledge of any programming concepts. Moreover, the data stored will be openly accessible according to different user defined permission models and the developed system should be efficiently designed to allow the integration with other software applications. To achieve this objective, a set of more specific goals can be defined and described, which will also help, in the end, to evaluate the proposed work:

1. Implement a Web-based application
   The Web Systems' centralised nature makes them easier to install and maintain, considering that once a new software version is installed on the host server, it becomes immediately accessible to anyone with an Internet connection [17]. Moreover [18], Web applications provide higher scalability (new servers can be added as the workload increases) and flexibility (software components can be easily updated and integrated with others);

2. Include support for collaboration
   The Web became a place for everyone, where new forms of collaboration are always being developed, and discussions or debates on all sorts of topics are occurring every minute, which guarantees the sharing of different perspectives, experiences, and expertise. Collaboration results in higher achievements and greater productivity;

3. Provide a rich Graphical User Interface for Timelines editing
   The functionality and usability of a website are key factors for measuring the difference between an excellent and a poor interface [19]. When visiting a website, users should obtain a clear picture of the type of information contained in the Web page, and how they can proceed to explore its contents further. Although a great interface is visually appealing, the real focus when designing a User Interface (UI) should be on the User Experience (UX);

4. Develop a public Web Service Application Programming Interface (API)
   An API is a particular set of practices and specifications that dictate how different software applications can interact with each other, promoting the sharing of data between multiple Web communities [20]. Not only client applications but also other external software programs should be able to use and redistribute the contents created and stored in the database easily.

This work is expected to be concluded by creating a series of timelines and events using data gathered from websites developed by Instituto Superior Técnico, regarding famous personalities that were actively involved in the creation and evolution of the referred institution [6–8, 21].

1.4 Contributions

By actively addressing the objectives presented in Section 1.3, the work herein described gave order to the following main contributions:

- The development of an application that accurately addresses the existing problems of organising and editing chronological data on the Web. The devised solution provides Internet users with a rich, simple-to-use UI to easily create and edit historical information while working on top of a collaborative basis, hence allowing to produce contents faster and with higher quality. Furthermore, the developed software was designed to be highly extensible and customisable, thus allowing the possibility to introduce, in the future, any additional features required;

- The design and development of a Web Service API allowing the full complete integration of *WikiTime*'s services with other external systems. It is undoubtedly true that data is one of the most valuable assets an application can have, and to have an API as the entry point to that data, is a necessary strategy for increasing the value of a software product. Having a robust, scalable services interface which allows resources to be consumed by other developers, is a step towards fostering the creation of an ecosystem around the designed application, thus making it more desirable and valuable.

The cumulative contributions of this work to the software development community have resulted in the programming of an authentication and user management Github repository, using the same technological stack employed in *WikiTime*'s application. As we started to program our application, we became aware of the absence of a full-stack JavaScript (JS) software project that was simple enough for unseasoned software developers to use and would allow for any specific improvements. In light of this situation, we decided that, since we were about to develop these features for our system, we might as well provide the source code for everyone to use. The repository is available, under a public license, at `https://github.com/darrais/staurther`.

Lastly, we would also like to highlight that, upon completing the development of WikiTime's software application, more than 110 events distributed among 17 timelines were created, depicting about 100 years of history regarding Instituto Superior Técnico. Personalities with important roles during the history of the institution were also included, as well as other relevant characters who actively contributed to the technical and scientific development in Portugal. The created contents include images and videos, along with the descriptional text that allows the readers to follow the rationale behind each event, and consequently, each timeline. Our primary goal was to gather all the data included in the websites created by Instituto Superior Técnico [6–8, 21], combine it with additional sources of information, and provide the readers with an interactive and dynamic experience, in which every relation is shown to allow for a fluid, more complete study on each topic.

1.5 Thesis Outline

The work presented in this dissertation is organised into six chapters. Chapter 2 will start by providing an historical background for the development of timelines, followed by an analysis of the currently used methods for the creation of such displays, along with a discussion on the advantages and disadvantages of each technique. In Chapter 3, a few use cases are defined to understand the design requirements and specifications that the application must meet, which will mainly serve as the basis to construct both the supporting data model and the software platform. In Chapter 4, after first introducing the process of architecting and developing a Web application, we describe the implementation details concerning the envisaged approach. Chapter 5 will evaluate the performance of the developed solution by performing a comparison with the existing methods for visualising and organising historical data in the Web. Finally, Chapter 6 concludes the document by addressing the most significant contributions of the proposed system and by discussing any relevant future work.

2

Cartographies of Time

Contents

The conceptual visualisation of time-related data is not an easy exercise. The idea of time is highly subjective and usually difficult to define since it can contain distinctive meanings depending on peoples' perspectives, beliefs, and feelings. The idea behind timelines is to provide an intuitive and comprehensive method for temporal data display, allowing to visualise how multiple different events are organised in time, what the relationships between them are, and which information they contain. Essentially, timelines are an efficient way of telling a (chronological) story.

This chapter will familiarise the reader on the history and evolution of the concept of timelines as a tool for chronographic presentation. In Section 2.1, we present historical background of the creation and evolution of timelines since the ancient times. Section 2.2 provides an explanation for the importance of developing new instruments for temporal visualisation in the modern times. To conclude, Section 2.3 explores the cutting-edge state-of-the-art techniques for constructing timelines, while discussing the advantages and disadvantages of each one.

2.1 The History of Timelines

Our historical understanding of timelines is closely related to the importance we assign to chronology as a subject of study. Although chronologies are part of people's daily lives, one is usually interested only in the mere fact that these work correctly and does not care about knowing when and how these instruments were developed [10, 11]. Nowadays, using timelines for displaying and locating multiple events in time seems like an acquired knowledge, but this has not always been the case and, although being such a familiar tool, timelines are still a relatively recent innovation, dating back to just 250 years. The current timeline has a history of its own, and it is today at our disposal thanks to the achievements of early chronologers who had to face multiple distinct conceptual problems.

2.1.1 The Ancient Time Tables

Chronology was once a study that gathered significant attention from historians, and the first pieces of evidence can be found in the ancient times when records of medieval lists, denominated as *Annals*, started to appear [11]. These tools, completely devoid of any narrative text, consisted of long vertical panels with multiple dates and the corresponding events, such that all the described entries were displayed with equal priority, having no distinction whatsoever between natural or human acts, and lacking the representation of any causal relationships. An excellent example of these manuscripts is the Annals of St. Gall, where events of the Frankish Kingdoms during the eighth through tenth centuries are displayed in chronological order (Fig. 2.1).

Over the following years, multiple techniques of chronological notation had emerged, but none was able to add actual value to the already existing methods. In the fourth century, Eusebius of Caesarea, bishop and historian, created a sophisticated table structure, referred to as a *Chronicle*, in which the relations between the Jewish, Pagans, and Christians were laid out in parallel columns. This Eusebian model intended to organise and connect historical and religious data from all over the world, allowing for the comparison of distinct eras and individual stories while facilitating the consultation process.

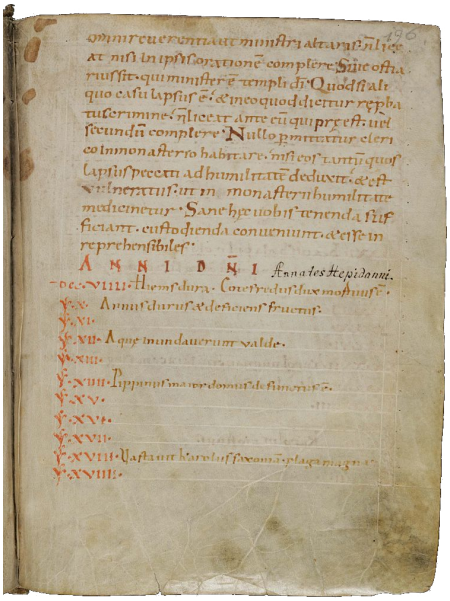

Figure 2.1: Annals of St. Gall, or *Annales Alamannici*, starting with the year 709 [22]

Throughout the Middle Ages and the Renaissance periods, Eusebius's chronological tables remained as the *de facto* choice for visual organisation and presentation. Despite the ongoing efforts to create new forms of data display, the overall simplicity and precision of this visual arrangement suited the demands of scholars significantly well, considering that it was easy to produce and it allowed to organise any type and quantity of data efficiently [11]. In fact, during the fifteenth and sixteenth centuries, the tabular format obtained such renewed attention that modern editions of the Eusebius's *Chronicle* were adapted to the format of the first printed books (Fig. 2.2), and were among the most important reference works of any scholar's collection.

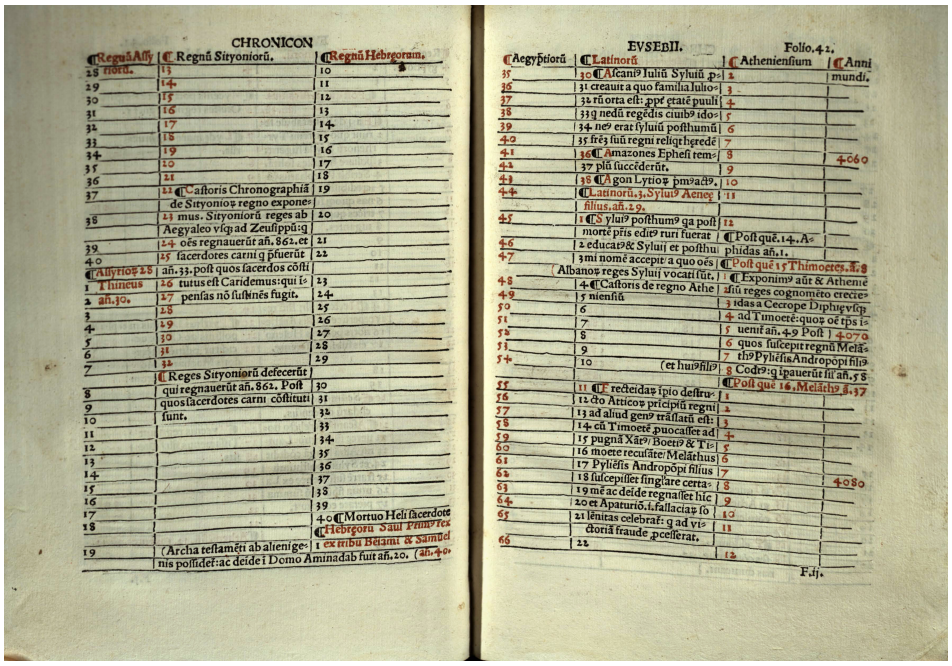

Figure 2.2: Fall of Troy, *Chronicle* of Eusebius, dated from the fifteenth century [23]

Chronology gains an increased interest and importance when time starts being mapped into a spatial dimension. In a pioneering example [10,12], in 1609, Christoph Helwig, a German chronologer, improved the existent table structure by using equal intervals of space for equal intervals of time, and announced a new system for time visualization (Fig. 2.3). Until then, historians had all made the same fundamental mistake, which was trying to represent history using the least amount of space as possible, contracting all the information and packing each entry immediately next to the previous one. With Christoph Helwig's table, the reader was now able to directly grasp in a single glance the temporal interim between two distinct dates.

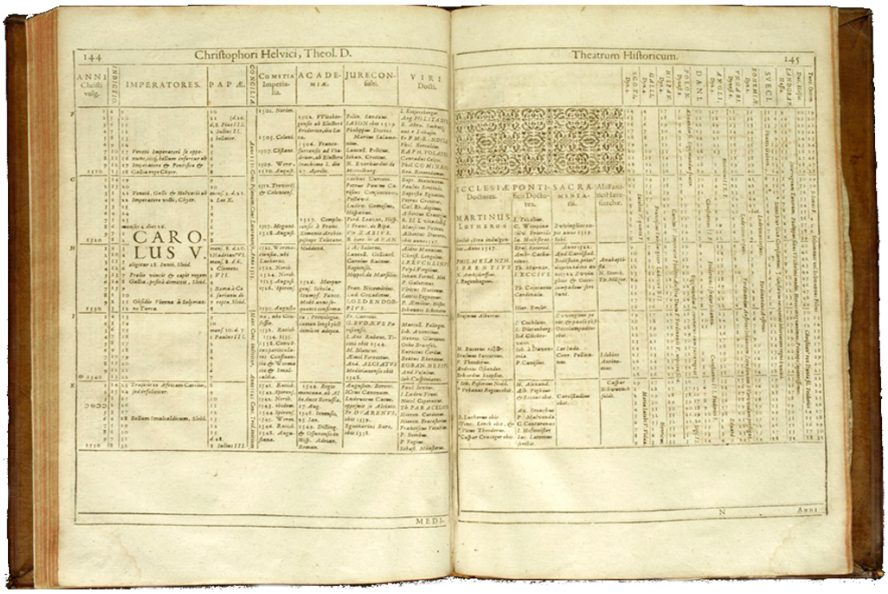

Figure 2.3: Theatrum Historicum, Christoph Helwig, dated from 1609 [24]

Dating back to the seventeenth century, it is possible to understand the emergence of chronology in parallel with geography. Historians, driven by their ambition and critical thinking were continuously searching for an appropriate tool for represent historical data, and the recurrent idea was that chronology and geography were the "two eyes of history" [10, 11]. Regarding the latter, new mapping techniques were developed and introduced, enabling cartographers to update the ancient maps with recently discovered regions and oceans and to display the newly attained knowledge on the Earth's surface. During the same period, chronology went through further innovations when historians, in cooperation with astronomers, began collecting astronomical evidence and started to plot events against lunar and solar eclipses, which could be measured more accurately, rather than against long lasting periods of years. Among the writers of that era, the popular opinion was that the temporal dimension of history was central since it brought rigour and meaning to history's connotations of narrative:

> *"History, indeed is the Body, but Chronologie the Soul of Historical Knowledge;*
> *for History without Chronologie [...] without mentioning the Times in which they were Acted,*
> *is like a Lump or Embryo without articulation, or a Carcass without Life."*
>
> *— Walter Raleigh, cited by Alexander Ross in [11]*

While the visual metaphor fit the concerns of geography perfectly, resulting in the development of new printing and engraving techniques, the representations of time were still stagnated in the chronographic table that had been employed during the entire preceding millennium. Even though chronologists began implementing the same techniques used by cartographers, the direct application of geography in the field of chronology proved to be inadequate. The problem, according to Rosenberg and Grafton [11], was in the approach taken by the many innovators of the seventeenth century, who instead of simplifying and trying to create a visual scheme that could communicate the information clearly and uniformly, were thinking about how to design more complex visual systems.

2.1.2 The Eighteenth Century Graphic Transitions

The creation of a common vocabulary for time-maps, capable of displaying the temporal data in a linear and continuous format rather than a tabular layout, would only occur in the middle of the eighteenth century, around 140 years later of Helwig's attempt to map time arithmetically to space. In 1753, Jacques Barbeu-Dubourg published his *Chronography or Depiction of Time*, which was proven to be one of the most influential works of that period [10]. Unlike other historians, Barbeu-Dubourg not only considered geography a discipline parallel to chronology, like in the "two eyes of history" metaphor, but also acknowledged it as an inspiration for chronology's conceptual basis:

> *"The study of Geography is pleasing, easy, attractive [...]*
> *This is not how it is with Chronology, which is a dry form of study, laborious, unforgiving, offering nothing to the mind but repellent dates, a prodigious accumulation of numbers which burden the memory, are difficult to lodge in the mind and escape thence all to easily."*
> *— Jacques Barbeu-Dubourg, cited by Stephen Boyd Davis et al. in [10]*

Behind Barbeu-Dubourg's rationale was the belief that visual representations of time could and should be designed using a uniform scale, with a constant representation of years, allowing the readers to compare different periods instantly. To prove this theory, he conceived a map that was 16.5 metres long, depicting history since the Creation until his own days. However, since the chart was so long, the posed challenge to display all at once impelled the production of an apparatus called *Chronographie Universelle* (Fig. 2.4), in which the paper could be mounted and viewed one section at a time (roughly 150 years). By using the small crank handles, the reader was able to scroll through history back and forth, hence revealing a continuous timeline [11].

Figure 2.4: Princeton University's copy of Jacques Barbeu-Dubourg's *Chronographie universelle*, from 1753 [25]

The eighteenth century was an era of extreme importance regarding the development of instruments for temporal visualisation. Besides Barbeu-Dubourg's publication, another work of serious relevance was *A Chart of Biography*, developed by Joseph Priestley in 1765, an English educator and theologist. While teaching history at the Warrington Dissenting Academy, Priestley needed to make sense of the lives of various historical figures, which led him to design a diagram that was long enough to record the lifespan of about two thousand individuals (Fig. 2.5).

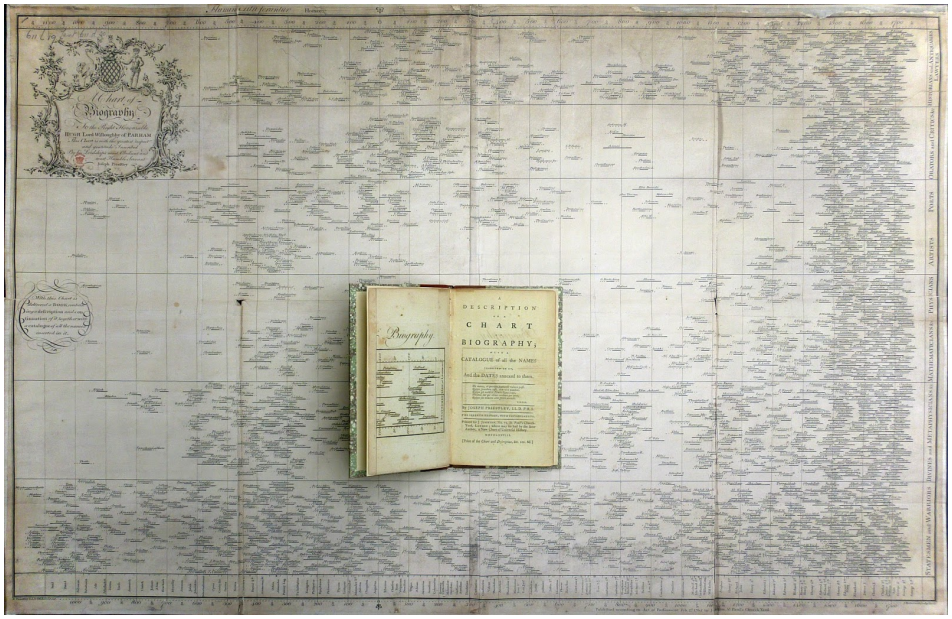

Figure 2.5: Joseph Priestley's *A Chart of Biography*, along with a copy of *A Description of a Chart of Biography*, from 1765 [26]

Even though Priestley cannot be considered the one responsible for the creation of the timeline, what he added to Barbeu-Dubourg's map was the unique concept of using a printed line to represent the lifespan of each individual [11, 27]. Since different amounts of time could be described by using a longer or a shorter line, the readers were now able to obtain clear and precise information about every individual, instead of just a rough estimation. Moreover, considering that historical dates of that time could not be marked accurately, each line was displayed with different levels of uncertainty: a full thick line was drawn for those who he was certain about the dates of birth and death, while dots or a broken line were used to represent uncertainty. By applying these techniques, what Priestley found most remarkable, as he explained in *A Description of A Chart of Biography*, was how this chronological map was able to give a broad view of history. Every life, marked in its appropriate place, was best understood in relation to its time, without requiring any use of words:

> *"[...] you have a line truly representing the situation of that life [...]*
> *and the proportion it bears to the whole period [...].*
> *They are the lines [...] which suggest the ideas;*
> *and this they do immediately, without the intervention of words [...]"*
> *— Joseph Priestley, cited by Rosenberg and Grafton in [11]*

The shift from the tabular layouts of events to clear and precise graphical time-maps was a significant milestone regarding the development of instruments for temporal visualisation. But ever since Barbeu-Dubourg's creation of the timeline, followed by Priestley's unprecedented improvement, that chronology has not received any serious attention and remained essentially in the domain of those who do not have the required knowledge that would allow further developments [11, 28]. Despite the evolution of modern technologies, which would be so deeply appreciated by the eighteenth-century chronographers, current historians have been more preoccupied in recovering the lost history of chronographics rather than in developing new systems for history visualisation.

2.2 The Emergence of Modern Chronographics

As soon as Barbeu-Dubourg and Priestley's inventions were published, the new linear format for designing chronological charts was instantly accepted [11]. Measuring time like distance, and specifically in line-lengths, not only seemed natural but also simplified a chronologer's work when using a solid substrate such as paper to display time-related data. Additionally, Priestley claimed that one of the motifs for using a linear model was the impression of progress it projected, comparable to a river flowing along its course [10]. Even though both Barbeu-Dubourg and Priestley's charts do not contain any figurative elements, the underlying metaphor of the river is always present, providing meaning especially to some of Priestley's decisions. In his informative book, the lives of men are described as being similar to rivers, considering their absence of beginning and end, and their similarity to "so many straws swimming on the surface". Moreover, Priestley evokes the adage *luminis ritu feruntur*, which refers to the maxim provided by Horace to Maecenas[1], to hold his position as the world swirled around him like a river.

