



Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN SISTEMAS DE INFORMACIÓN

Mapp, GIS web application for the cartographic representation of 17th century notices and news

Estudiante: Carmen Corrales Camello

Dirección: Miguel Ángel Rodríguez Luaces

Dirección: Alejandro Cortiñas Álvarez

A Coruña, xuño de 2020

To my family, for giving me the wings to fly.

Acknowledgements

To my mentors, Miguel and Alejandro for their patience and their time spent on this project. They helped me to solve the problems that arose during this process even if the circumstances were not the best ones due to the Covid-19.

To my family, for inspiring me every day and supporting me even in those moments when I felt hopeless

To my friends, for turning bitter moments into sweet ones through laughs.

Abstract

The objective of this end-of-degree work is to develop a web application that allows a group of philologists to carry out a study on the publication of gazettes during the 17th century, which they were newspapers that disseminated official, political, literary news... These gazettes could be published and edited in different cities and at the same time, they contained news that narrated events, which have happened in a specific place and could be reported in another different place. Therefore, it is difficult to see the historical evolution of news publishing.

To facilitate the search and visualization of the information, a web application has been designed to manage the information about the newspapers, their news, the publishers and the places they refer to. It also allows to visualize an interactive map that shows the places where a gazette was published or edited and where the news item occurred or was notified. For each location it will show information about the number of events that have occurred for each one of the activities (Edition, publication, event, notice).

At the beginning of this project, a previous analysis was made to obtain information about the objectives and thus establish the set of requirements necessary for the functioning of the application. Subsequently, the design and development of the application was carried out.

The application is composed of a back-end that takes care of the application logic, and is formed by a server that has been implemented with Java and Spring, and a DBMS, and a front-end that is a Web client that has been implemented with Vue.js.

In order to develop the application, it was decided to use an agile methodology. At the beginning of the project in the planning meeting the number and duration of the iterations that would divide the development process were defined.

Resumo

El objetivo de este trabajo de fin de grado consiste en desarrollar una aplicación web que permita a un grupo de filólogos realizar un estudio sobre la publicación de gacetas durante el siglo XVII, las cuales son periódicos donde se difundían noticias oficiales, políticas, literarias... Estas gacetas han podido ser publicadas y editadas en distintas ciudades y a su vez, contienen noticias que narran sucesos, los cuales han sucedido en un lugar concreto y pueden ser notificados en otro lugar distinto, esto provoca que resulte difícil ver la evolución histórica en la publicación de noticias.

Para poder facilitar la búsqueda y visualización de la información se ha diseñado una aplicación web que permite gestionar la información sobre los periódicos, sus noticias, sobre

los editores y los lugares a los que hacen referencia. También permite visualizar un mapa interactivo que muestra los lugares donde se publicó o se editó una gaceta y en los que ocurrió o se notificó el suceso de una noticia. Para cada lugar mostrará información sobre el número de sucesos que han ocurrido para cada una de las actividades.

Al comienzo de este proyecto se realizó un análisis previo para así obtener información sobre los objetivos y así establecer el conjunto de requisitos necesarios para el funcionamiento de la aplicación. Posteriormente se realizó el diseño y el desarrollo de la aplicación.

La aplicación consta de un backend que se encarga de la lógica de la aplicación, y está formado por un servidor que se ha implementado con Java y Spring, y de un SGBD, y de un frontend que es cliente Web que se ha implementado con Vue.js.

Para desarrollar la aplicación se decidió usar una metodología ágil para ello al inicio del proyecto en la reunión de planificación se marcó el número y duración de las iteraciones que dividirían el proceso de desarrollo.

Keywords:

- Gazette
- Publication
- Edition
- Notice
- Event
- Web application
- Leaflet
- Vue.js
- Rest Service

Palabras chave:

- Gaceta
- Publicación
- Edición
- Aviso
- Evento
- Aplicación web
- Leaflet
- Vue.js
- Servicio Rest