2.2.1 Observation and Immersion

According to the different uses described by Priestley, the apparently simple metaphor of the river affords two distinctive interpretations: observation and immersion [12]. While in *A Description of A Chart of Biography* the reader seems to be standing on the riverbank, at a distance, watching the time's flow, in Horace's maxim, the observer is actually in the middle of the river as time passes around him. The first use is related to the timelines' capacity to provide a clear and insightful overview of events, patterns, and personalities, while the second one, even tough less obvious, is related to the timeline's ability to create a sense of immersion in the historical moment. Nonetheless, since Priestley's charts were static, immersion was not a strong feature of his designs, but rather a feature of Barbeu-Duborg's *Chronographie Universelle*, where a sense of history as an immersive experience is the most important:

> "*a moving, living tableau, through which pass in review all the ages of the world [...]*
> *where the rise and fall of Empires are acted out in visible form [...]*"
> — *Jacques Barbeu-Dubourg, cited by Stephen Boyd Davis et al. in [10]*

[1]The *Odes* are a collection of four books, containing lyric poems written by Horace [29].

The historical visualisation charts always tend to favour either the observational or the immersive mode. What distinguishes both modes is the perceived distance of the observer towards the information: if a general perspective on a complete data set is privileged, the observer, who stands outside time, is able to recognise patterns and trends; but if the user approaches the data and studies the context of individuals then, a state of immersion is induced [10]. When applied to historical data, a timeline's primary purpose is to offer these two characteristics, enabling the user to obtain both a general, and a detailed perspective of events, patterns, and personalities.

2.2.2 The Problem in Visualising History

The study of history is not only about examining the existence of certain events in the past but also about identifying and studying the relations of cause and effect that determine these events. History is best understood as a series of interconnected facts and competing narratives that do not mean much when in isolation [30]. The goal of historical analysis is to develop a strong, coherent story that has the power to promote one's understanding of how different actions affect the evolution of multiple societies and what were the factors responsible for change. Once we accept that historical relations are an important part of our comprehension of the world's past, present, and future, we should take into better consideration any device that facilitates the discernment of these relationships.

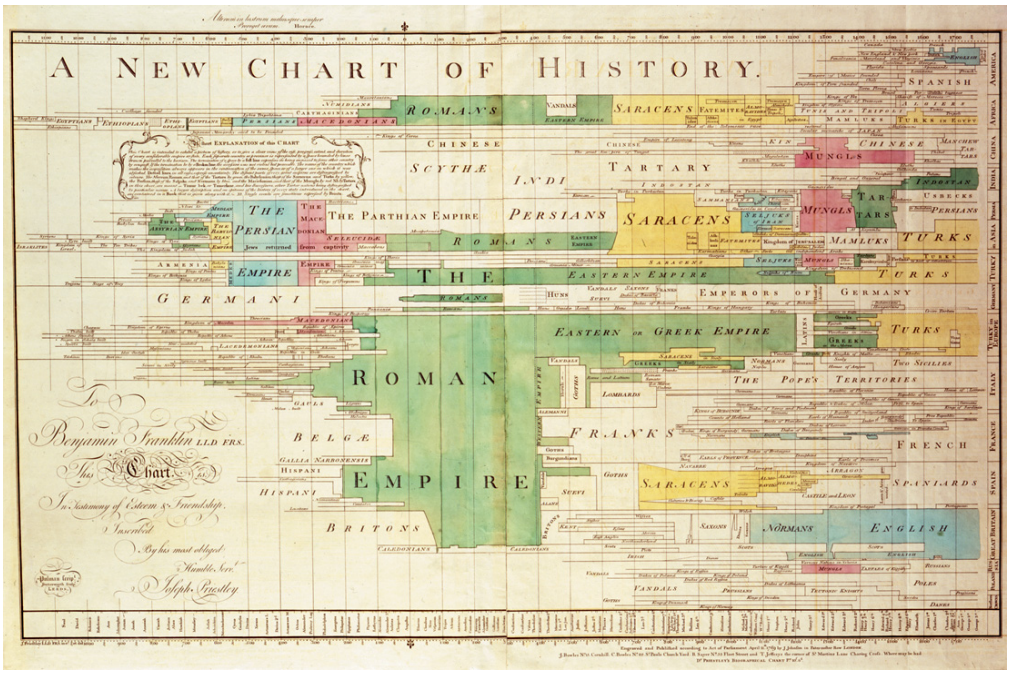

Figure 2.6: *A New Chart of History*, published by Joseph Priestley in 1769 [31]

When both Barbeu-Dubourg and Priestley published their charts, it was already evident that they had profound knowledge about the importance of relationships in the representation of history. The existence of multiple concurrent lives, or empires, through multiple periods, becomes ever so clear when we conveniently analyse and interpret either one of the diagrams. Even so, due to the lack of technologies powerful enough for representing the historical relations accurately, these timelines did not allow readers to understand efficiently if and how the distinct subjects were related to one another.

As an example, consider another one of Priestley's works, entitled *A New Chart of History* (Fig. 2.6), where empires from all over the world are depicted in the same chart. Almost immediately, we can observe the rise and fall of the different empires, and even that some of these coexisted during the same time span, but it is not apparent how the existing empires with parallel histories are interconnected with each other.

With the evolution of digital media, and specifically of the Web technologies, it would be expected that a solution for this problem would already be available, either by using hyperlinks or by employing a more dynamic and interactive form of presentation. However, like in the ancient times, the majority of new chronologies, limited by the early versions of HTML, are really still lists or tables, in which each entry is given just enough space to present its information and is packed immediately between the preceding and succeeding ones [28]. The timeline still occupies a position of little relevance and is rarely given any serious thought. Our comprehension of these tools is still weak, and it is only when all other alternatives have already been used that we think of them as a necessary instrument for the graphic instantiation of history [11, 30].

2.3 Web-based Timeline Software

In recent years, new methods of aggregating and integrating chronological data from multiple sources have been suggested. Methods that elevate the timeline's purpose and consider it as the central organising structure for the graphical display of historical data. By working on the principles of collaboration, and by taking advantage of recent technologies, new modern tools may be able to achieve what the time charts of the past could not: gather and display large amounts of data while presenting all the different existing relationships inherent to the exhibited information. What is important to remember is that, as Rosenberg and Grafton referred [11], "from the beginning, the biggest challenge of the time was not to include more data, but to clarify a historical picture – to offer a form that was intuitive and mnemonic, and that functioned well as a tool of reference".

2.3.1 TimelineJS

TimelineJS [13] is an open-source, Web-based tool for creating and visualising attractive and dynamic timelines. When Zach Wise, in association with the Northwestern University Knight Lab Team, decided to create TimelineJS, he claimed that "the tools that already exist on the Web are almost all either hard on the eyes or hard to use" [32]. Released in March of 2012, it is currently in its third version, being widely used mostly by news organisations from all around the world, including Le Monde newspaper, CNN, and Time Magazine. Among the many features included in TimelineJS, the following ones should be highlighted:

- Possibility to incorporate media content of different types. Users are able to add either image files from several separate sources, or data from other software platforms (e.g. Youtube, Instagram, Spotify, Google Maps, Twitter, or Wikipedia);

- Supports the customisation of elements such as the text font, the initial zoom level of the events' slider, the height and width of the display, or even the language in which the contents are written (around 60 are covered);
- Possibility to aggregate multiple events into a similar category. Events within the same group are organised to be in the same row or adjacent rows, which are labelled with the name of the correspondent category;
- Events can be either punctual or spread across a span of time. There is at least a starting date associated with every event, while an ending date can be defined or not.

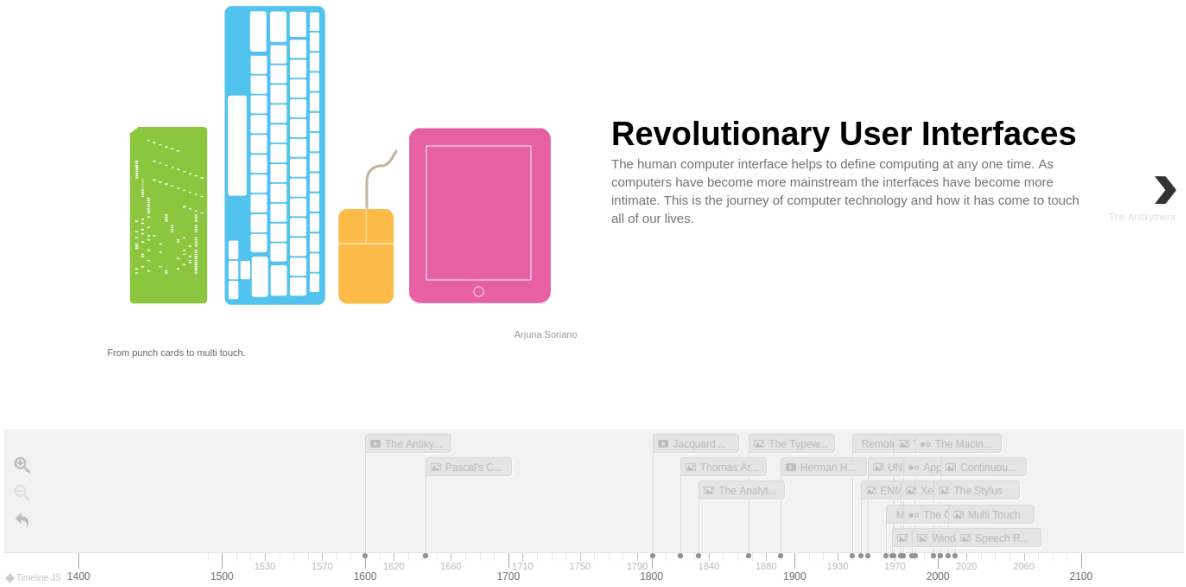

Figure 2.7: Example of a timeline, entitled "Revolutionary User Interfaces", created using TimelineJS [13]

By using the authoring tool available at TimelineJS' website, users can generate the desired timeline objects in four simple steps. The only input required is a properly formatted Google spreadsheet, which is then used to produce the embed code to be pasted into the HTML of a Web page. However, although this method may be sufficient to generate singular and independent timelines, it is still considered to be a relatively primitive technique if one needs to create multiple distinct timelines, interconnected around the same particular topic. In this case, for those who desire to extend TimelineJS' functionalities, Zach Wise and the Knight Lab Team offered the possibility to download and modify the source code supporting the application. Besides accepting a spreadsheet as input, TimelineJS also supports the introduction of contents in JavaScript Object Notation (JSON)[2] format, enabling programmers to design more dynamic and custom installations by moulding the software's functionalities and characteristics into their requirements and goals.

[2]An open-standard data-interchange format that is easy for humans to read and write [33].

2.3.2 Sutori

Initially named HSTRY, it is the developed idea of two business entrepreneurs who, in December 2012, were faced with the difficulty of presenting the history behind the Great Smog of 1952 on Twitter. Although the brand was created a year later, in 2013, it was only in May 2014 that a beta version of the application was conceived, followed by the official release in March 2015. Later on, in November 2016, HSTRY was rebranded to Sutori and remains as such ever since.

Defined as a "free digital learning tool" [14], the principal purpose of Sutori is to offer an instrument, devised mainly for the classroom, which enables teachers and students to create interactive stories using an intuitive collaborative interface. Each timeline can be customised with a description and a header illustration, and different types of event objects can be generated, containing either an image, video, audio, or even a quiz question. Nonetheless, upon further observation, we noted that the existing free package is rather limited since it only allows the creation of at most 200 stories and offers no possibility for collaborative work. Users are required to pay for either the premium package (supports countless stories and collaboration in pairs) or the unlimited package, in order to be able to enjoy such features.

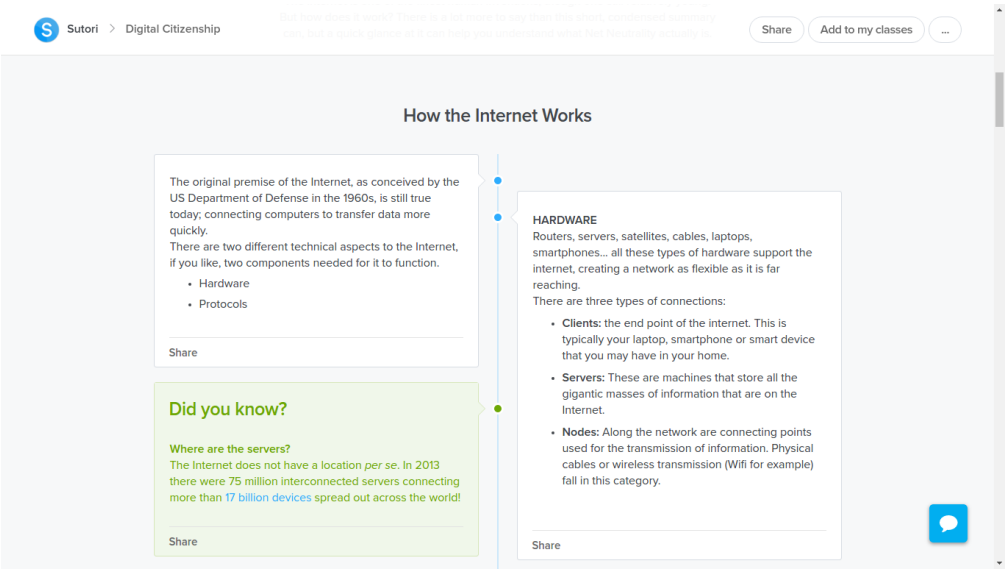

Figure 2.8: Example of a timeline, entitled "Net Neutrality: Why It Matters", created using Sutori [14]

Amid the several visual characteristics it exhibits, the one that draws the most attention is the ordering and arrangement of the events along a vertical axis (Fig. 2.8), contrary to what is commonly expected in the customary representations of time. While it may seem that this property offers no serious issues at all, this is not entirely correct, since the distance between distinct events is imposed by the size of the box containing the information. It is then an arduous task for someone to be able to calculate the time span between two events accurately and to map each event exactly in its position using linear time. Furthermore, because the created event objects are not associated with a date, these referred problems become even more critical. To devise and design a method for properly displaying and organising multiple parallel timelines, embedded into a larger one, would be a conceptual problem of towering difficulty, which would require major changes to the software.

2.3.3 Neatline

Defined as a "light-weight, extensible framework for creating interactive editions of visual materials" [16], Neatline is a project of Scholars' Lab, a team of experienced students and researchers, from the University of Virginia Library, who mainly focus on developing solutions in the fields of digital humanities and geospatial information. It was built as a plugin, with the intention of being used on top of the foundation provided by Omeka [34], which itself is an open-source Web-publishing framework, similar to Wordpress or Drupal, that was created by the Roy Rosenzweig Center for History and New Media, at George Mason University.

Onto the core Omeka's feature set, Neatline adds new functionalities that allow not only to create beautiful and interactive visual objects quickly but also to explicitly represent geospatial information as a collection of records plotted in both time and space (Fig. 2.9). The objective was to offer to mainly students, scholars, and curators new dynamic and engaging techniques for spatial and temporal representation. Nonetheless, even though current versions of Neatline are able to provide the methods for representing any contents that require some visual display, it is still too much focused mostly on maps, putting aside the aspects related to the temporal representation of information.

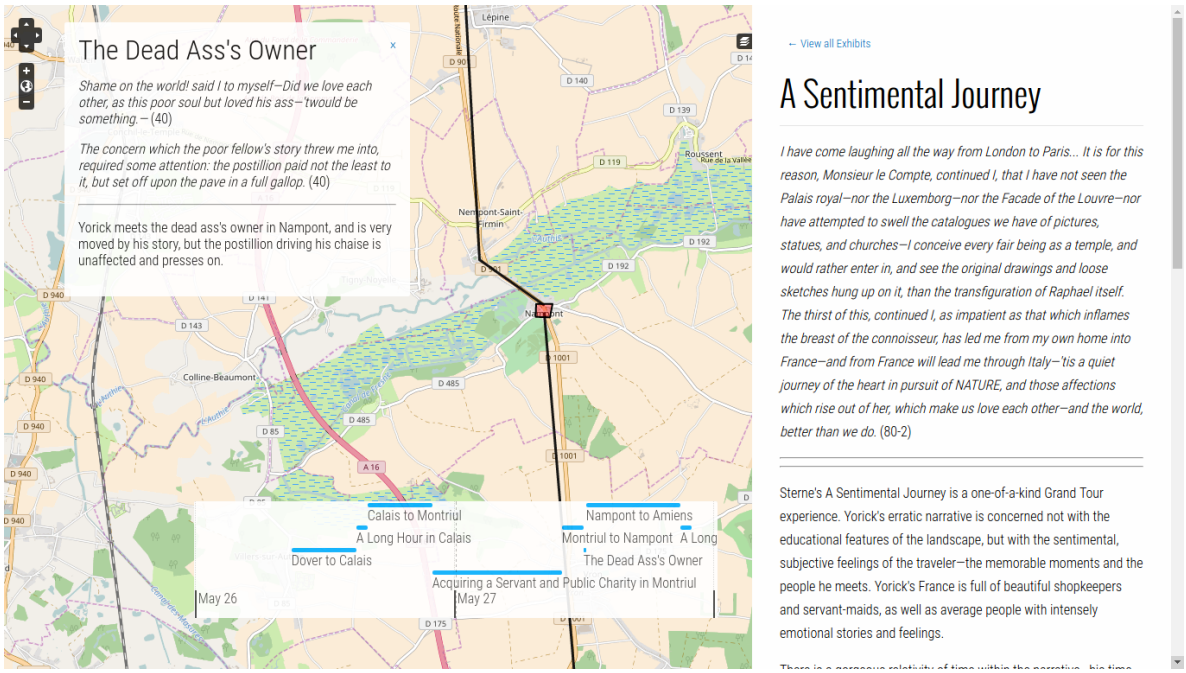

Figure 2.9: Example of a timeline, entitled "A Sentimental Journey", created using Neatline [16]

For establishing and using Neatline, one must first install the technological stack supporting Omeka, which is composed of the famous but rather old-fashioned software bundle entitled LAMP. This acronym refers to a generic software model consisting of four open-source components: **L**inux, **A**pache, **M**ySQL, and **P**HP [35]. After having installed the LAMP stack server, the Neatline plugin must be added and configured. Additionally, it may be necessary to download and incorporate any extra libraries, in order to take full advantage of Neatline's features and functionalities.

2.3.4 Tiki-Toki

Similar to Sutori in practically every aspect, Tiki-Toki [15] is a Web-based application for creating interactive and dynamic timelines which can be shared with anyone on the Internet. It was set up by a company called Webalon, an English Web design and development enterprise, which launched the product in the year 2011. Accordingly, it also makes use of Web-based forms to allow the easy creation and editing of the contents, while supporting the possibility to add data of multiple different types, namely images, text, and videos (Youtube or Vimeo).

Figure 2.10: Example of a 3D timeline, entitled "Tower of London", created using Tiki-Toki [15]

In addition to accommodating the visualisation of timelines in the traditional horizontal format, Tiki-Toki's Web-based application also permits to display the contents in a 3D display (Fig. 2.10). No other software platform provides this feature, which combines both the observational and immersional modes that were so dearly referred by Priestley in the eighteenth century. The problem with this application however, is that, unless users are willing to pay, only one timeline can be created using a free account, with no possibility to perform any collaborative work. Furthermore, even the paid versions still contain a rather small limit on the number of timelines allowed to produce.

2.3.5 Overview of Modern Technologies

After searching for some of the existing solutions, we came to the conclusion that, despite the existence of some particular differences, most of the Web-based applications for creating timelines share a lot of common features, such as the possibility to add different types of media, to customise the appearance of some elements, or to share the created display with anyone on the Internet. Even so, for us to make a more correct and fitting decision, the described applications should be evaluated based on a set of specific parameters:

1. Open-source

   The Web is a place where everyone is accustomed to using different fundamental services, without owners expecting any monetary return. If a distinct platform requires upfront payment, typically there are other alternatives which provide similar functionalities and that are free to use. Nonetheless, the term *open-source* does not refer to the non-requirement of payment to use a particular application, but instead to software programs in which the source code is made available for anyone to employ and modify [36];

2. Documentation

   Having a detailed documentation describing an application and how to use the services exposed, is always a valuable asset to any software product. Not only does it allow to keep track of all the implemented features and characteristics but also clarifies the application's goals and requirements. Ultimately, one of the primary advantages of a proper documentation is that it helps new developers to understand how to use the distributed code [37];

3. Community Support

   Issues and questions always arise, and there are always a few aspects that the documentation does not cover accurately. In this case, support from not only the developers of the application but also from the online community using the tool becomes utterly essential, since it helps to increase customer satisfaction and engagement;

4. Active Development

   It is not uncommon that, from the moment a particular software is released, no further attention is given to it. When an application is maintained under active development, with new updates and versions being frequently launched, this shows concern from the developers for improving and minimising any errors or issues that may exist in the code, hence enhancing the overall performance of the product;

5. Integration

   In software engineering, system integration is related to the process of combining multiple smaller components into an integrated whole, ensuring that these subsystems are capable of correctly functioning when linked [38]. What we are evaluating in this parameter is the capability of an application to be combined with others.

Considering the first parameter of evaluation, both Sutori and Tiki-Toki can be immediately discarded. These systems were developed with the intention of functioning mainly as stand-alone applications, hence offering no possibility to download and reuse the source code supporting the software. On the other hand, TimelineJS and Neatline are classified as libraries, or frameworks, created using Web-based technologies to allow the simple integration of these technologies into larger systems and, consequently, develop more complex software products. In this case, aspects such as the quality of the documentation or the reliability of the community support take greater significance, as well as the fact if the referred project is under active development or not.

Table 2.1 contains a subjective evaluation of both TimelineJS, and Neatline, regarding all the parameters described except the first. Considering that Neatline was designed as a built-in plugin for Omeka, the decision of using this library would also imply the development of our Web application using the technological stack employed by the referred publishing framework. As such, we feel that this obligation would limit the potential of developing a more attractive and dynamic software since, at some point, we would be pushed towards using a set of technologies with which we are not comfortable. Moreover, besides the fact that the official documentation lacks the fundamental information required to understand how to use this library efficiently, the existing online community still counts with a small number of supporters. The learning curve when using these tools for the first time is usually steep and, in the event of any serious obstacle, we would be entirely on our own when trying to solve the problem.

Table 2.1: Summary evaluation of some Web-based Timeline Softwares

	Documentation	**Community Support**	**Active Development (last release)**	**Integration**
TimelineJS	The technical documentation available is sufficient to understand how to use the software	At the moment, the existing online community is already of considerable size, thus offering substantial support	12-01-2017	Simple library that can be easily integrated using Web-based technologies
Neatline	Minimalist and confusing, lacking the information required to understand the fundamentals of the system	Despite the existence of some tutorials, created by the online community, its support still falls short	07-12-2015	Built as a collection of modular plugins for Omeka

Despite the existence of other Web-based timeline software tools, such as Timeglider [39], SIMILE Timeline [40], or Timesheet.js [41], we decided to describe in detail only the four presented ones, mostly because of how these instruments are distinctively organised and how they apply different techniques for creating and visualising the input chronological data. Additionally, other reasons that led us to abandon, at first, any of the other existing programs, were mainly related to the fact that these were still rather immature, requiring the design of more appealing visual displays and the implementation of more powerful features. In comparison with the numerous Web-based libraries for creating timelines, TimelineJS proved to be the one with the most potential to be used and modified according to our application's requirements and goals. In fact, when evaluating this tool based on the presented parameters, we verified that there are not any significant downsides associated with the library: it contains an impressive set of features, displays an attractive and interactive interface, and allows the simple integration with other components.

3

Designing a Prototype

Contents

There is more to software engineering than just writing the required code to make an application work as expected. Logical thinking and design also take part in this long-lasting process that usually begins with the analysis of the project's requirements and functionalities which then give order to the conception of the data structure that serves as the basis for sustained incremental development. In other words, software design is the process of defining and describing the set of constraints that allow us to guide the implementation of an application while ensuring that the system remains flexible enough to undergo any necessary modifications [42].

Different approaches may be carried out at this stage of the software development cycle, each producing different results when tackling a particular issue. While in an analytic approach, the components of the system are isolated and studied in detail, in a systematic one, what matters are the multiple distinct interactions that occur between the components and the functionalities of the system as a whole [43]. In our case, we consider the second approach to be more suitable, mainly because the resulting system will probably be composed of a vast diversity of elements linked together by strong interactions.

This chapter aims to define the functionalities and specifications involved in the characterization of the designed data model supporting *WikiTime*'s application. In Section 3.1 we will perform the software requirements analysis by describing the primary needs and conditions that our application must meet. Section 3.2 contains information regarding the process of data design, that is, a description of the steps required to produce the presented data model schema.

3.1 Requirements Analysis

Software requirements analysis is the first and one of the main components in software design. It is at this stage that users and engineers decide on the key system's specifications and functionalities, which are then used to guide the design of the data model, the definition of the system's architecture, the implementation of the application, and the final evaluation of the product [44]. These should be clear and consistent, and also measurable and testable, providing a sufficient level of detail for the system to be efficiently designed, implemented, and improved.

3.1.1 Product Functions

When entering the home page of the developed Web application, the users will be faced with a list of the timelines stored in the system, together with a search input box that allows limiting the number of results based on the input criteria. The entered value will be matched with both the timeline's headline as well as with the tags associated with each timeline. Moreover, besides being able to search and view the displayed timelines, the most basic operation one will be able to perform is registering an account on *WikiTime*'s application and, after completing the registration process, log in. Authentication should be possible either via a local account (username and password) or via integration with a social account, like for example Google.

3.1.1.1 Events

Upon careful observation of TimelineJS' documentation about how the input JSON data is structured [45], we denoted that events are mainly defined by containing a set of properties regarding their information (headline and description), the associated media object (image or any of the supported integrations [46]), and the starting and ending dates (identified by year, month, day, hour, and minute). Table 3.1 contains a description of all the required functionalities to be implemented within our software application regarding events, that are essentially related to creating and updating the referred object structure.

Table 3.1: Overview of the required functionalities regarding events

	Functionality		Description
1	Create		Create a new event's object
2	View		Visualize an existing event
3	Reuse		Create a copy of the event's object
4	Edit	Media	Associate a media file with the event
5		Information	Modify the event's headline and description
6		Dates	Update the event's starting and ending date
7		Domain	Change the event's domain
8	Delete		Delete the event

Like in TimelineJS, events should be either punctual or spread across a span of time. While the starting date should always be required to be specified (at least the year), it should be up to the user's choice to decide if the ending date is defined or not. Moreover, when creating an event, the user will be entitled to classify its visibility as either public or private: a public event means that it can be visualised by anyone accessing the application, while a private one should only be exposed to the users responsible for it. Nonetheless, it should also be possible to modify this information at any moment.

3.1.1.2 Timelines

Timelines can be described as a set of events grouped together and arranged over a definite time period. As such, it becomes evident that editing a particular timeline means editing the events contained in the referred timeline. Even so, it should be possible to modify certain aspects of a timeline that are unrelated to the objects it contains, such as its headline and description, the media associated with it, or its domain. Furthermore, access to the timeline, regardless if it is for viewing or editing, should only be allowed upon authorisation through the specification of an access list. Table 3.2 presents an overview of the functionalities that are required to be implemented regarding timelines' objects.

Every time a user tries to add a new event to a timeline, a copy of the event's object shall be created, and it should include a reference to the original. This way, we are allowing events to be updated independently, and since all the copies will share mutual ancestors, we will be able to define the relationships between multiple timelines more accurately. Additionally, there should exist the

Table 3.2: Overview of the required functionalities regarding timelines

	Functionality		Description
9	Create		Create a new timeline's object
10	View		Visualize an existing timeline
11	Reuse		Create a copy of the timeline and its events
12	Edit	Add	Add events and sub-timelines to timeline
13		Remove	Remove events and sub-timelines from timeline
14		Group	Group events and sub-timelines into a specified category
15		Media	Associate a media file with the timeline
16		Tag	Create and remove tags associated with the timeline
17		Information	Modify the timeline's headline and description
18		Domain	Change the timeline's domain
19		Access	Update the timeline's access list
20	Delete		Delete the timeline and its events

possibility to associate any type of tags with each timeline, which can then also be used to relate distinct timelines that have one or more tags in common.

The concept of *sub-timelines* is no more than a timeline embedded into a larger one. These should be displayed as simple events with the differentiating factor being the possibility to interact more closely with these particular objects. Upon clicking, for example, in the headline of a sub-timeline, the user should be redirected to the page containing all the information regarding the specific timeline.

3.1.2 User Profiles

Depending on the context in which they are described, there are several types of users that will interact with the system. In a first instance, it is possible to identify three principal types: *administrators*, *registered users* of the application, and *guests*. Each of these will make different use of the software, and so, each of them has different requirements and will be able to perform separate operations. One important aspect to refer is that, after a user performing the registration process, the account will still need to be evaluated and accepted by an administrator. During this time, the user will be considered as inactive and, therefore, he will still be rather limited in the number of actions allowed to execute. An inactive registered user will only be authorised to perform operations regarding the configuration of his account, like for example, changing the email, changing the password, updating the profile information, and such. For the purpose of this document, this type of users is included in the same category as guests, since the kind of interaction that they will be able to perform regarding timelines and events will be more or less the same.

3.1.2.1 Administrators

Administrators are the type of users who are completely free of any constraints, hence being able to access all aspects of the application and carry out all kinds of operations. In addition to all the described functionalities regarding the creation and editing of timelines or events, administrators will be entitled to execute actions related to the management of all the other users of the application. These operations include modifying the status (inactive or active) and the role (normal or administrator) of a user or deleting a user's account.

Since the application is built upon a collaborative basis, the work of administrators is of extreme importance. In general, administrators are required to manage the entire application, ensuring that there is no incorrect or prejudicial information within it and that registered users are behaving in good faith within the bounds of civility. As such, they will also be allowed to modify and delete every timeline and event created by the users of the application, if these contain any harmful content.

3.1.2.2 Registered Users

Registered users are those who have an account on *WikiTime*'s software application which was already accepted and marked as active by an administrator. Primarily, this type of users will have the authority to create and edit multiple timelines and events, as well as to visualise and reuse other already created objects. However, the latter two operations will always be susceptible to if a particular timeline or event is classified as either public or private: if marked as private, only users included in the object's access list (and administrators) will be entitled to view and reuse it. Public timelines and events, however, will be exposed to everyone and will be totally available for reuse.

The registered users' category may be divided into two distinct types, which are based on the user's role regarding a particular timeline. The one who decides to create a timeline's object is classified as its *owner*, responsible for inviting other users, the *contributors*, to collaborate on the creation and editing of contents. The type of functionalities each registered user will be entitled to perform when working on a timeline will differ. Any modification related to more sensitive information, such as the timeline's headline, description, and domain, shall be the responsibility of the owner, as well as any operations about deleting a timeline or managing its access list. Contributors will then be entitled to add or remove events and sub-timelines, create or delete tags, modify the media file, or group multiple events and sub-timelines into a specified category.

Regarding events, if these belong to a particular timeline, there will be no differentiation between the type of users that are included in the access list, i.e., both the owner and the contributors of a timeline will be entitled to perform the same kind of operations on the events it contains. These include editing the event's information, dates, media, or domain, and deleting the referred object. On the other side, independent events that are not included in a timeline will only have one owner, since there will be no possibility to invite any contributors to work in just one event.

3.1.2.3 Guests

Every user who accesses the application and does not have a registered account is included in the guests' category, which refers to the most simple type of users interacting with the application. Besides being able to visualise the current timelines and events, and to undergo the registration process, there is nothing more that they will be able to perform. Even the visualisation of timelines and events will be limited according to either the particular object's domain value is defined as public or private: only public timelines will be exposed to guest users. On the other hand, registered inactive users will be able to visualise private timelines and events, as long as they are included in the defined access list, but will not be able to perform any editing or management of the displayed contents.

3.1.2.4 Organising the Requirements

Table 3.3 displays a summary of what are the functionalities that each distinct type of user will be able to perform, taking into consideration factors such as the domain of a particular object and the user's role in the timeline or event. The checkmark "✔" and the cross "✖" symbols identify, respectively, a positive and a negative value. For other situations where the resulting allowed functionalities do not depend on the value of the particular variable, both a checkmark and a cross symbol are used. The possible values for the domain variable are either "Public", "Private", or "Any". To better understand the presented table, the following two examples can be considered:

- Administrators will be authorised to execute all kinds of operations, regardless of whether the event or timeline is public or private and of whether the user is an owner or a contributor (functionalities 1 to 20);
- A registered active user will only be allowed to create new timelines or events (functionalities 9), and view or reuse existing ones (functionalities 10 and 11) if these are identified as public, and he is neither an owner nor a contributor of the timeline.

The highlighted cells refer to the examples described. Another important aspect is that the functionalities one and nine do not depend on the object's domain nor the user's role, but rather on if the user has an active account or not. As such, only administrators and registered active users will be entitled to create new timelines or events.

Table 3.3: Overview of the allowed functionalities (from Tables 3.1 and 3.2) for each type of user

User Profile		Object	User's Role		Domain	Allowed Functionalities
			Owner	Contributor		
Administrator		Event	✔ ✖	N.A.	Any	1-8
		Timeline	✔ ✖	✔ ✖	Any	9-20
Registered Active		Event	✖	N.A.	Private	-
					Public	1-3
			✔		Any	1-8
		Timeline	✖	✖	Private	-
					Public	9-11
			✖	✔	Any	9-16
			✔	✖		9-20
Guest	Registered Inactive	Event	✖	N.A.	Private	-
					Public	2
			✔		Any	2
		Timeline	✖	✖	Private	-
					Public	10
			✖	✔	Any	10
			✔	✖		10
	Anonymous	Event	✖	N.A.	Private	-
					Public	2
		Timeline	✖	✖	Private	-
					Public	10

3.2 Data Design

Up until this point, we have been planning and analysing all the fundamental requirements that will guide us during the development process of our software application. Transforming some of these specifications into distinct data entities that compose the database system supporting the project's logical and physical structure represents the next step in the software development cycle. Data design is about identifying the type of information that requires being persistently stored, determining the relationships between the different elements and, ultimately, organise and model this information into a diagram representing all the data structures and processes [47]. A reliable data design increases the maintainability and flexibility of a software product, allowing the database to be easily accessed and modified according to present and future requirements.

3.2.1 Data Model

A data model is a conceptual, visual representation of the data structures, including the distinct elements and the several associations between them, that compose the database supporting the application. The primary focus when designing a data model is to represent the information in such way that it should be easily understood by anyone with knowledge of these concepts, serving as a bridge between the real-world components and their physical representation in a database. Despite being a

time-consuming process, it is indeed essential that one should take the necessary time to design the software's database correctly. Without careful planning, not only will we be more vulnerable to the production of any errors, but it will also be harder to promote any changes to the system to solve these problems [48]. The process of designing the software's data model is comprised of four major steps:

1. Identify Entity Types
   An entity is typically referred to as a collection of objects that share common features and aspects. These are usually recognisable concepts such as people, places, or events;

2. Identify Attributes and Keys
   Each entity type may contain one or more attributes, which are commonly used to describe and define the referred entities. When determining the attributes of an entity, one should also define how a specific instance will be identified, i.e., what its unique key will be;

3. Identity Relationships
   Since entities are usually connected with one another, the relationships' purpose is to describe these associations. Additionally, these are typically classified concerning their cardinality, a measurable quantification of the association between two entities specifying how many instances of one are related to a single instance of another (this value may be either One-to-N or N-to-N);

4. Apply Data Model Techniques
   Upon performing all the previous steps, some patterns can be applied to the identified elements. These techniques were developed to facilitate the design of the data model, by providing solutions that describe common domain practices.

There are two primary methodologies commonly used to design the application's data model: the Entity-Relationship (ER) diagram approach [49], and the Unified Modeling Language (UML) class diagram technique. In this document, we will focus only on the latter since it was the one that we used to model our data structure.

3.2.1.1 Unified Modeling Language

Developed in 1997 by computer scientists James Rumbaugh and Ivar Jacobson, and software engineer Grady Booch, the UML is a standard, general-purpose, modelling language widely used in the field of software engineering [50, 51]. The primary purpose of UML is to offer the development community with a common, standardised design language that allows to model the different aspects of a software project and helps in the visualisation of the system's design. Considering that UML is described as a language rather than a methodology, it is possible to use it across different domains of the software applications, hence increasing one's understanding of the system that is under development.

Among the many existing types of UML diagrams, the one that interests us the most is the *Class Diagram*, typically used for modelling the database processes and structure. This kind of chart allows representing how multiple different entities are related to one another by using elements such as classes and the correspondent associations between them. A class diagram is also more close to a

practical code implementation when using an object-oriented programming language, so much that there are even some instruments which allow the direct translation of a UML class diagram to code that follows the referred paradigms.

3.2.1.2 WikiTime's Data Model

By following the guidelines described previously in Section 3.2.1, and with the help of a UML class diagram, we were able to design the data model supporting *WikiTime*'s software application (Fig. 3.1). The first aspect that needs to be highlighted is the definition of the three primary entities, corresponding to the conceptual components that mainly define our project's functionalities and goals: `user`, `event`, and `timeline`. Each of these entities contains a different set of attributes related to the information required to be stored, according to the specifications and requirements previously defined:

- The user entity allows storing information about any registered client, such as their full name, email, or location. Other information concerning a user's account (e.g. the authentication method, the password, the role, etc.) is also required to be stored for the proper operation of *WikiTime*'s application;
- Although there are some subtle differences between events and timelines, the similarities are much more evident, considering that both contain fields related to their domain (private or public), to the starting and ending dates, or to the media associated with a particular record.

With the exception of the users' entity, both timelines and events are entities that cannot be described on their own, as these must always be associated with one or more users. The three major relationships described in our data model are the following:

- User-Timeline Relationship - There are two types of associations between a user and a timeline. As previously explained in Section 3.1, users can either be owners of a timeline or contributors. Nonetheless, it is important to refer that, for every timeline, only one user can be considered as its owner, but multiple users can be defined as its contributors;
- User-Event Relationship - Every user that is associated with a particular instance of events is deemed to be an owner of that instance. Even so, events can have multiple owners, considering that if an event belongs to a particular timeline, the owner and every contributor of that timeline are also owners of every event included in it;
- Timeline-Event Relationship - Timelines are mainly composed of multiple events grouped together. As such, it is quite self-explanatory the reason why every timeline is associated with multiple events, and every event only belongs to a particular timeline. Even though there may be some situations where the same event may appear to be displayed in multiple distinct timelines, the fact is that every timeline always contains a set unique events, and those that seem to be the same are really just copies.

Other types of relationships are also represented in the data model, which refer to the type of associations between an entity and itself (Timeline-Timeline and Event-Event). The reason why, is related to the possibility of the user reutilizing already existing timelines and events, hence producing several copies of the object at stake: a reference from the parent to the children, and vice-versa, must be defined. The cardinality, as well as the role that each entity represents in the relationship, is displayed next to the corresponding element.

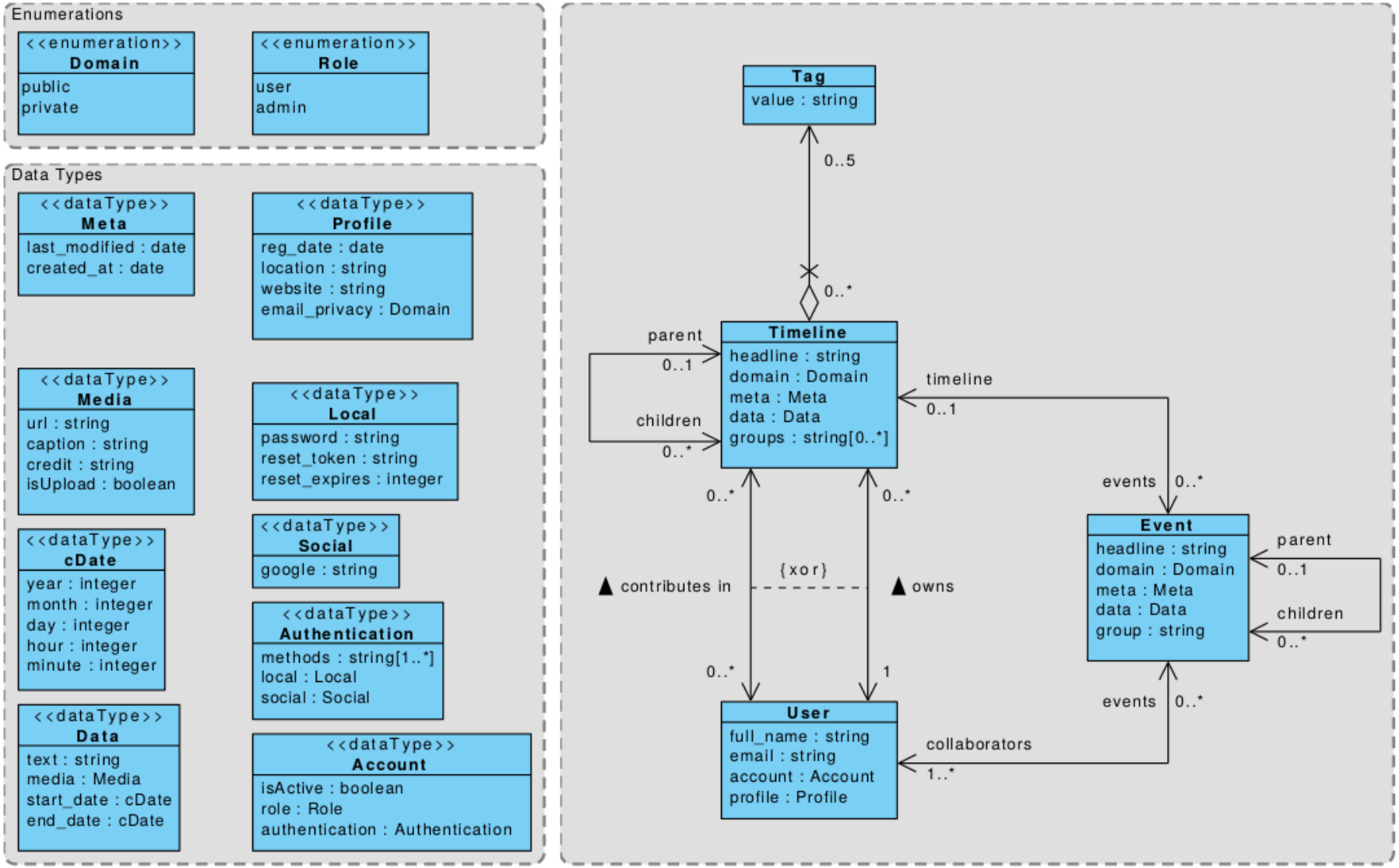

Figure 3.1: *WikiTime*'s UML data model

4

WikiTime's Development

Contents

The process of creating a Web application can be quite demanding and exhausting. With such a large number of frameworks and tools to choose from, and many more being launched practically every week, narrowing down our options can be a frustrating process. When making a decision, there are several factors to consider, such as the tools' ability to adjust to the defined goals or to integrate with other tools, the prior experience of the developer with certain technologies, or the quality of the documentation and of the community support. A careful and thought selection can make a significant impact on the success and efficiency of a software project. The key question is not how to choose the best options, but instead how to choose those that are most appropriate for the application's purpose and goals [52].