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	1
2	Technological fundamentals	3
2.1	State of the art	3
2.2	Technologies used for the project	7
3	Methodology and project planning	9
3.1	Development methodologies	9
3.1.1	Artifacts	10
3.1.2	Roles on the Scrum Team	10
3.1.3	Events	11
3.1.4	Methodology support tools	12
3.2	Planning and monitoring	13
3.2.1	Resources	13
3.2.2	Initial Planing	13
3.2.3	Schedule tracking	14
3.3	Cost evaluation	17
4	Analysis	19
4.1	Requirements	19
4.1.1	Actors	20
4.1.2	Product backlog	20
4.2	System architecture	22
4.3	User interface	24
4.3.1	Components and navigation	24
4.3.2	Component Mockups	26

4.4	Conceptual data model	29
5	Design	33
5.1	Technological architecture of the system	33
5.2	Back-end	34
5.2.1	Entities (Domain)	35
5.2.2	Repositories	35
5.2.3	Services	37
5.2.4	Controllers	37
5.3	Front-end	40
5.3.1	Components	40
5.3.2	Routing	41
5.3.3	Communication with the server	43
6	Implementation and tests	47
6.1	Implementation	47
6.1.1	Get the number of times a place is referenced	47
6.1.2	Visualization of the places	50
6.1.3	Flow between locations	50
6.2	Testing	52
7	Solution developed	55
7.1	Use of the map	55
7.1.1	Displaying the information	55
7.1.2	Display the number of references.	55
7.1.3	Visualize the communications	55
7.1.4	Filter by a year or a range of years	57
7.1.5	View an animation	57
7.2	Authentication	60
7.3	Data Management	60
8	Conclusions and future work	65
8.1	Objectives	65
8.2	Lessons learned	65
8.3	Future work	66
A	Mockups	69
B	Glossary of Acronyms	75

CONTENTS

C Glossary of terms	77
Bibliography	79

List of Figures

2.1	List of the facsimiles on the Die Fuggerzeitungen website	3
2.2	Photos of the fascimiles on the Die Fuggerzeitungen website	4
2.3	Map of the Die Fuggerzeitungen website	4
2.4	List of the people on the six degree of Francis Bacon website	5
2.5	Map on the six degree of Francis Bacon website	6
2.6	Map on the Resource trade .earth website	6
2.7	Trade flows information on the resouce trade .earth website	7
4.1	System architecture diagram	23
4.2	Home prototype	26
4.3	Gazettes list prototype	27
4.4	Gazette form prototype	28
4.5	Gazettes detail prototype	29
4.6	class diagram	31
5.1	Diagram of the architecture with the used technologies	34
5.2	Gazette and news entities	35
5.3	Place and editor entities	36
5.4	Gazette DAO	36
5.5	Editor service	37
5.6	Place service	37
5.7	Gazette service	38
5.8	News service	38
5.9	Structure of a Vue.js component	41
5.10	Component diagram at the front-end	42
5.11	Communication of the common components from the Client to the Server . .	43
5.12	Communication of the user components from the Client to the Server (1) . . .	43
5.13	Communication of the user components from the Client to the Server (2) . . .	44

5.14	Communication of the user components from the Client to the Server(3) . . .	44
5.15	Communication of the user components from the Client to the Server(4) . . .	45
6.1	getList() function in PlacesRepository	48
6.2	ListAll() function in PlaceService	48
6.3	NumberplaceG function in GazetteDao	49
6.4	getSQLQuery function in GazetteDao	49
6.5	MapSource() function in Home component	51
6.6	Getproperties function in Home component	51
6.7	MapFlow() function in Home component	52
7.1	Map page	56
7.2	Map page combination of actions	56
7.3	Map pop-up	57
7.4	Map flow	58
7.5	Map flow (2)	58
7.6	Map filtered by a year	59
7.7	Map animation	59
7.8	Login page	60
7.9	Editors list page	61
7.10	Places list page	61
7.11	Gazettes list page	62
7.12	Gazette form page	62
7.13	Gazette modification page	63
7.14	Gazette detail page	63
7.15	Gazette detail page (2)	64
A.1	Login page prototype	70
A.2	Main page prototype	70
A.3	Editors list prototype	71
A.4	Editor form prototype	71
A.5	Places list prototype	72
A.6	Place form prototype	72
A.7	Gazettes list prototype	73
A.8	Gazette form prototype	73
A.9	Gazette detail prototype	74

List of Tables

- 3.1 human resources cost table 17
- 3.2 Planned project costs 18
- 3.3 Actual project costs 18

- 5.1 Component path table 42

Introduction

In this chapter we comment the factor that motivated the development of this project and on the goals to be achieved.