For the reader's convenience, the chapter is organised as follows: Section **??** is entirely dedicated to the subject of Web applications, namely the architecture and development methods that are usually employed; then, in Section 4.2, we will discuss about the different types of databases, while presenting the advantages and disadvantages of each one; Section **??** proceeds to detail the aspects of our implementation, referring the technologies and frameworks chosen for the development; finally, Section 4.4 describes the designed and developed Web Service API.

4.1 Web Applications

Software development, particularly for the Web, has come a long way since the first ever website was created and published in 1990 [53]. At that time, the existing internet connections were rather poor and slow, hence limiting the development of more interesting and powerful Internet sites. The first ones ever created had to be basic, mainly composed of static contents and links, having limited interaction with the user. However, those days did not last long, and the trend towards increased user interactivity impelled the Web to evolve, leading to the creation of new improved technologies that would allow users to interact more with Web pages [54, 55]. Along with these developments, a new concept appeared, referring to the new type of more dynamic, interactive, and user-centric websites that began being developed and are still in use today: *Web Applications*.

The definition of a Web application is any software application that can be accessed and executed by the users either through the usage of a Web browser or a specialised user agent [56]. While entirely correct, this definition is slightly confusing, since the term *Website* is usually understood as a similar concept. The difference, in this case, lies mainly in the dynamic and interactive nature of Web applications. A website is typically considered as a collection of web pages with the sole purpose of providing informational content for the reader to consume. A Web application, on the other hand, relies on the existence of interactive elements that enable users to perform actions of any kind [57]. While Gmail, Facebook, or Instagram, are considered common examples of applications, news web pages and blogs belong to the websites' category. Nonetheless, as technology evolves, the line between these two notions becomes thinner, since it is now rare to see an Internet site without any ability for user interaction. A popular manner to define a Web application in terms of a website is to declare it as a dynamic website.

4.1.1 Web or Desktop Applications

The discussion over using a Web or a desktop application is a serious subject that has lingered around the Internet for some time. While some people argue Web applications to be far superior, for numerous reasons, others disagree with this position and favour the opposite one. Nonetheless, the truth is that an argument of this kind is not as simple as it may seem, since both applications are designed to serve different purposes. The most basic definition of a *Desktop Application* is any software application that can be installed and executed on a single computer, not forgetting that many desktop applications also support the possibility of running in a networked environment. In contrast, a Web application is accessed and delivered to multiple specific devices through a remote server located on the Internet [18, 58]. Despite both types being software-based, there are fundamental differences between them, and deciding by either one or the other is a matter of weighing a series of factors and aspects that will define which type best suits the requirements of our project:

1. Maintenance
   The centralised nature of a Web application makes them easier to install, update, and maintain. While for desktop applications, the software is required to be installed and upgraded on every client, when deploying a Web application the developed program needs only to be placed on the host server, allowing users to access it in a single location instantly;

2. Accessability
   Web systems are described as platform-independent, meaning that they can be accessed from any device with an Internet connection, and be executed on different browsers without major noticeable differences. On the other hand, desktop applications are platform-specific, since these are developed to run only on specific settings and a certain Operating System (OS);

3. Security
   The grand number of users and threats that exist in the Internet community turn security into a major concern when developing Web applications. Generally, Internet users are not confident about entering and storing any confidential information on third-party servers. Even for developers, it is usually easier to protect standalone applications from the multiple different vulnerabilities than client-server applications;

4. Performance
   In a networked environment, performance on a single desktop application is usually higher since there are only small amounts of data and information being exchanged with a server. The problem with Web applications is that, for every request made new page layouts and behaviour specifications must be downloaded, thus degrading the overall performance of the system. Additionally, when developing a desktop application, software engineers are enabled to take full advantage of the hardware that is already on the user's machine, which is not the case when producing a Web application. Although we may do so for the server application, the existing limitations on the client-side do not allow to deliver enhanced software;

5. Connectivity
   Internet connectivity is one of the aspects that most affects Web applications. Since these type of software programs rely heavily on connectivity and speed, any issue related to the referred factors (either a weak or a non-existent connection) will degrade the Web applications' performance;

6. Cost
   Hosting a website for a desktop application has minimal or even no costs at all. Typically, the only requirement needed is the design of a few simple pages, with the possibility to purchase and download an executable file that is then installed on the client's machine. On the other hand, Web applications development and maintenance imply higher costs and require a service that provides larger bandwidth and computational power;

7. Scalability
   Web applications provide better scalability than desktop ones. As the workload increases, new servers can easily be added to the system, thus constructing a distributed system architecture that allows to augment the application's overall performance and response capacity.

There will always exist the two perspectives, and software engineers will forever argue about the differences between the two types of applications. What is more important to consider is how any of these may be used in our favour, and how they can limit us from reaching our final goal. In the end, after careful consideration of all the advantages and disadvantages, the result will be the development of either a Web or a desktop application, but in some cases, it may happen that our choices do not fit exactly into one of the specified categories, and the decision becomes just a personal preference.

4.1.2 Architecture

From the concept's creation to the results' presentation, there are a series of steps and phases one must go through when developing a software application. Throughout this whole process, the stage that can truly determine the success of a project, thus being of utmost importance, is the architecture planning. There are multiple approaches to design and plan an application, and different architectures provide distinct characteristics to serve separate objectives. Taking the time to develop a stable, complete, and well thought out architecture is always recommended since this practice can reduce the cost and time of development significantly, ensuring that the final software product is secure, flexible, manageable, and scalable from the beginning. Ultimately, the goal of software architecture is to expose the complete structure of the system while hiding all the implementation details.

Software application architecture is "the process of defining a structured solution that meets all of the technical and operational requirements, while optimising attributes such as performance, security, and manageability" [56]. To ease this process, developers have been, over time, defining and describing a set of high-level patterns and principles, with the primary goal of providing a general, abstract framework for groups of systems that share common features. In other words, this set of clear, well-defined principles, denominated *Architectural Styles*, are meant to mold the application, taking into account its objective and purpose.

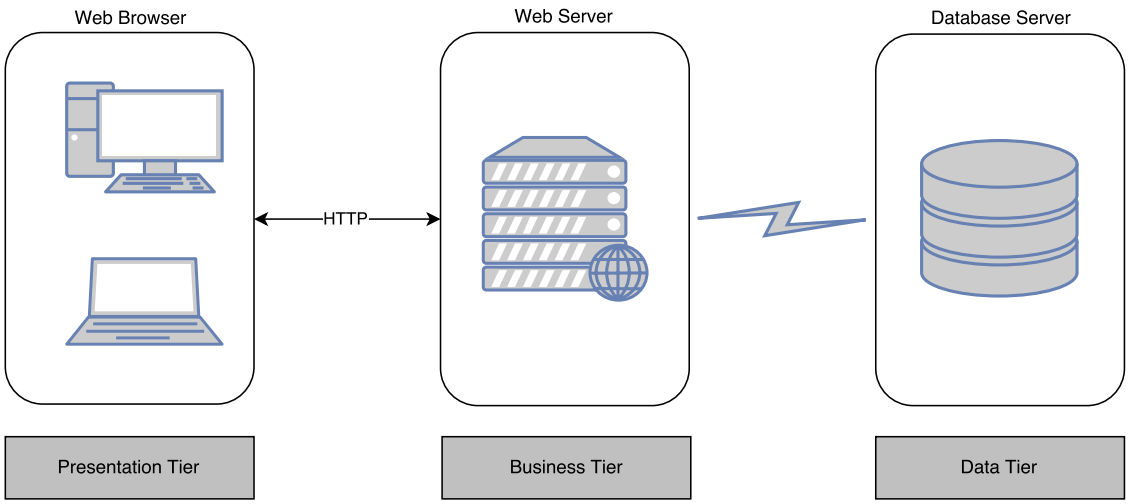

Figure 4.1: General overview of a of a Web application's 3-Tier Architectural Style

Among the existing architectural styles, the one that is more commonly employed when developing a Web application is the *3-Tier Model*, represented in Fig. 4.1. This type of application architecture is characterised by the existence of three distinct components, each generally located on separate physical machines: a Web browser, or client; a Web server; and a database server. In situations where the resources are not vast, the two latter components may be operating on the same machine but as different processes. Despite being in constant communication, the responsibilities and functionalities of a Web application are duly distributed among the three components, such that each is usually unaffected by changes or upgrades in the others, hence providing a clear separation of concerns for the system:

- Presentation Tier - responsible for providing the UI and handling all the user interaction;
- Business Tier - accountable for providing the business logic code which actually allows executing the core functionality of the system;
- Data Tier - responsible for storing data and managing the accesses to it.

The exchange of information between these two elements occurs via a standard protocol denominated HyperText Transfer Protocol (HTTP)[1], and typically is as follows [60]:

1. The user visits the specified Web page and triggers an HTTP request addressed to a specific Uniform Resource Locator (URL);
2. Upon receiving the request, the server performs the requested task, which may require querying the database server, or any external provider, for specific data;
3. The Web server proceeds to generate the results and returns the processed data to the client;
4. After receiving the data, the client interprets it and displays the information to the user.

[1]An application-level protocol for distributed systems that specifies how data is transferred between different parties [59].

Depending on the responsibilities and functionalities one wishes to assign to each component, some specific variants of this architectural style may be considered. Initially, essentially due to browser constraints, the tasks that could be performed on the client's machine were rather limited, and all the processing work was left entirely for the server to handle. But as modern technologies evolved, the overall workload started to be distributed, almost evenly, among the components, with the goal of increasing the reusability, maintainability, and flexibility of the software [61].

4.1.3 Development

Following the planning and design of the architecture, the next stage of any software development project is related to the actual coding of the application. Upon definition of the software's requirements and specifications, it is the developer's responsibility to partition the application into multiple autonomous units, or modules, that can be developed and updated individually. In the case of Web applications, there are at least two major intercommunicating scripts, each serving different purposes and performing distinct assignments: the one that resides on the server (the server-side, or back-end application) and the one that lies on the client (the client-side, or front-end application) [62]. Despite being in constant communication, the coding procedures for each one of the applications are very different, and deciding on what are the browser's script functionalities opposing those of the server is only one of the decisions that need to be addressed when starting a Web software project.

As technologies evolve, a plethora of diverse programming languages, libraries, and frameworks continue to appear, with the sole intention of simplifying the developer's job and speeding up the development process [63]. Since many variations of front and back-end tools may be considered, choosing the right ones for a project is utterly essential. Even though these technologies allow focusing on actually writing the logic of the application, rather than on less important surrounding aspects, there are still some concerns one should be aware of. The implementation details and aspects of these technologies are usually well hidden, which results in numerous coding errors and malicious practices since developers are prevented from learning the underlying logic behind such systems. In addition, programmers commonly become specialised in just a set of well-bundled tools, thus narrowing one's ability, in the future, to develop different projects that require another technological stack.

4.1.3.1 Client-Side Development

Client-side or front-end development is related to the coding of the application that resides on the client's machine, which is parsed and executed by the browser, and is responsible for reacting to any input and managing all the user interaction. The major aspect one has to consider when developing such a software program is the design of a rich and dynamic Graphical User Interface (GUI), capable of providing the user with easy access to an engaging and interactive experience. When entering a Website, users should be able to advance across the Web pages without difficulties, using the application efficiently while obtaining a clear sense of what is its purpose [19,64]. To develop such a script, programmers are entitled to take advantage of the three following Web technologies:

- HyperText Markup Language
  HTML is not exactly a programming language, since it does not perform any logical operations, but rather the standardised markup language that is used to construct and format the contents of a Web page. Typical HTML documents contain page elements, represented by tags, which are rendered by the browser's to create the tree structure of the page – the Document Object Model (DOM)[2].

- Cascading Style Sheets
  As the name suggests, Cascading Style Sheets (CSS) is a style sheet language that, when applied to a document written in a markup language like HTML, allows the developers to define the style and presentation of several Web pages. The primary advantage of providing a CSS stipulation is the fact that features such as colours, fonts, and the positioning applied to the elements can be described in a single file that may serve the multiple HTML pages composing the application. This characteristic offers more flexibility and control of the display specifications and reduces any code redundancy in the designed structural contents.

- JavaScript
  Along with HTML and CSS, JS is the other language of the trio that is present on virtually every Web page. As opposed to the previous ones, JS is considered to be a programming language, which runs mainly on the client-side applications with the essential purpose of providing the animated, interactive, and dynamic effects that exist in the majority of the websites. In conclusion, JS is "a lightweight [...], prototype-based, multi-paradigm, dynamic scripting language, supporting object-oriented, imperative, and declarative styles [...], widely used for controlling Web page behaviour" [66].

Even though the combination of HTML and CSS may be sufficient for designing a Website, the result obtained will be a mere collection of static Web pages, usually for providing the users with informational contents only. The addition of JS functionality is similarly required to allow the creation of interactive elements, if it is the developer's desire to produce more complex and dynamic Web pages. However, although it is presented as highly flexible, JS is also associated with connotations of not being a well-designed language. Writing an application from the start using pure and straightforward JS is a tiresome process that would potentially drift us away from the main objective, i.e., writing the application logic [63]. With the intention of helping developers during this stage, some open-source frameworks and libraries have proliferated, improving JS programming practices and enabling the creation of software programs without having to reinvent and redesign regularly expected solutions. But while some are only meant to provide standard structural elements and style specifications, such as Bootstrap, Foundation, or Semantic UI [67], others like AngularJS, React, Backbone, or EmberJS [68], actually describe different methods for organising client-side JS applications.

[2]An interface standard that defines the logical structure of documents and how these are accessed and manipulated [65].

4.1.3.2 Server-Side Development

Server-side or back-end development is a technique used in the creation of Web applications, which involves coding the software scripts that reside on the Web server. The main objective of these scripts is to produce a customised response for each of the user's requests, by returning the desired data over to the client. Nonetheless, other tasks related to security concerns, such as user authentication and authorization, or to database management, also take part of the server-side application's responsibilities and obligations. Contrary to front-end scripts, back-end ones are executed on the server's machine, enabling developers to produce programs that require heavier processing, and thus additional resources, which could not be operated in a simple Web browser. Moreover, since these are stored on the Web server, rather than being transferred to the client's machine, the written source code remains hidden from the individual users, offering greater security for the Web application's owner [62, 69].

While for the client-side development software engineers are typically limited to a small set of languages and technologies, in the case of back-end scripting, the list is quite vast. Any functional programming language that can be executed on a computer and respond to HTTP requests is also entitled to run a server, widening the range of options available to the developers. Accordingly, for each language, there is at least one corresponding framework, which provides a series of features that help developers in writing the server-side code. Among several possibilities, the most popular ones include Ruby (Ruby on Rails), PHP (Laravel or Symphony), Python (Django or Flask), Java (Spring), C# (ASP.NET Boilerplate), JavaScript (Express or Meteor), and others [70].

Although programmers are not required to develop the application using a server-side framework this practice is strongly advised, since these were created with the sole purpose of simplifying the whole process of writing, maintaining, and scaling a Web application. Common Web development tasks such as routeing the URLs to its appropriate handlers, interacting with databases, supporting user authentication and authorisation, or formatting the output retrieved to the user, are usually well covered by the tools and libraries that these frameworks provide [71]. However, one should take into account that, when choosing such technology, an important aspect that requires serious thought is related to the tool's integration with other technologies employed in the development of the Web application, namely with the client-side's application.

4.2 Databases

An important aspect of every application is related to the storage of data in an organised and standardised manner [48]. Nowadays, applications have to deal with multiple distinct types of information, for example regarding their users, and in a world that is increasingly digitalised, it becomes utterly essential to have practical technological techniques that help storing, retrieving, and manipulating data efficiently. It is in these cases that databases come in handy and gain significant importance: much of the computing power existing in the world is dedicated to using and maintaining databases, which allow to store and update the referred records accurately while providing companies with a system that they can fully utilise and thoroughly trust.

A database is defined as an organised collection of data that is used by software applications to provide efficient retrieval and manipulation of the stored contents [72]. Typically, access to the data is obtained through the usage of a Database Management System (DBMS), a software service that enables users to interact with the designed database and perform the required operations. The organisational structure of a database fundamentally determines how the data is stored and manipulated, and according to the type of model that these management systems support, they can either be classified as relational or non-relational.

4.2.1 Relational Databases

Relational databases were the first type of database systems employed by a vast amount of corporations around the world. These kind of technologies are defined as such because the contents stored are organised based on the relational model of data, proposed by Edgar Codd, an English computer scientist, in 1970 [73]. This model defines an approach for managing and describing data structures according to a set of predicates which are described as follows:

- Data is stored into one or more separate, interconnected tables, with each table representing a different entity;
- Individual records are stored as rows in tables, each row being an instance of a specific entity;
- Each row contains a set of attributes, which are the columns of the table, and is identified by a unique key.

The presented integrity rules allow us to ensure that the data contained in the database always stays valid and accurate. Examples of software systems that maintain relational databases and assure the application of the referred constraints include MySQL, PostgreSQL, Oracle Database, or Microsoft SQL Server, which are generally categorised as Relational Database Management Systems (RDBMSs) [74]. For querying, updating, and maintaing the database, these systems virtually use a special-purpose domain-specific language entitled Structured Query Language (SQL)[3].

[3] A Standard declarative language designed for managing data stored in relational databases [72].

4.2.2 Non-Relational Databases

Developed in the late 2000s, non-relational databases, or just NoSQL ("Not only SQL") databases, were designed to overcome the limitations that the existing relational databases had [75, 76]. The term *NoSQL databases* refers to all the databases which do not employ the relational model applied in RDBMS, hence the data is modelled using different techniques and structures. Despite the existence of numerous distinct NoSQL databases, depending on the type of model they employ these usually fall into one of the four categories:

- Document Databases
  As the name suggests, this type of databases store data in documents, which are usually structured using a simple format similar to JSON. Each record is identified by a unique key and may contain different fields that are paired with a value, such as a string or an integer, an array of values, or even other documents. The most popular examples of document databases are MongoDB and CouchDB;

- Graph Databases
  Among the many existing Graph databases, the most used are Neo4J, Giraph, and HyperGraphDB. These databases are especially useful for organising data as a network of relationships between multiple elements by using a graph structure composed of nodes and edges;

- Key-Value Databases
  Considered as the most simple type of non-relational databases, Key-Value databases store each record as an attribute name, or key, together with the correspondent value. This value, however, is entirely opaque and the querying of documents is only possible by specifying the key. Notable examples of this type of databases include Riak, BerkeleyDB, and Redis;

- Wide-Column Databases
  Wide-Column databases make use of a sparse, distributed, multi-dimensional sorted map, which is highly optimised for queries, to store large amounts of information. Each record can contain a different number of columns, and these can be grouped or nested within one another. Both HBase and Cassandra are great examples of Wide-Column databases.

Each distinct type of non-relational database serves different applications with different purposes. Typically, document databases have the broadest applicability, considering that it is the one model that maps documents directly to the objects used in object-oriented programming languages. Nonetheless, there is one major similarity between every one of these databases, related to the fact that they all offer schema flexibility. As opposed to relational models, where the data schema is required to be defined before any information is stored, non-relational databases allow the insertion of data without first defining this schema. This characteristic provides higher flexibility and offers better performance when querying the database.

4.2.3 Summary Comparison

Even though they are almost 50 years old, relational databases still maintain a position of significant relevance today. The fact that these systems have been able to handle efficiently all the arduous business needs, combined with the existence of a large online community supporting the relational technology, has permitted them to be considered as the top choice for record keeping over the years [76]. However, to meet the requirements of current applications, companies have been increasingly searching for alternatives that provide superior performance and allow for higher flexibility. The NoSQL databases were created for this purpose, as an answer to the problems that relational databases are unable to solve: the appearance of the new types of unstructured, semi-structured, and polymorphic data, and the introduction of the object-oriented paradigms of current programming languages.

Table 4.1 contains a summary of the characteristics and features included in each database category. By presenting developers with the possibility to use dynamic schemas and by allowing for horizontal scaling, there is no doubt that non-relational databases offer improved scalability and flexibility while providing better performance results when reading and updating the stored records. Even so, although seen as an evolution of the relational models, there are still use cases and situations in which a relational database is better suited, namely when using a data model that requires a significant number of relationships between the multiple entities [75]. Both technologies are the best in what they were designed to perform, thus being the engineer's responsibility to examine all the aspects and factors that determine which type of database better suits the project's requirements and specifications.

Table 4.1: Summary comparison between relational and non-relational databases

	Relational	**Non-Relational**
Data Storage	Relational model, where each record is stored as a row within a table, and each column corresponds to a specific attribute	Depends on the type of database: Document, Graph, Key-Value, or Wide-Column
Data Schemas	Fixed, meaning that the data structure and types are defined in advance	Dynamic, such that each record may contain a different structure, and fields can be added and modified on the fly
Scaling	Vertical, meaning that to add capacity more computational power must be added to the server, which is expensive	Horizontal, meaning that it is possible to add capacity by simply adding new, cheap machines, and the database automatically spreads the contents over the multiple servers
Development Strategy	Some relational databases are open-source, others are closed-source	All non-relational databases are open-source

Regarding our software application, when deciding on the type of storage mechanism to be employed, we came to the conclusion that a NoSQL document database would be the best option. First, since TimelineJS accepts JSON objects as input, documents can be easily transferred from one point to another without being required any mid-process transformation. Secondly, considering that both events and timelines have some fields that are entirely optional, such as the media or the ending date, a relational database would most surely contain a significant amount of empty entries for which it would be necessary to allocate the required but unused space. This problem does not exist if one decides to use the dynamic schemas provided by NoSQL databases, hence the total size occupied by the storage system will be significantly lower. Finally, despite being classified as non-relational, the existing NoSQL management systems provide some of the features existing in RDBMS, namely the ability to model different types of relationships either by using document references or embedded documents.

4.3 Software Implementation

Above all, our desire was to create a robust, maintainable, and stable application that could provide the users with a rich, interactive, and engaging experience. An application in which the contents could be displayed efficiently and attractively, that was simple enough for everyone to use.

4.3.1 WikiTime's Technological Stack

When deciding on the technological stack that was going to be employed for the development of our software project, we mainly focused on technologies that would allow us to meet the predefined requirements and specifications. As such, we logically chose to implement the system adopting a set of technologies that have been found to work remarkably well when used together. The MEAN stack is comprised of the following libraries and frameworks:

1. **M**ongoDB:
   MongoDB is an open-source document-oriented database management system that provides high performance and availability while supporting a rich query language syntax for executing all the required operations. Despite being classified as non-relational, MongoDB, like any relational database, offers the possibility to model relationships between the documents, either by using embedded documents or document references. Additionally, since it uses dynamic schemas, the documents, which are represented in a format similar to JSON, can thus be created and updated easily without first having to define its structure [77].

2. **E**xpress:
   Express is a light-weight, open-source JS framework, designed to provide a set of common procedures and methods for developing Web applications. It is the *de facto* standard server framework for Node.js, created with the intention of organising and simplifying the server-side application's code, by providing a thin layer of fundamental features on top of those already offered by Node.js' system engine [78].

3. AngularJS:
   AngularJS is an open-source Web application framework, developed by Google, aimed to simplify the development of client-side applications capable of providing an interactive and dynamic UX. Built on the belief that, despite HTML and CSS are exceptional tools for creating the UI, these still fall short when we try to develop more dynamic Web pages. Hence, AngularJS lets developers extend the traditional HTML vocabulary, improving the overall performance of the application [79].

4. Node.js:
   Node.js is a light-weight, efficient server runtime environment, built on top of Chrome's V8 JS engine, for designing and implementing scalable Web applications. Besides being single-threaded, Node.js uses an event-driven, non-blocking I/O model, hence freeing developers from the pain that is handling concurrency issues when dealing with a lot of connections and requests [80]. Moreover, Node.js contains a package management system, called Node Packaged Modules (NPM), which allows Node.js' supporting community to publish and share a collection of libraries that provide additional features to the technology.

The MEAN software bundle is a full-stack collection of Web-based JS technologies that helps to simplify and accelerate the process of creating a Web application. Despite what people might think, the acronym MEAN is not just a small arrangement of four letters, each corresponding to a different framework, but rather an entirely modern approach to Web software development [81]. By coding with JS throughout both the client and the server's application, developers are entitled to obtain clear performance gains in both the software itself and the productivity of development. Furthermore, given the fact that documents are stored in JSON-like format, using a set of technologies that are JS-based allows the data to be transferred seamlessly from the database all over to the client without having to proceed to its serialization or deserialization.

4.3.2 WikiTime's Architecture

Fig. 4.2 represents the general architecture of WikiTime's Web application. As expected, it consists of the three distinct components that compose the 3-Tier Architectural Model. The Web browser, responsible for displaying the user interface and managing all of its interaction, was developed with the help of the three technologies previously referred in Section 4.1.3.1. While HTML and CSS provide, respectively, the structure and styling of the Web pages, JS is accountable for adding the interactivity. In this type of system, the rendering of the Web pages that compose the UI occurs only within the browser, after the client-side application had tampered the DOM structure.

Upon each user interaction, the client's application performs a HTTP request to the server, using the server's exposed API. There are four distinct types of components included within the Web server, each having its own specific purpose and being accountable for different tasks:

- Operational Management - includes the components that perform input data validation, error handling, and email messaging;

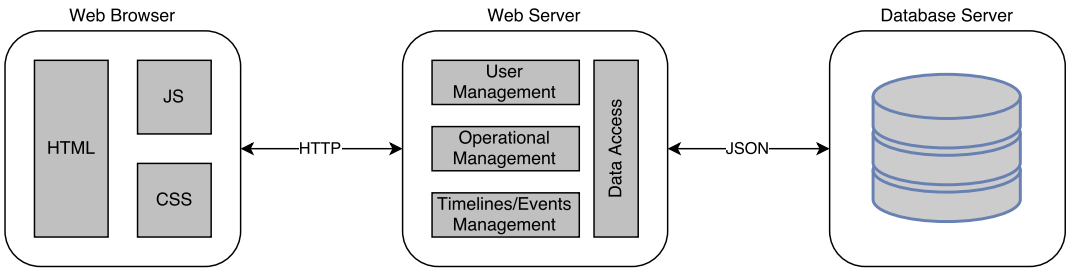

Figure 4.2: *WikiTime*'s Architecture

- User Management - as the name suggests, covers the components that are responsible for authenticating users and controlling accesses to data and executable functionalities. Operations concerning the administration of users' accounts are also included in this category;
- Timelines/Events Management - includes the components that are accountable for the creation and editing of timelines/events, as well as for the automatic construction of the inherent relationships between the multiple existing objects;
- Data Access - includes the components that allow communicating with the database server.

By using this type of architecture, it was possible to implement the client-side application to be completely decoupled from the server-side. The client only needs to have knowledge about the routes designed at the server, and which type of data these return. This decoupling allows software engineers to implement a server-side script that functions mainly as a Web Service API, meaning that it can be consumed not only by the developed client-side application but also from other external systems [82], hence increasing the overall flexibility and manageability of the platform.

4.4 Web Service APIs

When used in the context of Web software development, an API is typically defined as a collection of rules and specifications that describe how multiple different systems can access a specific Web-based software application [20, 83]. By working on the principles of information hiding, Web Service APIs allow us to encapsulate the implementation details of the server's application: while consuming its resources, developers only need to know how and which routes they can use to access the desired information and services.

Besides allowing to reduce the costs and improve the speed of the development process, by increasing the reuse of services, a well-designed API also raises the value of an application, since it provides a general, common method for the product to be integrated with other services, and thus to be used by a larger number of systems. However, designing and developing an API that can be easily understood by its users is usually a morose process that, if not taken seriously, will most likely result in a deviation of the application's final purpose. As developers decide to tackle this challenging task, the primary focus should be on simplifying the design as much as possible, while maintaining the internal consistency of the platform. Ultimately, the programmed service needs be sufficiently stable to handle multiple users, and flexible enough to be easily updated and enhanced.

4.4.1 ReSTful APIs

The term *Representational State Transfer (ReST)* was introduced in 2000 by Roy Fielding, an American computer scientist, and defines an architectural style for designing network-based software systems [84, 85]. When deciding on this type of architectural style, developers automatically become limited to the employment of a Client-Server architecture supporting a stateless, cacheable communications protocol, which is typically HTTP. Despite not being classified as a standard itself, ReSTful services contain a set of features that allow the development of systems that are considered to be:

- Platform-Independent – the server can be running on Unix, Mac, Windows, or any other OS that is able to interpret HTTP requests;
- Language-Independent – the HTTP request can be generated or parsed in any programming language, as long as it is correctly formatted.
- Standards-Based – runs on top of the HTTP protocol.

Web Service APIs that are built using a ReSTful architecture are denominated ReSTful APIs. These services make use of HTTP requests, addressed to specific URLs, to perform any necessary Create, Read, Update, and Delete (CRUD) operations for handling and managing data through proper, standard methods. For each request, the response is basically raw data typically in JSON, HTML, or Extensible Markup Language (XML) format. By using the HTTP protocol as the mediator of the communications, we are entitled to perform operations using the standard HTTP methods. Table 4.2 holds the most commonly used HTTP verbs, as well as the generally associated CRUD op eration.

Table 4.2: Summary of the most commonly used HTTP methods

HTTP Verbs	CRUD	Action
GET	Read	Retrieve whatever information is specified by the Request-URL
POST	Create	Mostly used to create new resources
PUT	Update	Often used to update new resources
DELETE	Delete	Deletes whatever resource is specified by the Request-URL

Depending on whether a particular operation produces or not any side-effects, regarding the update of the data stored, each method is then classified as either safe (or nullipotent) or unsafe. For example, while the GET method is defined as a safe read-only procedure (when executed, the state of the system remains unchanged), the other three presented methods (POST, PUT, and DELETE) are not [59]. However, one must take into account that, as a result of performing a GET request, there is no way for the client to know if the server performed any changes to the data. Even though, the important aspect here is that the user surely did not request the production of any side-effects and, therefore, cannot be held accountable for any updates.

4.4.2 WikiTime's API

One of the main advantages of using a multi-tier architecture is related to its versatility. To meet current business requirements, modern Web applications not only must be able to provide rich and interactive UIs as well as ReSTful Web services to be consumed by external client applications. By providing a clear separation of concerns between the three components, we are ensuring that both the business and data tiers can be shared by both automated and browser clients. The only differences are then related to the nature of the client: while one uses the HTTP protocol to communicate with the server, the other uses a ReSTful API (Fig. 4.3).

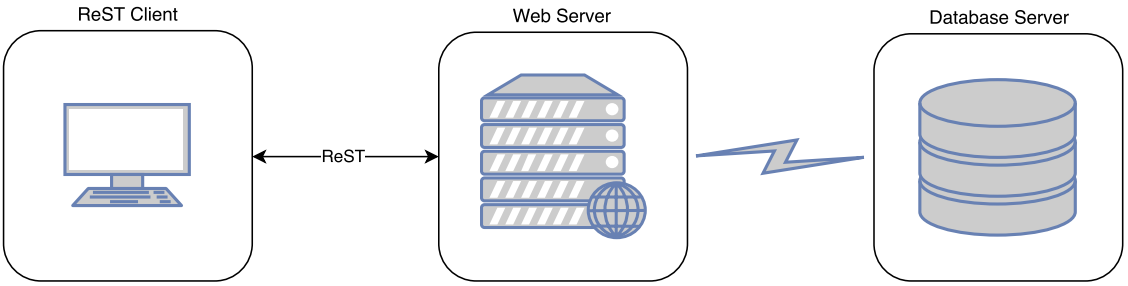

Figure 4.3: 3-Tier Architectural Style of a Web application, with a ReST Client

When designing and developing our Web application, we tried to make sure that the server-side program, functioning mostly as a Web Service API, was flexible enough to serve both the client-side code as well as other external applications that look to integrate the implemented services. The developed API platform provides programmatic access to read and write *WikiTime*'s data, while supporting at least the four HTTP verbs previously mentioned. In addition, when requested to perform an operational task, the response given by the server is structured in JSON format.

Despite the existence of a total of 78 different routes, in Table 4.3 are presented only the most relevant ones, which allow the proper and substantial use of the system's functionalities. To correctly execute any of the desired operations, it is necessary to prepend the base URL of the server, which is given by: `~/api/<resource>`, where `<resource>` can be either users, events, or timelines, according to the values presented in the first column. Another important aspect that can be withdrawn from this is the presence of URL parameters, which can be either `:timeline_id`, `:event_id`, or `:user_id`, that allow us to obtain specific resources by specifying its primary key as an identifier. These identifiers are automatically set by MongoDB when storing the document in the database and are composed of a 12-byte hexadecimal string.

While for every route that uses a safe method, a meaningful response can always be obtained, the same does not occur for every other endpoint that requires employing unsafe verbs. In this case, when the user tries to perform a request without having the necessary credentials to access the resource, an HTTP error is returned. Even for the GET routes, depending on whether the user is allowed to or not, the amount and type of information retrieved may differ (e.g. if not, only data classified as public is presented). Authorisation is granted only to those who are registered at *WikiTime*'s application using an email and password account. Additionally, to validate that the user is actually entitled to access

the requested services, a token must be sent within each request by appending it to the Request-URL: `~?token=<token>`. The token provided, entitled JSON Web Token, is issued with the help of another Express library, JWT.IO [86], and follows the open standard specifications for definining a compact, self-contained method for securily transmitting information between two parties [87]. To obtain this token from the developed Web Service API, one must first perform a GET request to the first URL displayed on the last row of Table 4.3 (`<email>` and `<password>` correspond to the values entered by the user when the registration process was executed).

Table 4.3: Overview of the implemented API endpoints

Resource	URL	HTTP Method	Description
Users	~/search?name=<name>&email=<email>	GET	Searches for a user either by name or by email
	~/all	GET	Lists information about all users
	~/self	GET	Lists information about the user
	~/self/events	GET	Lists information about the events of the user
	~/self/timelines	GET	Lists information about the timelines of the user
	~/:user_id	GET	Lists information about a user
	~/:user_id/events	GET	Lists information about the events of a user
	~/:user_id/timelines	GET	Lists information about the timelines of a user
Events	~/new	POST	Creates a new event
	~/search?headline=<headline>	GET	Searches for an event by headline
	~/all	GET	Lists information about all events
	~/:event_id	GET	Lists information about an event
	~/:event_id/reuse	POST	Creates a copy of an event
	~/:event_id/delete	DELETE	Deletes an event
	~/:event_id/collaborators	GET	Lists information about the collaborators of an event
	~/:event_id/timeline	GET	Lists information about the timeline an event belongs to
	~/:event_id/related	GET	Lists information about the timelines to which an event is related to
Timelines	~/new	POST	Creates a new timeline
	~/search?q=<q>	GET	Searches for a timeline either by headline or by tags
	~/all	GET	Lists information about all timelines
	~/:timeline_id	GET	Lists information about a timeline
	~/:timeline_id/reuse	POST	Creates a copy of a timeline
	~/:timeline_id/delete	DELETE	Deletes a timeline
	~/:timeline_id/collaborators	GET	Lists information about the collaborators of a timeline
	~/:timeline_id/collaborators/:user_id/add	PUT	Adds the specified user to the access list of a timeline
	~/:timeline_id/collaborators/:user_id/remove	PUT	Removes the specified user from the access list of a timeline
	~/:timeline_id/events	GET	Lists information about the events and sub-timelines of a timeline
	~/:timeline_id/events/:event_id/add	PUT	Adds the specified event to a timeline
	~/:timeline_id/events/:event_id/remove	PUT	Removes the specified events from the timeline
	~/:timeline_id/related	GET	Lists information about other related timelines
Account	~/account/authorize?email=<email>&password=password	GET	Obtains the JSON Web token required for authentication

5

Solution's Evaluation

Contents

Evaluation is typically the last phase of the software development cycle. During this stage, developers are responsible for verifying if the implemented solution meets all the required goals and specifications [88]. As the demand for better designed and user-friendly applications increases, the need for a proper evaluation of the software system rises as well. Usually, it is at this stage that programmers find the issues and bugs that were not detected during the development step. As such, an accurate evaluation allows to reduce the costs of maintenance and to improve the confidence one has in the developed product.

In this chapter, we will start by discussing a series of aspects that should be taken into consideration when evaluation the performance of a Web application (Section 5.1). Then, Section 5.2 will proceed to verify if all the essential functionalities were successfully implemented, finalising with a comparison with the existing timeline applications. The last part of this chapter, Section 5.3, is concerned about demonstrating the execution of the developed system using real historical data.

5.1 Performance

In a world where users are gained and lost in a split second, optimisation of a Web application's performance becomes utterly essential. Despite the existence of several fundamental metrics, the ultimate goal of a developer when trying to improve the performance of a Web application is to minimise the perceived delay experienced by the user between the moment he performs an action within the Web site and the moment the results are finally displayed. After all, an application without users has no actual value, and thus it is no wonder that, in the end, what matters is the users' experience when utilising the developed software [89, 90].

Even if a Web page appears to the users as a single entity that is simply accessed by a URL, it is, in fact, an assemblage of multiple Web resources that must be fetched and rendered by the browser to be able to present the desired result on the user's display. As such, multiple distinct techniques can be employed to minimise the amount of time taken during this whole process:

1. Minimising the number of HTTP requests
   For every file that is included in a website, the browser is required to execute a request to the server to download the file. Therefore, it is beneficial to aggregate all multiple separate CSS files, or JS files, into one large file that can be retrieved with a single request;

2. Minifying HTML, CSS, and JS
   The code is usually written in a format that is easier for humans to read and understand. However, this format generally contains an enormous amount of unnecessary characters (e.g. whitespaces, comments, or new line characters) that increase the global size of the file. By removing these, we are actually reducing the file size of the code and thus, the amount of data required to be downloaded, without making any impact on the code's efficiency and performance;

3. Compressing images
   Even though CSS allows the creation of some visual graphics, like buttons for example, some decorative elements require higher design technicalities (e.g. an image or photograph). Nonetheless, these are the type of assets which downloading can cause a major impact on a Web application's performance. It is then recommended to properly format and compress images, or photographs, hence saving on the amount the data that is required to be downloaded by the browser;

4. Deferring parsing of JS
   The browser must download all the associated files of a website before being able to display the page to the users. As such, JS files should be placed at the bottom of the HTML documents (before the closing tags) rather than at the beginning, thus allowing the browser to not having to wait and to render the content to be displayed before JS is fully loaded;

5. Caching
   Caches are high-speed storage mechanisms that support the collection of previous requests, such as images, CSS or JS files, and other data, on either the client or on the server side. The caching of assets that rarely have any modifications made to them helps to reduce the bandwidth and to increase the performance of a Web application. Every time a user makes a request, the server or the browser will verify if the required asset has not been changed and only return a fresh new copy otherwise.

The suggested methods for improving the performance of a Web application are mainly small amendments to the website itself that allow optimising the processing and rendering speed, decreasing the number and amount of data to be requested, and reducing the time taken for transferring any resource minimally. Nonetheless, while these techniques require little effort and almost no additional resources, others are a bit more complicated and difficult to implement.

One of the major draggers to a Web application's performance is related to the physical distance between a user and the server. As an example, if our server is located in Portugal, at Instituto Superior Técnico, and the user is in the United States of America, the transfer of information will require a longer time compared to if the user was also located in Portugal. A solution to this problem would be the use of a Content Delivery Network (CDN), a system of globally distributed network servers whose purpose is to deliver Web contents to a user according to its geographic location [91]. Instead of having a single copy hosted on a single server, multiple copies are dispersed through several servers, so that when a user performs a request, the system will redirect this to a server in the CDN that is closest to the user (Fig. 5.1). The goal is to serve the contents with high availability and performance.