1.1 Motivation

Today we live in a society where it is no longer conceivable to live without technological devices and the Internet, since they are so integrated into our routines that we would not know how to act. We use them both at work and in our leisure time, because of the many advantages they offer us when it comes to simplifying tasks or keeping us connected, so we are always looking to improve our lifestyle with the help of technology.

This project arises from the idea of being able to help a group of philologists in their investigation to facilitate and reduce the time spent on some of the tasks in the research process. For this research, the group needs to be able to see the evolution in time of the publication of gazettes and their interaction with the places. There are other applications that provide some of the functionalities they need for the investigation, but they do not offer a correct solution for the requirements that the group of researchers needs.

Hence, the main objective of this end-of-degree project is to develop a web application that will allow them to store and manage in a simpler way the information about the gazettes, their news, editors and places to which they refer, at the same time it will show an interactive map where they can visualize the information they want in a more intuitive way about the places and their interaction.

1.2 Objectives

From the main objective of the project, creating a web application to manage and visualize historical gazettes in maps, we can define the following specific objectives:

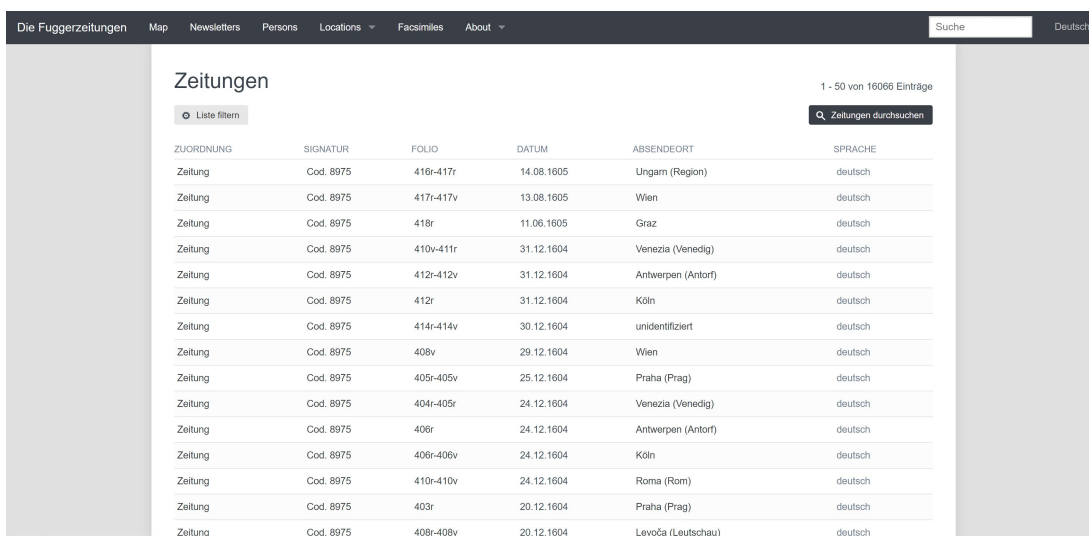
- To allow the **complete management of the basic data** of the application. This implies that an administrator-type user can register, modify and delete the data of gazettes, news, publishers and geographical places.
- To **associate the different relevant geographic places** with the **news and gazettes**. Specifically, the administrator-type user must be able to associate with each news item the place where the news occurs and where it is reported, and with each gazette the place where it is edited and the place where it is printed.
- Allow to **explore on a map the information of the gazettes**. To do this, an unauthenticated user must be able to view the news and gazettes on a map according to the type of location chosen and also filtering by year.
- **Display the networks** that are established with the news and the gazettes from the place where the news occurs to the place where it is printed. To do this, it is necessary to visualize the flow of news between the different geographical locations, also taking into consideration the time.
- **Provide an animation** where the users can see the evolution of the publication of the gazettes.

Technological fundamentals

In this chapter, we focus on existing tools that perform similar functions to the required objectives, but do not provide a complete solution to the problem. This part of the development provided a better understanding of how the project should be implemented.

2.1 State of the art

Die Fuggerzeitungen [1] is a tool to visualize information about facsimiles, handwritten newsletters in the 16th century. These bulletins contain news items that mention places and people involved with the event told in the news. The application displays information about facsimiles (see Figure 2.1), people and places in several separate lists. As we can see in Figure 2.2, it also allows you to display images of the pages of these newspapers .