5.2 Implemented Functionalities

In Section 3.1, we described all the primary functionalities to be implemented within the system. As we recall, not only were these related to distinct general aspects of a Web application, such as user authentication and management, but also, in our particular case, to other operations regarding the

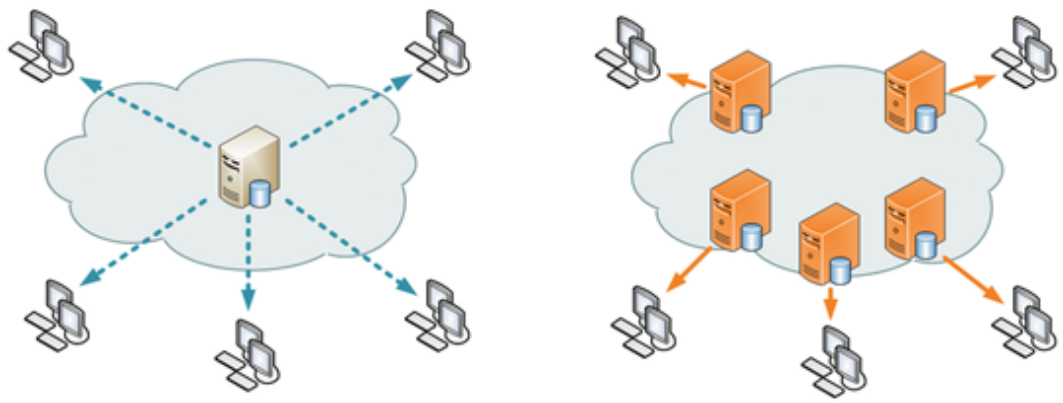

Figure 5.1: Single-Server Distribution compared to a CDN Distribution [90]

creation, visualisation, and editing of timelines and events. In this section, our main objective is to present all the key features included in *WikiTime*'s application as well as the projected interfaces that have allowed us to achieve this goal.

When designing the interface for a software project, a developer's primary concern should be the experience they want to provide to the users of the application. A clean, attractive, well-designed interface can indeed make a significant impact on the product's success since it is part of the software that users are entitled to interact with. As mentioned in Section 4.1.3.1, programmers can only resource to coding with HTML, CSS, and JS for the development of the client-side application. Nonetheless, there is a rather significant amount of frameworks one can use, designed to provide common structural elements and style specifications and to help programmers in the development of an interactive and dynamic interface. Included among the most popular ones are Bootstrap, Foundation, and Semantic UI [67]. But considering that they all offer more or less the same functionalities, provide documentation that is easy to read and understand, and have a large number of supporting users, choosing between one or the other becomes just a matter of personal preference. Therefore, our decision ended up falling on the utilisation of the Semantic UI library mainly because of the simplicity it displays.

5.2.1 Home Page

The basis of any website, regardless of its subject, is on the "Home" page. When entering the application, currently available at `http://146.193.41.162.nip.io/wikitime/`, the users are faced with a list of publicly exposed timelines and a search input bar (Fig. 5.2). Upon entering a value of any kind, this will be matched with either the timeline's headline or with the tags associated with it, hence limiting the number of results that are shown on display.

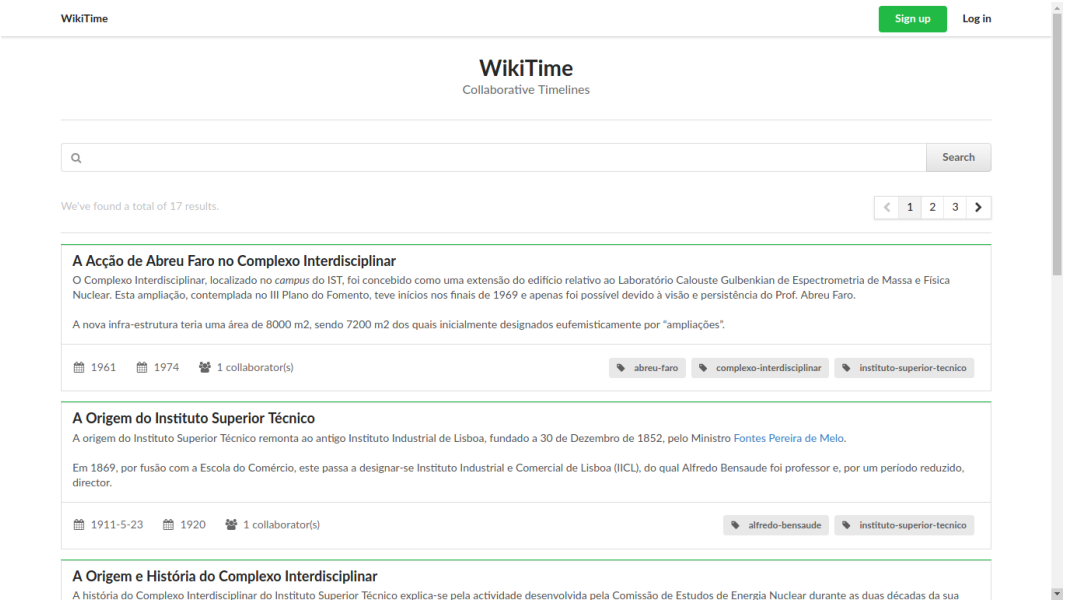

Figure 5.2: *WikiTime*'s "Home" page

At this point, since the user does not have an account, the type of operations he will be entitled to execute is still rather limited (only the visualisation of timelines and events that are classified as having a public domain). To take advantage of all the implemented functionalities, one must first proceed to the next step, that is, performing the registration process and applying for an active account on *WikiTime*'s application. This option is directly available by clicking on the "Sign Up" button shown on the top right of the screen.

5.2.2 Managing Timelines and Events

Once a user has his account marked as active, he is entitled to perform all sorts of operations regarding timelines and events. However, we should always take into consideration if a certain user has the required permissions to execute particular actions, i.e., if he is included in the object's access list or not. Nonetheless, the most basic operation a user can perform, without being required to have any other quality other than being classified as active, is related to the creation of timelines and events.

5.2.2.1 Creating Timelines and Events

The process of creating a timeline or an event is actually pretty similar. As such, it is no wonder that the pages designed for this purpose appear to be almost identical (Fig. 5.3 and Fig. 5.4). The primary difference between one operation or the other lies in the fact that, when creating an event, the user is asked to specify both the event's starting and ending dates, as well as to decide if the event to be created is automatically inserted into a timeline (one in which the user is included in the access list) or just past placed in the user's personal repository. When the event to be created is marked to be automatically inserted into a timeline, a copy of the original is consequently generated and stored in the user's repository. By doing so, we are ensuring that the user always contains a backup of the produced information in case of any occuring issue with the original.

Create new event

Create each event as a part of a larger narrative.

Daniel Arrais

Headline (Required)

A maximum of 80 characters is allowed.

Description

B I

Domain

- Public
  Anyone can see this event. Registered users may reuse it.
- Private
  Only you can view and reuse this event. If in a timeline, all collaborators will have access to it.

Dates

Date

2017 | Month | Day

Hour

Hours | Minutes

Figure 5.3: Screenshot of the "Create New Event" page

Create new timeline

A timeline is a great way to tell a story.

Headline (Required)

A maximum of 80 characters is allowed.

Description

B I

Tags

Search tags or add new ones. Maximum of 5 tags.

Domain

- Public
  Anyone can see this timeline. Registered users may reuse it.
- Private
  You choose who can view and reuse this timeline.

Create timeline

Figure 5.4: Screenshot of the "Create New Timeline" page

Regarding these two tasks, users are at least required to fill in the "headline" field of the illustrated form and, in relation to events, to also enter the year of the starting date object. If none of the values with respect to the ending date is specified, the event is only assumed to be punctual. Moreover, while filling in the form, one also has the possibility to immediately define the object's domain (if not indicated, the default value is public) and to enter a descriptive text associated with the respective object.

5.2.2.2 Editing or Deleting Events

Once an event is created, the user is automatically redirected to the correspondent editing pages. Editing an event basically means updating and modifying the fields that describe and characterise the object. Therefore, to simplify this process, we decided to split the task of editing and deleting an event into multiple distinct views: one centered in updating the media content associated with the event (Fig. 5.5), another related to the modification of its dates (Fig. 5.6), and the last one, represented in Fig. 5.7, which allows changing the event's headline, description, or domain, as well as to delete the referred object.

As specified in Section 3.1.2.2, the aforementioned operations are only possible to be executed by a user that is included in the event's access list. This list can include only one or, if the event is contained in a particular timeline, a group of users, that is the same as the one designated in that specific timeline's access list. Nonetheless, regarding events' objects, there is no actual differentiation between a contributor or an owner and, as such, the ones who have access to editing and deleting the event are all classified as the latter.

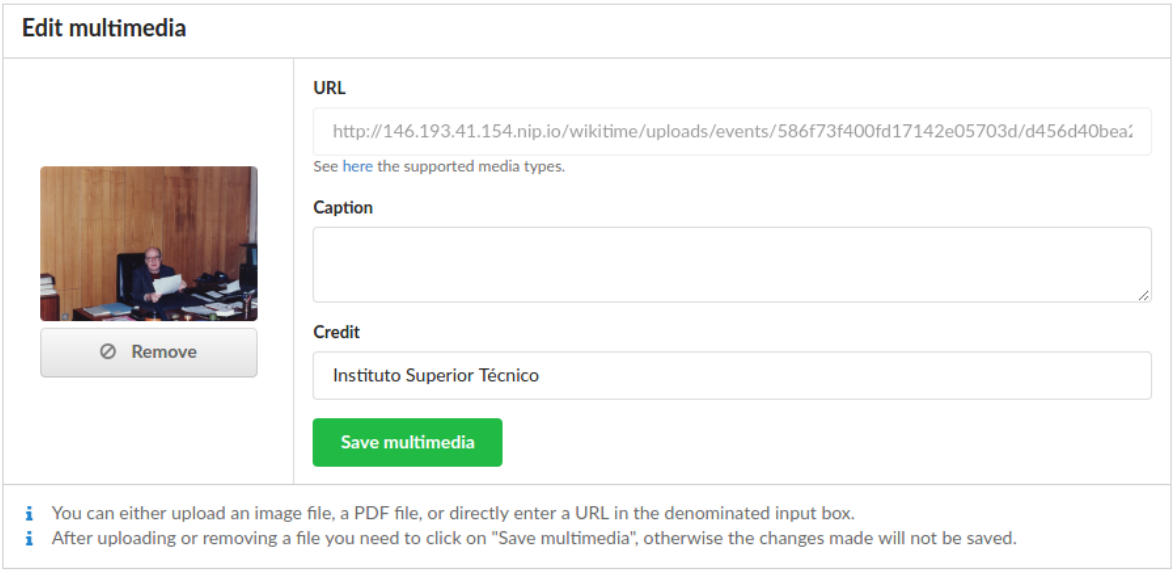

Figure 5.5: Cutout of the event's "Edit Multimedia" page

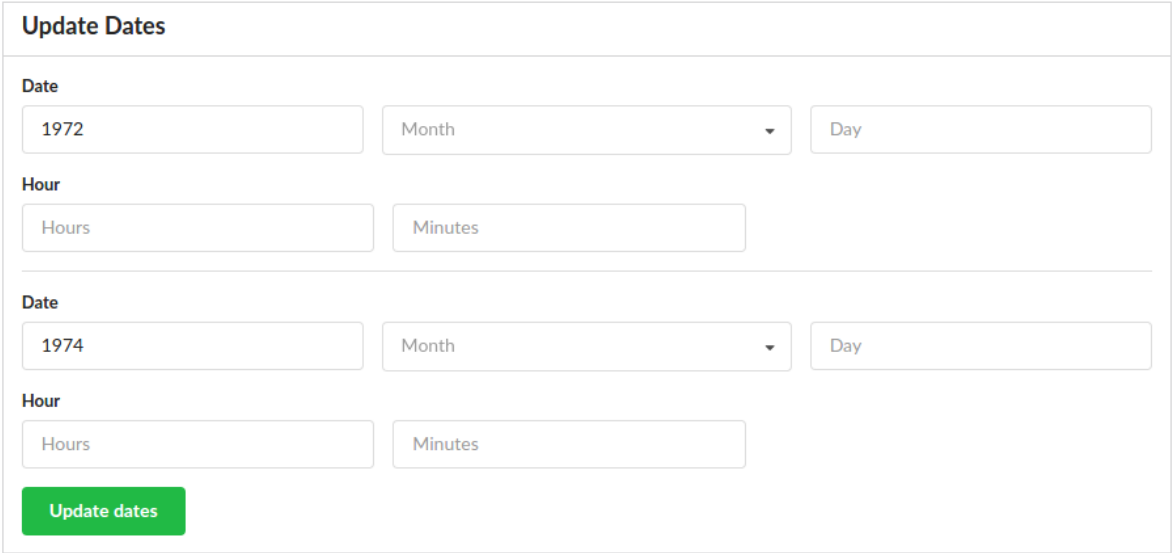

Figure 5.6: Snippet of the event's "Update Dates" page

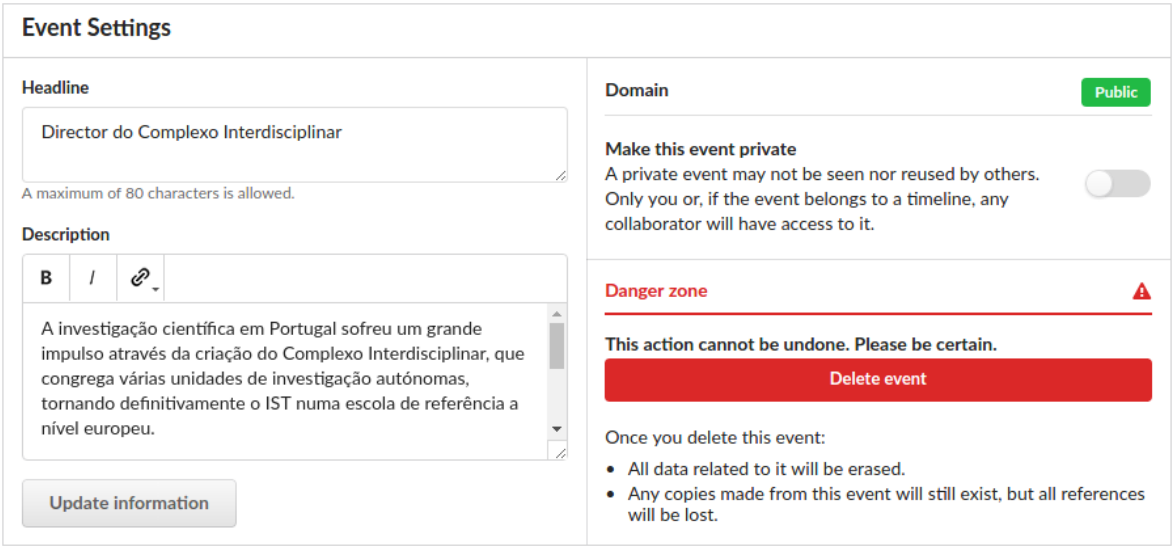

Figure 5.7: Clipping of the "Event Settings" page

5.2.2.3 Editing or Deleting Timelines

As previously mentioned in Section 3.1.1.2, editing a timeline is essentially about managing all the events and sub-timelines it contains. Therefore, the primary page that is displayed to the user when entering the timeline's editing mode is mainly composed of a list with all the events and sub-timelines included in it (Fig. 5.8). In this described view, accessible to both the contributors and the owner of the timeline, the user is entitled to edit, remove, or group into a common category the multiple distinct objects. To add new contents, one must click the "+Add" button exhibited in the upper right corner of the "Manage Contents" page, which will trigger the application to display a different list containing all the events and timelines that are either classified as public or for which the user is included in the access list (Fig. 5.9).

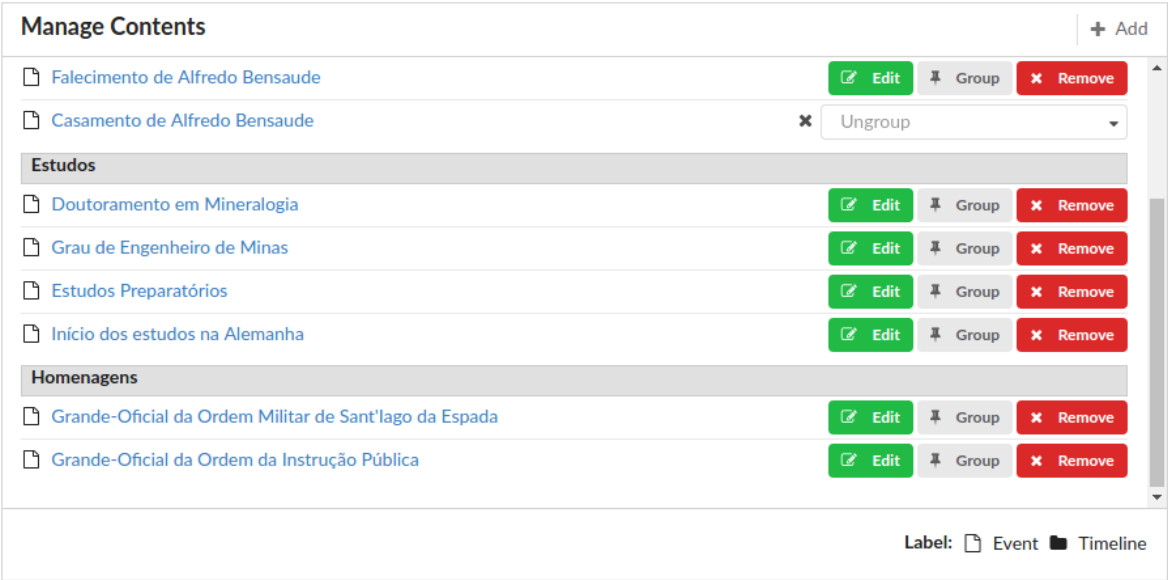

Figure 5.8: Clipping of the timeline's "Manage Contents" page

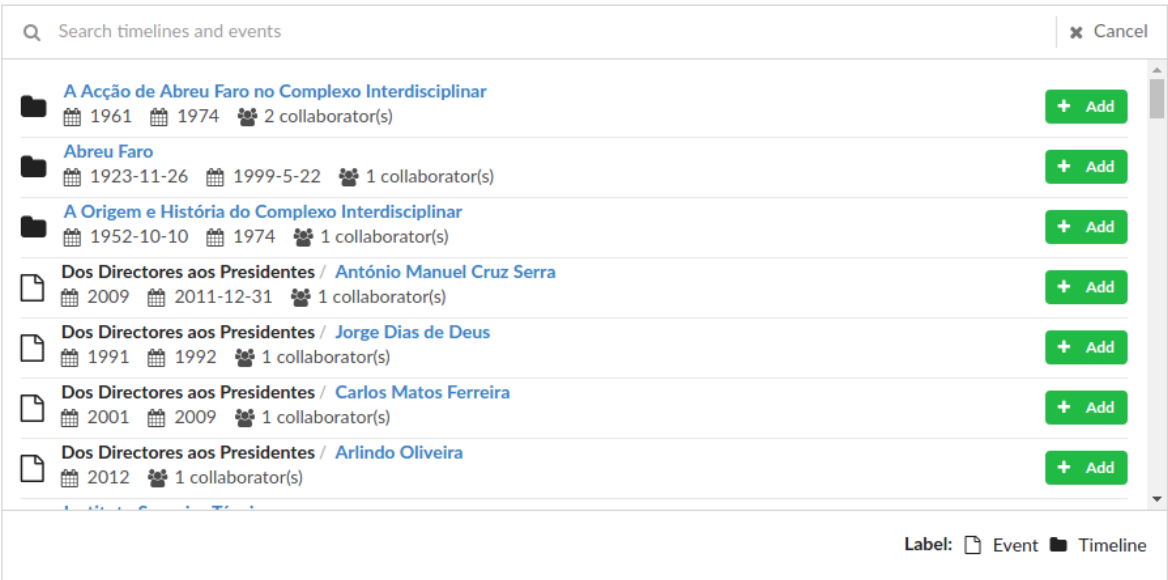

Figure 5.9: Snippet of the timeline's "Add Contents" page

The events and sub-timelines that are aggregated within the same category always appear under the group's specified name. However, unless the desired category has already been created, the option "Group" presented in the "Manage Contents" page will not be displayed. To perform such operation, one must first access the "Groups and Tags" page (Fig. 5.10), which is completely aimed at managing all the groups and tags associated with a timeline. An important aspect to keep in mind is that, when a user tries to associate a tag with a particular timeline, it is recommended that he uses one of the already created tags, considering that these take an active role in the internal definition of the relationships between multiple different timelines.