ZUORDNUNG	SIGNATUR	FOLIO	DATUM	ABSENDEORT	SPRACHE
Zeitung	Cod. 8975	416r-417r	14.08.1605	Ungarn (Region)	deutsch
Zeitung	Cod. 8975	417r-417v	13.08.1605	Wien	deutsch
Zeitung	Cod. 8975	418r	11.06.1605	Graz	deutsch
Zeitung	Cod. 8975	410v-411r	31.12.1604	Venezia (Venedig)	deutsch
Zeitung	Cod. 8975	412r-412v	31.12.1604	Antwerpen (Antorf)	deutsch
Zeitung	Cod. 8975	412r	31.12.1604	Köln	deutsch
Zeitung	Cod. 8975	414r-414v	30.12.1604	unidentifiziert	deutsch
Zeitung	Cod. 8975	408v	29.12.1604	Wien	deutsch
Zeitung	Cod. 8975	405r-405v	25.12.1604	Praha (Prag)	deutsch
Zeitung	Cod. 8975	404r-405r	24.12.1604	Venezia (Venedig)	deutsch
Zeitung	Cod. 8975	406r	24.12.1604	Antwerpen (Antorf)	deutsch
Zeitung	Cod. 8975	406r-406v	24.12.1604	Köln	deutsch
Zeitung	Cod. 8975	410r-410v	24.12.1604	Roma (Rom)	deutsch
Zeitung	Cod. 8975	403r	20.12.1604	Praha (Prag)	deutsch
Zeitung	Cod. 8975	408r-408v	20.12.1604	Levoča (Leutschau)	deutsch

Figure 2.1: List of the facsimiles on the Die Fuggerzeitungen website

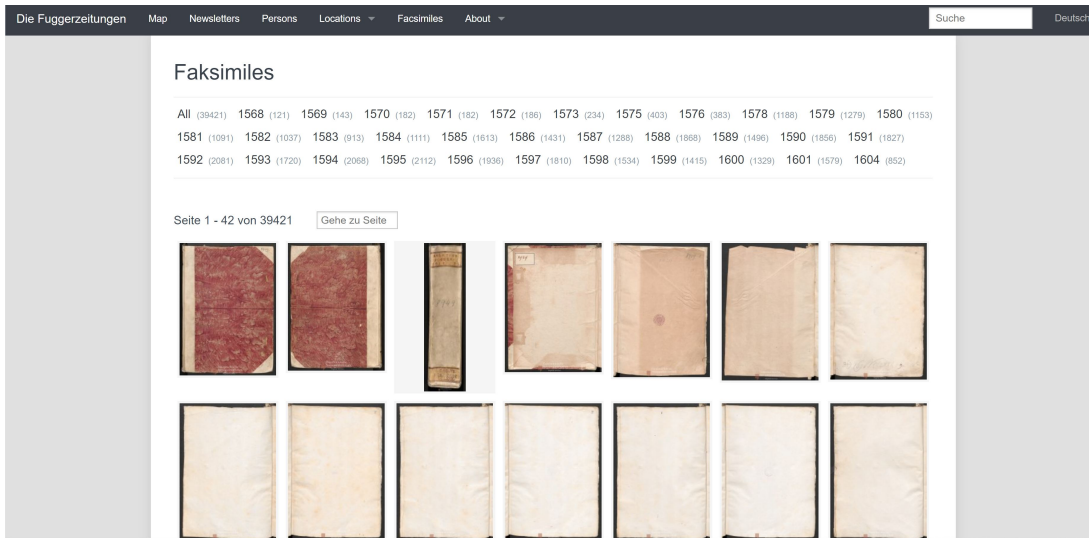


Figure 2.2: Photos of the facsimiles on the Die Fuggerzeitungen website

To visualize the places where the newspapers were distributed, the app shows a map (see Figure 2.3), which indicates them by displaying circles of different colours according to the number of dispatches that have been made in that place. The map allows you to modify the range of years over which the information is displayed and to change the focus of the map by zooming in.