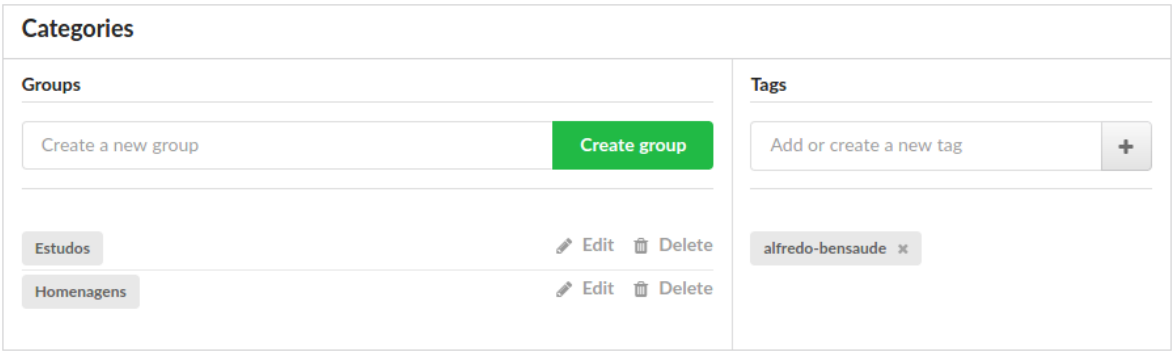

Figure 5.10: Cutout of the timelines's "Groups and Tags" page

Through the "Contributors" page (Fig. 5.11), the owner of a timeline can add new collaborators, or remove any of the already included, to and from the access list. Even though all the illustrated views are screenshots taken from the owner's perspective, the allowed functionalities that the contributors or the owner of the timeline are entitled to execute are not the same: while the first type of users is only authorised to manage the contents of a timeline, to create new groups and tags and edit existing ones, and to update the timeline's media object, the owner is capable of performing all the operations regarding a particular timeline.

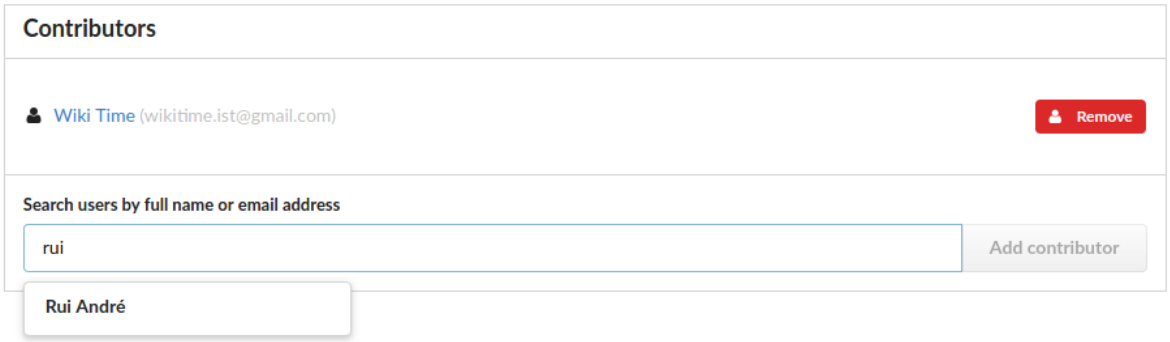

Figure 5.11: Snippet of the timeline's "Contributors" page

In Section 3.1.1.2, it was also referred the necessity to implement functionalities related to modifying the media content associated with a timeline, to changing its information (headline, description) and domain, or to the deletion of the timeline's record. Nevertheless, although these have been developed, we feel that it is not necessary to display the correspondent pages, since they are in every aspect identical to the ones presented in the previous section, concerning the editing and deletion of events (Figs. 5.5 and 5.7, respectively).

5.2.2.4 Reusing Timelines or Events

When the user clicks to reuse an event, by using the "Reuse" button (Fig. 5.12), the system internally creates a copy of the original object and inserts in both documents a reference to each other. The type of relationship formed is similar to a parent-child association, in which the original record is the parent, and all the generated copies are the child. Accordingly, when reusing a timeline, not only the document of the specific timeline itself is cloned, but all the events included in it are as well, and all the respective connections are defined. By performing as such, we are able to produce a network of associations that allow us to further create and display all the existing relationships between multiple distinct timelines and events.

Figure 5.12: Sample of the "Reuse" button

The operation of reusing timelines and events is also susceptible to the defined user-model permissions. In a first instance, only registered active users and administrators are entitled to execute this type of action. But while administrators are capable of reusing any timeline or event, normal registered users can only reuse those that are identified as public or for which they are included in the object's access list. As an example, we can consider the case in which a user decides to reuse a timeline that contains both public and private events: if, in this case, the user is neither an owner nor a collaborator, he will only be able to reuse the public events contained in the respective timeline.

5.2.3 Web-based Timeline Software Comparison

In Section 2.3, we presented a few Web-based applications that allow the creation of visually attractive timelines. These tools, despite displaying some similarities with the technology developed for this project, still do not provide some of the functionalities that we consider to be strictly necessary to allow the proper comprehension of the historical contents. Represented in Table 5.1 is a summary of the key functionalities that were developed for *WikiTime*'s application, along with a comparison with the other existing Web-based timeline software systems. While the checkmark "✔" and the cross "✖" symbols indicate, respectively, that the specified feature is or is not included in the product, the black dot "●" symbolises that the mentioned functionality undergoes some limitations.

Hypotethically, if we had decided to develop our application using Neatline on top of the Omeka framework, we would most probably also be able to implement the same features as we did for current *WikiTime*'s application. However, Neatline was not considered for this evaluation because there is no actual system that has been implemented using this plugin and, as such, we can not perform a truthful comparison between this library and the other existing platforms.

Table 5.1: Comparison between WikiTime and other Web-based platforms

		WikiTime	TimelineJS	Sutori	Tiki-Toki
Events	Create	✔	✔	✔	✔
	Edit Media, Dates, and Information	✔	✔	✔	✔
	Reuse	✔	✖	✖	✖
	Delete	✔	✔	✔	✔
Timelines	Create	✔	✔	✔	✔
	Add and Remove Events	✔	✔	✔	✔
	Group Events	✔	✔	✖	✔
	Add and Remove Sub-Timelines	✔	✖	✖	✖
	Remove Sub-Timelines	✔	✖	✖	✖
	Group Sub-Timelines	✔	✖	✖	✖
	Add and Remove Tags	✔	✖	✖	✖
	Edit Media and Information	✔	✔	✔	✔
	Reuse	✔	✖	✖	✖
	Delete	✔	✔	✔	✔
Privacy Control Management		✔	✔	✔	✔
Collaboration Mechanism		✔	✖	●	●
Define Relationships		✔	✖	✖	✖
Open-Source		✔	✔	●	●

5.3 Proof-of-Concept Demonstration

Anyone accessing *WikiTime*'s application, regardless of having an account or not, is entitled to visualise any of the already created timelines and events, as long as these are classified as public. If not, only the users included in the object's access list (and administrators) are allowed to perform such action. To demonstrate the uniqueness of the concept behind the developed system, 17 timelines and 114 events were created, mainly regarding the history of Instituto Superior Técnico and three famous personalities that were actively involved in the creation and evolution of the institution. However, to allow a proper comprehension of how the implemented system solves the existing problems with current presentations of history, we must first discuss the traditional methods for organising and displaying historical data on the Web.

5.3.1 Traditional Methods

The origins of Instituto Superior Técnico date back to the former Instituto Industrial e Comercial de Lisboa, founded in 1852 and of which Alfredo Bensaude was first a professor and then, for a short period, the principal. After the Portuguese Republic was established in 1910, the Minister of Public Works at that time decided to extinguish the referred Institute and, consequently, to divide it into the

Instituto Superior Técnico and the Instituto Superior do Comércio. Alfredo Bensaude was nominated the first principal of Instituto Superior Técnico, for which he fought hard to be able to provide better conditions for its students. Despite being extremely interactive, the existing website which concerns the history of Alfredo Bensaude (Fig. 5.13) lacks in the presentation of some historical contents that give context to specific events, namely the given short introduction. Furthermore, it is troublesome when one desires to comprehend how the different aspects of Alfredo Bensaude's life (for example regarding his scientific activity, his personal life, or his education) are overlapping and interconnecting with each other. Since the multiple events are organised by different themes, it is not clear how these are placed in time and, at some point, it seems that the multiple parts of Alfredo Bensaude's history are entirely independent of each other and occurred at different moments. On the other hand, considering that various types of textual and media documents are presented in a format similar to a slideshow, one interesting feature that we withdraw from this website is related to how the data is displayed to the user, through the usage of several simple dynamic elements.

Figure 5.13: *Alfredo Bensaude: Vida e Obra*, created by Instituto Superior Técnico and Núcleo de Arquivo [6]

Considering the poor and inadequate shape of the infrastructures of the Instituto Industrial e Comercial de Lisboa, Alfredo Bensaude had the desire to perform works of improvement to the institute, which never took place because of a lack of funds. A few years after Alfredo Bensaude left the post of principal of the Instituto Superior Técnico, Duarte Pacheco was appointed to the position. Known for having a great futuristic vision, he is then the one responsible for the translation of the Instituto Superior Técnico's facilities to its present location, hence giving shape to the ideas previously conceived by Alfredo Bensaude. Even though this fact is referred to in Alfredo Bensaude's website, there is no actual link to redirect the user to a place where more information about this topic is displayed. In fact, despite the notable influence that Duarte Pacheco had in the history of Instituto Superior Técnico, the only Web page, created by some people of the referred institute, that contains information about his work is the one presented in Fig. 5.14. This website, however, presents the same problems that were previously discussed in Section 1.1: the displayed information lacks the design and functionality required to provide the readers with a more engaging and fluid experience. In this case, data is presented as a simple big block of text, having no component of interactive and dynamic nature. Additionally, the written contents are too focused on Duarte Pacheco's life as a politician, rather than on the aspects of his life as the principal of Instituto Superior Técnico.

Figure 5.14: *Duarte Pacheco: A Vida e a Obra*, created by Instituto Superior Técnico [7]

Another person of interest that had a grand influence in the development of not only the Instituto Superior Técnico but also of the Portuguese republic is Manuel Abreu Faro. Even though his actions at Instituto Superior Técnico only took place many years after Duarte Pacheco left the direction of the Institute (due to a tragic dead in a road accident), in 1953, Abreu Faro is considered a person of high regards. He is the one responsible for the creation of the Complexo Interdisciplinar, a popular research facility which is still in operation in the current days. As a way to honour the type of person he was and his achievements, the people from Instituto Superior Técnico, together with the department responsible for the gathering and preservation of historical contents regarding the referred Institute (Núcleo de Arquivo), developed a website containing some information about Abreu Faro (Fig. 5.15). Nevertheless, despite the fact that the presented Web pages contain different types of textual and visual elements, these are poorly organised and do not follow a logical scheme. There are lots of pictures and documents, but little to no information to give context to this elements. Overall, the Web site itself does not include a conducting wire that allows the reader to understand the information to be transmitted.

It is important to remember that all these three personalities have a common background related to Instituto Superior Técnico but in no way these associations are displayed anywhere. When representing history, more than being able to provide clear and complete information about each individual, it is essential that we describe and define the relationships between the multiple existing entities. Timelines give the ability to represent history as a series of interconnected and overlapping events, hence providing the reader with the possibility to detect patterns and trends across time, and to understand how multiple different personalities and facts cross each other. However, the transition from the paper versions to the digital ones was still rather incomplete and immature. The digital technologies already have a series of features that had not been used efficiently in current representations of history, and further thought had to be given about it. Ultimately, the idea that was always present in our mind was that the reader must be able to travel through history as if he was reliving his own personal experience, even in moments that he did not experience in the first place.

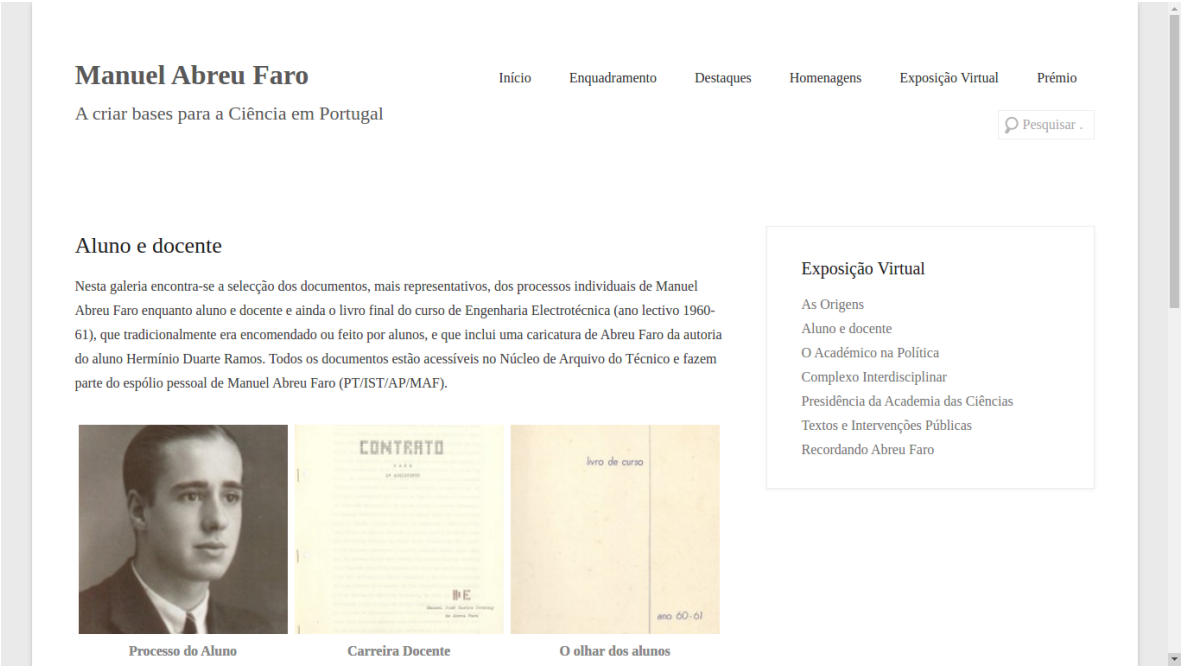

Figure 5.15: *Manuel Abreu Faro: A criar bases para a Ciência em Portugal*, created by Instituto Superior Técnico and Núcleo de Arquivo [8]

5.3.2 WikiTime's Method

When we first thought about developing such application, our main goal was to preserve all the beneficial aspects that the traditional methods have while working on the improvement of the defective ones. As discussed in Section 2.3.1, TimelineJS itself already allowed, for a single entity, the presentation of historical data in a proper format, enabling the use of several different types of textual and media documents and the organisation of the multiple existing events in chronological order. The biggest problem, however, lied within the impossibility to associate any related timelines, or events, with one another. As such, *WikiTime* was developed with this issue in mind and, from the start, one of our goals was to design and develop a platform that would permit displaying all the existing associations between the several distinct timelines.

The history of an organisation is mainly composed of the people that belonged to it, and by knowing about the life of each individual, we are also allowing ourselves to comprehend the impacts one had to a particular institution. In the case of Instituto Superior Técnico, it is most probable that, for example, the time that Alfredo Bensaude passed studying in a foreign country was what most influenced him to fight thoroughly for a better and more organised educational system for the referred Institute. On another perspective, Duarte Pacheco's decision of giving shape to Bensaude's desire and build new and improved facilities for Instituto Superior Técnico, in a central location of the city of Lisbon, was justified by his great visionary mind. Lastly, it is also likely that Abreu Faro's passion for science was what led him to pursue the dream of building a top-notch infrastructure for researching purposes. These are all notions and aspects of the history of the referred institute that somehow get lost when using history's traditional methods, but that can be easily understood by the reader once he can utilise a platform, such as *WikiTime*, that enables him to "travel in time".

To support and demonstrate the validity of this concept, we have begun by designing the timeline concerning the history of Instituto Superior Técnico, in which were included three other timelines that also take part in the ones related to Alfredo Bensaude, Duarte Pacheco, and Abreu Faro. As such, once a user starts exploring the timeline "Instituto Superior Técnico", he will surely be able to move to the main timelines of the referred characters, as these also contain some sub-timelines that are included in the timeline of the institute's history. It becomes clear that an underground network of inter-related timelines and sub-timelines exists, and the methods for access and travel along its channels are the following two:

1. Fig. 5.16 is a screenshot of the timeline "Instituto Superior Técnico". The first aspect that we would like to highlight is related to the existence of two types of elements shown in the slider: the events, marked with an "E", and the timelines, referred to with a "T". Regarding the latest, if an element contains such mark, this means that the slide's headline is actually a link that, upon being clicked, will trigger the application to redirect the user to that particular timeline, as shown in Fig. 5.17;

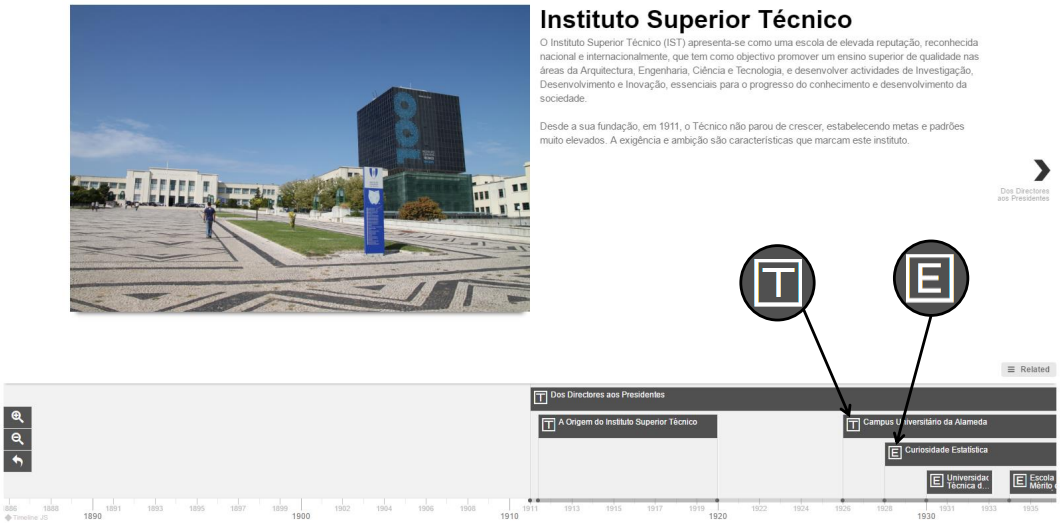

Figure 5.16: Timeline of "Instituto Superior Técnico", created using *WikiTime*'s software application

2. The other technique is represented in Fig. 5.18, which consists of displaying a list of all the timelines that are associated with the one the user is currently visualising. This list can be obtained by clicking on the "Related" button that is placed on the right bottom corner of the interface. One must keep in mind that every slide included in the slideshow will display a different list: while for events, this list is limited to all the timelines that possess a copy of the event, for sub-timelines, the list includes all parent and children timelines as well as other that have equal tag values;

By implementing the described techniques, we allow users to move from timeline to timeline almost seamlessly and enable them to obtain further information about a certain topic. Even so, it is true that further thinking regarding this topic can be made, with the intention of improving the developed

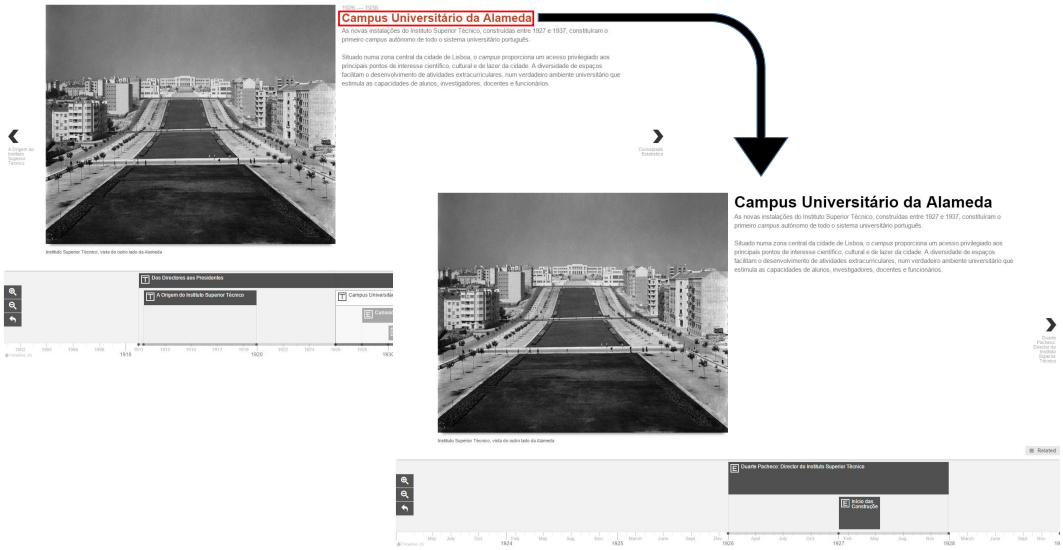

Figure 5.17: Representation of the "Sub-Timelines" concept

Figure 5.18: Timeline "A Acção de Abreu Faro no Complexo Interdisciplinar" and the list of all related timelines

methods for presenting related information. Ideally, it would be nice to have multiple sliders, one on top of the other, containing the distinct events of the multiple associated timelines. Deciding on which slides, from which timelines, would be displayed in the interface, would be the responsibility of the user. Upon moving the time frame in one timeline, the other could also go forward, or backwards, accordingly. However, in our understanding, a task like this would require the knowledge of someone more experienced in the graphic design for the Web, as sometimes it can be hard for a developer like us to have this kind of sensibility.