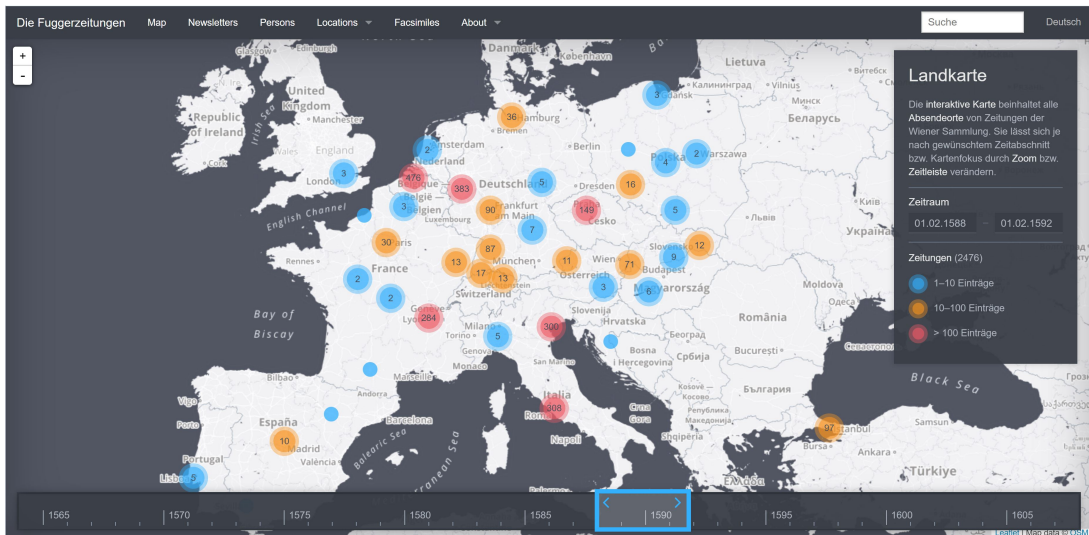


Figure 2.3: Map of the Die Fuggerzeitungen website

A problem with this tool is that it does not show the interaction between the places, that is, it does not show the communication that exists between the places where the gazettes were sent and the places where the news events occurred. On the other hand, I would like to

ID	Name	Historical Significance	Birth Date	Death Date	Gender	Created By	References	Action
10002047	Aaron Cappel	Reformed minister	IN 1560	IN 1620	male	ODNB_Adn		VISUALIZE
10005280	Aaron Guerden	physician and master of the mint	CA 1602	CA 1676	male	ODNB_Adn		VISUALIZE
10005944	Aaron Hill	writer and entrepreneur	IN 1685	IN 1750	male	ODNB_Adn		VISUALIZE
10010124	Aaron Rathborne	land surveyor and author	IN 1571	BF 1681	male	ODNB_Adn		VISUALIZE
10010233	Aaron Rhames	printer	AF 1624	IN 1734	male	ODNB_Adn		VISUALIZE
10011188	Aaron Smith	conspirator and lawyer	AF 1591	IN 1701	male	ODNB_Adn		VISUALIZE
10012966	Aaron Wilson	Church of England clergyman	IN 1588	IN 1643	male	ODNB_Adn		VISUALIZE
10010874	Abednego Seller	nonjuring Church of England clergyman and religious writer	IN 1646	IN 1705	male	ODNB_Adn		VISUALIZE
10001409	Abel Boyer	lexicographer and journalist	CA 1667	IN 1729	male	ODNB_Adn		VISUALIZE
10004026	Abel Evans	Church of England clergyman and poet	IN 1675	IN 1737	male	ODNB_Adn		VISUALIZE
10006880	Abel Ketelbey	barrister and politician	IN 1675	IN 1744	male	ODNB_Adn		VISUALIZE

Figure 2.4: List of the people on the six degree of Francis Bacon website

mention that the application is in German and although they offer an English translation it is not enough to be able to understand the interface properly.

Six degree of Francis Bacon [2] is a web application dedicated to showing the social networks of early modern Britain from 1500 to 1700. It stores all the necessary information about people and then shows all the relationships they had with each other on a map. On the website you can select whether you want to see a list of the people or the relationships and then you can click on one of them to see a map of all the interactions. The Figure 2.4 shows the list of people provided by the web application.

The map offered is a set of graphs (see Figure 2.5), but without showing the geographical location, so you can only see the interactions without knowing the space or time, which also causes it to be an unintuitive interface as it is difficult to understand the representation it shows. Another point to consider is that it does not allow the page to be displayed on small screens, so if you minimize the size of the page you can no longer see the map.