6

Conclusions

Contents

6.1 Thesis Overview

History has never been so actively disseminated as it is today. Throughout the years of its short existence, the WWW has allowed historians to present their knowledge and experiences by using distinct types of strategies, like for example blogs, forums, or even personal Web pages. Nonetheless, the prevailing presentations of historical data, limited by the early versions of HTML, are still structured in a tabular layout that does not allow readers to efficiently comprehend the topics under examination. Recently, new digital technologies have been introduced, enabling the creation of visually engaging timelines that display the contents in a more interactive and dynamic format. But in our perspective, these modern software applications still face some problems that we consider to be of significant importance, namely the inability to represent the multiple linked data accurately or to reuse and embed previously created information. Furthermore, the majority of these tools either do not have support for collaborative sharing and production of contents, or it is rather limited in the numper of people allowed to work on a joint project.

The objective of this research was to address these referred limitations, by defining and implementing a Web-based software system, entitled *WikiTime*, that allows the simple creation and editing of timelines for the proper visualisation and organisation of time-related data. Moreover, the envisaged solution was developed with the words flexibility and integration in mind and thus, building a Web Service API that would allow the complete integration of *WikiTime*'s functionalities with other external applications was also a key aspect that was taken into consideration. To approach this question, we started by analysing in detail some of the existing tools: once we clearly identified all the posed issues contained in these technologies, we were able to precisely define all the requirements and objectives that have guided us during the development stage of our software product. Based on these specifications, we proceeded to describe the data model that serves as the basis for our platform, to specify the technologies and frameworks that were used for the implementation, and to define the supporting architecture of the system.

To confirm that the devised software product can indeed be utilised as an instrument for the graphic representation of history, distinct data was gathered and utilised to create 17 timelines and more than 110 events. When compared with the traditional methods, *WikiTime*'s application offers a more effective interface, enabling its users to move quickly through time, to efficiently analyse and compare different historical periods, and to quickly move between closer or wider perspectives. Built upon a collaborative working environment, it is capable of providing its users with the possibility to properly organise events in time, to display the complex relationships that compose history, and to reuse or encapsulate previously created timelines and events. In general, *WikiTime*'s software application has proven to be a tool worthy of chronological display, providing meaning and form to history's connotations of narrative and allowing for greater comprehension of the world's past, present, and future. Ultimately, we expect that it may be utilised as the fuel for further developments that will, hopefully, take into consideration some of the the aspects that require greater thinking.

6.2 Future Work

Even though the objectives defined for this project were successfully achieved, one aspect that we have learned from this long-lasting process was that there is always room for further improvements, meaning that new features can always be added and implemented into the system. With that being said, the following future work is proposed:

1. Timelines and events' objects can only have one media file associated with each record. However, there are a few cases in which it would be useful to be able to display multiple images related to a single event. This functionality would only require some slight modifications to the system's data structure and implementation, being the greatest challenge associated with the amendment of TimelineJS' library to support the display of several images using, for example, a carousel slideshow;

2. Similar to what Neatline does, allow users to map events not only in time but also in space. The temporal and spatial representation of events would allow the readers to feel more involved in the subject, thus increasing the overall experience provided by the application;

3. Study how multiple different timelines can be displayed in a single TimelineJS' instance. At the user's choice, multiple interconnected timelines could be affixed to the events' slider, allowing for better comprehension of how these were related to one another and at which point in time they crossed each other. Then again, to implement this functionality, it would be necessary to apply some changes to TimelineJS' code;

4. Instead of all the associations between timelines and events being built internally by the system, design new techniques that would allow users to manually define some of these relationships. For example, when creating an object of any kind, the user could tag already existing events, which would then produce a link;

5. Provide users with the possibility to export any of the existing timelines or events into a format that could be displayed outside the scope of the browser, like in a Microsoft Powerpoint presentation, or into a Portable Document Format (PDF) file.

Bibliography

[1] Stearns, P. N., "Why Study History?" 1998. [Online]. Available: https://www.historians.org/about-aha-and-membership/aha-history-and-archives/archives/why-study-history-(1998). (accessed on 02-11-2016).

[2] "Brief History of the American Historical Association." [Online]. Available: https://www.historians.org/about-aha-and-membership/aha-history-and-archives/brief-history-of-the-aha. (accessed on 02-11-2016).

[3] Cohen, D., and Rosenzweig, R., *Digital History: A Guide to Gathering, Preserving, and Presenting the Past on the Web*. University of Pennsylvania Press, 2005.

[4] O'Luanaigh, C., "The birth of the web." December 2013. [Online]. Available: http://home.cern/topics/birth-web. (accessed on 05-11-2016).

[5] "History of the Web." [Online]. Available: http://webfoundation.org/about/vision/history-of-the-web/. (accessed on 05-11-2016).

[6] Instituto Superior Técnico, and Núcleo de Arquivo, "Vida e Obra de Alfredo Bensaúde." 2013. [Online]. Available: http://alfredobensaude.ist.utl.pt/. (accessed on 13-10-2016).

[7] Instituto Superior Técnico, and Núcleo de Arquivo, "Duarte Pacheco: A Vida e a Obra." 2010. [Online]. Available: http://100.ist.utl.pt/momentos/duarte-pacheco/. (accessed on 13-10-2016).

[8] Instituto Superior Técnico, and Núcleo de Arquivo, "Manuel Abreu Faro – a criar bases para a Ciência em Portugal." 2015. [Online]. Available: http://abreufaro.tecnico.ulisboa.pt/. (accessed on 13-10-2016).

[9] Ribecca, S., "The Data Visualisation Catalogue." [Online]. Available: http://www.datavizcatalogue.com/index.html. (accessed on 06-11-2016).

[10] Davis, S. B., Bevan, E., and Kudikov, A., "Just in time: defining historical chronographics." in *Electronic Visualisation and the Arts*, July 2010, pp. 355–362.

[11] Rosenberg, D., and Grafton, A., *Cartographies of Time: A History of the Timeline*. Princeton Architectural Press, 2010.

[12] Davis, S. B., “Time machines.” in *Technologies and ’the death of Art History’: Computers and the History of Art annual conference*, ser. Computers and the History of Art Yearbook, 2010.

[13] Wise, Z., and Knight Lab Team, “Timeline JS3 — Beautifully crafted timelines that are easy, and intuitive to use.” Northwestern University. [Online]. Available: https://timeline.knightlab.com/. (accessed on 16-01-2017).

[14] Sutori, “Sutori.” [Online]. Available: https://www.sutori.com/. (accessed on 17-01-2017).

[15] Webalon Ltd, “Beautiful web-based timeline software.” [Online]. Available: http://www.tiki-toki.com/. (accessed on 17-01-2017).

[16] Scholars’ Lab, “Neatline.” University of Virginia Library. [Online]. Available: http://neatline.org/. (accessed on 19-01-2017).

[17] Nielsen, J., “Does Internet = Web?” September 1998. [Online]. Available: https://www.nngroup.com/articles/does-internet-web/. (accessed on 11-11-2016).

[18] Sheriff, P. D., “Designing for Web or Desktop?” January 2002. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms973831.aspx. (accessed on 06-01-2017).

[19] Landrum, A., “Focusing on UI/UX and The Importance of Good Design.” July 2014. [Online]. Available: https://www.upandup.agency/creative-design/importance-of-good-ui-ux. (accessed on 10-01-2017).

[20] Patterson, M., “What Is an API, and Why Does It Matter?” April 2015. [Online]. Available: http://sproutsocial.com/insights/what-is-an-api/. (accessed on 12-01-2017).

[21] Instituto Superior Técnico, and Núcleo de Arquivo, “100 Anos IST - dos Directores aos Presidentes.” [Online]. Available: https://narq.tecnico.ulisboa.pt/extensao-cultural-e-educativa/exposicoes-fisicas-eou-virtuais/100-anos-ist-dos-directores-aos-presidentes/. (accessed on 13-10-2016).

[22] Wikipedia, “Annales Alamannici – Wikipedia, The Free Encyclopedia.” June 2016. [Online]. Available: https://en.wikipedia.org/wiki/Annales_Alamannici. (accessed on 21-11-2016).

[23] “52 Weeks of Fantastic Bindings, Week 41: Gorgeous Early 16th Century French Clasps.” March 2012. [Online]. Available: https://standrewsrarebooks.wordpress.com/2012/03/21/52-weeks-of-fantastic-bindings-week-41-gorgeous-early-16th-century-french-clasps/. (accessed on 21-11-2016).

[24] University of Oregon, “A Cronographic Machine – Ludlow’s Concentric Chart of History.” [Online]. Available: http://pages.uoregon.edu/dbr/time-online/ludlow/chronographic_machine.php. (accessed on 21-11-2016).

[25] Davis, S. B., “Chronographics: The First Modern Timeline?” August 2009. [Online]. Available: http://chronographics.blogspot.pt/2009/08/first-modern-timeline.html. (accessed on 25-11-2016).

[26] Davis, S. B., “Chronographics: "The Idea of a Measurable Space" - Joseph Priestley’s 1765 Chart of Biography.” May 2016. [Online]. Available: http://chronographics.blogspot.pt/2016/05/the-idea-of-measurable-space-joseph.html. (accessed on 25-11-2016).

[27] Davis, S. B., “Joseph Priestley: The Man Who Drew Time.” *Postings From Priestley House, the Newsletter of The Friends of Joseph Priestley House*, no. 49, April 2011.

[28] Davis, S. B., “History on the Line: Time as Dimension.” *Design Issues*, vol. 28, no. 4, pp. 4–17, 2012.

[29] Wikipedia, “Odes (Horace) – Wikipedia, The Free Encyclopedia.” 2016. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Odes_(Horace)&oldid=752500966. (accessed on 28-11-2016).

[30] Champagne, M., “Diagrams of the past: How timelines can aid the growth of historical knowledge.” *Cognitive Semiotics*, vol. 9, no. 1, pp. 11–44, 2016.

[31] Friendly, M., “Gallery of Data Visualization – Timelines and Visual Histories.” December 2016. [Online]. Available: http://www.datavis.ca/gallery/timelines.php. (accessed on 02-12-2016).

[32] Silver, M., “New tool: An easy way to build attractive timelines.” March 2012. [Online]. Available: http://knightlab.northwestern.edu/2012/03/21/an-easy-way-to-build-attractive-timelines/. (accessed on 16-01-2017).

[33] “Introducing JSON.” [Online]. Available: http://www.json.org/. (accessed on 13-01-2017).

[34] Roy Rosenzweig Center for History and New Media, “Omeka.” George Mason University. [Online]. Available: http://omeka.org/. (accessed on 19-01-2017).

[35] Wikipedia, “LAMP (software bundle) — Wikipedia, The Free Encyclopedia.” 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=LAMP_(software_bundle)&oldid=758448488. (accessed on 19-01-2017).

[36] Open Source Initiative, “Licenses & Standards | Open Source Initiative.” [Online]. Available: https://opensource.org/licenses. (accessed on 18-01-2017).

[37] Azarian, I., “Why is Documentation Extremely Important for Developers?” September 2013. [Online]. Available: http://www.seguetech.com/why-is-documentation-extremely-important-for-developers/. (accessed on 18-01-2017).

[38] Carnegie Mellon University, “Software System Integration (A Framework for Software Product Line Practice).” [Online]. Available: http://www.sei.cmu.edu/productlines/frame_report/softwareSI.htm. (accessed on 18-01-2017).

[39] Richardson, M., “Timeglider: Web-based Timeline Software.” [Online]. Available: https://timeglider.com/. (accessed on 18-01-2017).

[40] Huynh, D. F., “SIMILE Widgets | Timeline.” [Online]. Available: http://simile-widgets.org/timeline/. (accessed on 18-01-2017).

[41] Müller, S., “Timesheet.js - Beautiful time tables with HTML, JavaScript and CSS.” [Online]. Available: https://sbstjn.com/timesheet.js/. (accessed on 18-01-2017).

[42] Takur, D., “Principles of Software Design & Concepts in Software Engineering.” [Online]. Available: http://ecomputernotes.com/software-engineering/principles-of-software-design-and-concepts. (accessed on 24-01-2017).

[43] Bartlett, G., “Systemic Thinking, a simple thinking technique for gaining systemic focus.” in *The International Conference on Thinking "Breakthroughs"*, 2001.

[44] Takur, D., “Requirements Analysis in Software Engineering.” [Online]. Available: http://ecomputernotes.com/software-engineering/requirementsanalysis. (accessed on 24-01-2017).

[45] Wise, Z., and Knight Lab Team, “TimelineJS JSON data format.” Northwestern University. [Online]. Available: https://timeline.knightlab.com/docs/json-format.html. (accessed on 26-01-2017).

[46] Wise, Z., and Knight Lab Team, “Media Types.” Northwestern University. [Online]. Available: https://timeline.knightlab.com/docs/media-types.html. (accessed on 26-01-2017).

[47] Takur, D., “Data Design in Software Engineering.” [Online]. Available: http://ecomputernotes.com/software-engineering/data-design. (accessed on 23-01-2017).

[48] Mamčenko, J., “Lecture Notes on INFORMATION RESOURCES.” *Introduction to Data Modeling and MSAccess*, 2004.

[49] Cinergix Pty Ltd., “Ultimate Guide to ER Diagrams (Entity-Relationship Diagrams).” March 2012. [Online]. Available: http://creately.com/blog/diagrams/er-diagrams-tutorial/. (accessed on 23-01-2017).

[50] Bell, D., “An introduction to the Unified Modeling Language.” June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html. (accessed on 22-01-2017).

[51] Fakhroutdinov, K., “The Unified Modeling Language.” [Online]. Available: http://www.uml-diagrams.org/. (accessed on 22-01-2017).

[52] Niles, S., “Selecting the Right Software Development Tools for Your Developers.” [Online]. Available: https://blog.smartdraw.com/selecting-the-right-software-development-tools-for-your-developers/. (accessed on 05-01-2017).

[53] "The World Wide Web project." [Online]. Available: http://info.cern.ch/hypertext/WWW/TheProject.html. (accessed on 05-01-2017).

[54] Jazayeri, M., "Some Trends in Web Application Development." in *2007 Future of Software Engineering*, ser. FOSE '07. IEEE Computer Society, 2007, pp. 199–213.

[55] Carlson, R., "The Evolution of Web Design: 1990-Present." [Online]. Available: http://www.designjuices.co.uk/2011/09/web-design-evolution/. (accessed on 05-01-2017).

[56] Microsoft Patterns, *Microsoft Application Architecture Guide*, 2nd ed. Microsoft Press, October 2009.

[57] Roestamadji, P., "Website VS. Web Application — What's the difference?" June 2015. [Online]. Available: https://www.lionandpanda.com/website-vs-web-application-whats-the-difference/. (accessed on 06-01-2017).

[58] Smith, J., "Desktop Applications Vs. Web Applications." [Online]. Available: http://www.streetdirectory.com/travel_guide/114448/programming/desktop_applications_vs_web_applications.html. (accessed on 06-01-2017).

[59] World Wide Web Consortium, "HTTP — Hypertext Transfer Protocol." [Online]. Available: https://www.w3.org/Protocols/. (accessed on 08-01-2017).

[60] Ndegwa, A., "What is a Web Application?" May 2016. [Online]. Available: https://www.maxcdn.com/one/visual-glossary/web-application/. (accessed on 06-01-2017).

[61] Dale, T., "You're Missing the Point of Server-Side Rendered JavaScript Apps." February 2015. [Online]. Available: http://tomdale.net/2015/02/youre-missing-the-point-of-server-side-rendered-javascript-apps/. (accessed on 08-01-2017).

[62] Klimushyn, M., "Web Application Architecture from 10,000 Feet, Part 1 — Client-Side vs. Server-Side." April 2015. [Online]. Available: https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/. (accessed on 08-01-2017).

[63] Klimushyn, M., "Web Application Architecture from 10,000 Feet, Part 3 — Frontend Development." April 2015. [Online]. Available: https://spin.atomicobject.com/2015/04/20/front-end-dev-frameworks-libraries/. (accessed on 08-01-2017).

[64] Laurent, S. S., "Web Application Development is Different (and better)." January 2014. [Online]. Available: https://www.oreilly.com/ideas/web-application-development-is-different-and-better. (accessed on 10-01-2017).

[65] World Wide Web Consortium, "Document Object Model (DOM)." [Online]. Available: https://www.w3.org/DOM/. (accessed on 10-01-2017).

[66] Mozilla Developer Network, "About JavaScript." October 2015. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. (accessed on 10-01-2017).

[67] Github, Inc., "Design Essentials." [Online]. Available: https://github.com/showcases/design-essentials. (accessed on 10-01-2017).

[68] Github, Inc., "Front-End JavaScript Frameworks." [Online]. Available: https://github.com/showcases/front-end-javascript-frameworks. (accessed on 10-01-2017).

[69] Shiotsu, Y., "Web Development 101: Top Web Development Languages in 2014." 2014. [Online]. Available: https://www.upwork.com/blog/2014/03/web-development-101-top-web-development-languages-2014/. (accessed on 10-01-2017).

[70] Github, Inc., "Web Application Frameworks." [Online]. Available: https://github.com/showcases/web-application-frameworks. (accessed on 10-01-2017).

[71] Mozilla Developer Network, "Server-Side Web Frameworks." December 2016. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. (accessed on 10-01-2017).

[72] Heng, C., "What is MySQL? What is a Database? What is SQL?" October 2016. [Online]. Available: https://www.thesitewizard.com/faqs/what-is-mysql-database.shtml. (accessed on 21-01-2017).

[73] Serra, J., "Relational Databases VS. Non-Relational Databases." August 2015. [Online]. Available: http://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/. (accessed on 21-01-2017).

[74] solid IT gmbh, "DB-Engines Ranking of Relational DBMS." [Online]. Available: http://db-engines.com/en/ranking/relational+dbms. (accessed on 21-01-2017).

[75] MongoDB, Inc., "NoSQL Database Explained." [Online]. Available: https://www.mongodb.com/nosql-explained. (accessed on 21-01-2017).

[76] MongoDB, Inc., "Top 5 Considerations When Evaluating NoSQL Databases." Tech. Rep., February 2015.

[77] MongoDB, Inc., "MongoDB for GIANT Ideas." [Online]. Available: https://www.mongodb.com/. (accessed on 13-01-2017).

[78] StrongLoop et al., "Express — Node.js Web Application Framework." [Online]. Available: http://expressjs.com/. (accessed on 13-01-2017).

[79] Google, "AngularJS — Superheroic JavaScript MVW Framework." [Online]. Available: https://angularjs.org/. (accessed on 13-01-2017).

[80] Node.js Foundation, "Node.js." [Online]. Available: https://nodejs.org/en/. (accessed on 13-01-2017).

[81] Karpov, V., "The MEAN Stack: MongoDB, ExpressJS, AngularJS and Node.js." April 2013. [Online]. Available: https://www.mongodb.com/blog/post/the-mean-stack-mongodb-expressjs-angularjs-and. (accessed on 13-01-2017).

[82] Salesforce.com, "Create a Web App and RESTful API Server Using the MEAN Stack." October 2016. [Online]. Available: https://devcenter.heroku.com/articles/mean-apps-restful-api. (accessed on 14-01-2017).

[83] Henning, M., "API Design Matters." *Queue*, vol. 5, no. 4, pp. 24–36, May 2007.

[84] Friedrich, T., *RESTful Service Best Practices: Recommendations for Creating Web Services.*, Pearson eCollege, 2013.

[85] Elkstein, M., "Learn REST: A Tutorial." [Online]. Available: http://rest.elkstein.org/. (accessed on 15-01-2017).

[86] Auth0, "JSON Web Tokens — jwt.io." [Online]. Available: https://jwt.io/. (accessed on 17-01-2017).

[87] Jones et al., "JSON Web Token (JWT)." May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519. (accessed on 15-01-2017).

[88] Stamelos, I., Vlahavas, I., Refanidis, I. and Tsoukiàs, A., "Knowledge Based Evaluation of Software Systems: a Case Study." *Information and Software Technology*, vol. 42, no. 5, pp. 333–345, 2000.

[89] Bursztein, E., "Analyzing Web Application Performance." March 2011. [Online]. Available: https://www.elie.net/blog/web/analyzing-web-application-performance. (accessed on 28-03-2017).

[90] Lumsden, A., "Best Practices for Increasing Website Performance." October 2012. [Online]. Available: https://webdesign.tutsplus.com/articles/best-practices-for-increasing-website-performance--webdesign-9109. (accessed on 28-03-2017).

[91] Beal, V., "CDN - Content Delivery Network." [Online]. Available: http://www.webopedia.com/TERM/C/CDN.html. (accessed on 28-03-2017).