Resource trade .earth [3] is a web-based application that explores the fast-evolving dynamics of international trade in natural resources, the sustainability implications of such trade and the related inter dependencies that emerge between importing and exporting countries and regions [4].

As we can see in Figure 2.6, it displays an easy-to-understand map of geographical networks indicating locations and their trade relationships (see Figure 2.7). This map can be filtered by exporting countries, importing countries or even by the year in which we want to see the trade activities that took place. The problem you have is that it does not allow you to zoom in on the map. Since it cannot be zoomed in, it always shows the world map so if you want to see a country in detail it is not possible. This makes it sometimes difficult to see

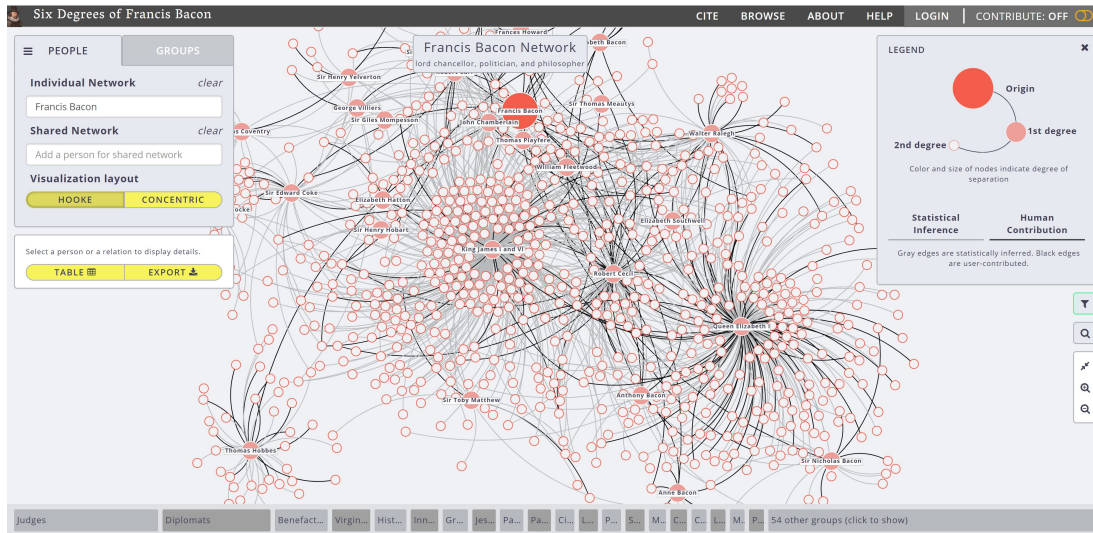


Figure 2.5: Map on the six degree of Francis Bacon website

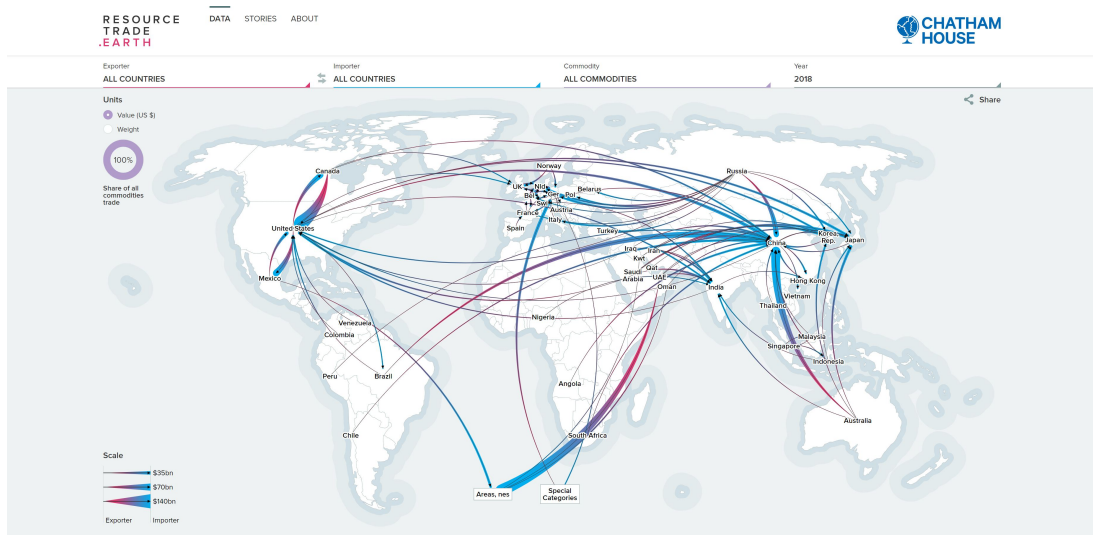


Figure 2.6: Map on the Resource trade .earth website

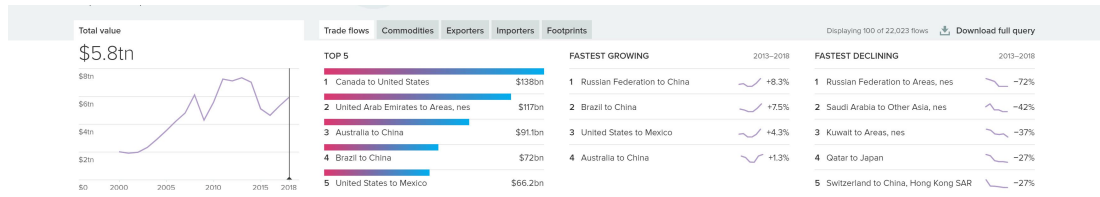


Figure 2.7: Trade flows information on the resource trade .earth website

which countries appear on the map.

After analyzing the tools mentioned above, we have found that none of them offers an optimal solution for this project. Some of the reasons are that the maps do not have all the desired functionalities, they don't show the flows between places, they don't allow to zoom, the filters they have are not the most suitable ones... Another reason is that two of the three applications are not dedicated to old newspapers, so their functionalities are not focused on them.

Even so, it has served us as inspiration and to get ideas about how the web interface should be. The first tool we commented on showed a map in which you could zoom in on the map and to indicate the places used coloured circles that changed depending on the number of shipments. Then, on the .earth application we can get the idea of the lines that represent the commercial relations between the countries and the type of filter used for the map was simple and easy to understand.

2.2 Technologies used for the project

In this section, we mention and briefly describe the technologies that have been used throughout the development of the project

- **Vue.js** [5] [6] [7] is a progressive framework for building user interfaces. One of the most important features is that Vue works with components, elements that encapsulate reusable code. The components allow us to develop modularized and easily scalable projects. Its core is quite small and it scales through plugins. It includes HTML, CSS and JavaScript in a single file.
- **Node.js** [8], is a development platform for the creation of web-based, network oriented applications and focused on speed and scalability.
- **CSS (Cascading StyleSheets)** [9] is a style language for defining and create the presentation of a web page (colours, layout and fonts). Normally it is used with HTML although both are independent.

- **HTML (HyperText Markup Language)** [10] is the standard markup language to describe the meaning and structure of web pages.
- **JavaScript** [11] is a programming language used in Web pages to increase the functionality of them and the interaction with the user.
- **Leaflet.js** [12] is an Open-source JavaScript library for interactive maps creation and visualization.
- **Java** [13] is a general-purpose programming language that is class-based, object-oriented, multi platform and designed to have as few implementation dependencies as possible.
- **Spring Boot** [14] [15] is a framework that facilitates the configuration of a Spring project, making it automatic.
- **Hibernate** [16] is an ORM (Object-Relational Mapping) framework, that is, a tool for matching objects and tables in a relational database.
- **Hibernate Spatial** [17] is an open-source generic extension for handling geographic data in Hibernate.
- **Geotools** [18] is an open source Java library that provides tools for geospatial data.
- **PostgreSQL** [19] is a free open-source and object-oriented relational database management system. It features multi-version concurrency control (MVCC) and the use of read locks.
- **PostGIS** [20] is a spatial database extension for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.

Methodology and project planning

In this part of the document, we explain the development methodology that has been carried out for the execution of this project and the tools used to apply the method. Then, the planning performed will be commented together with the deviations and costs.

3.1 Development methodologies

To perform this project, it was decided to use an incremental agile methodology [21]. These methodologies are characterized by allowing to adapt the way of working according to the conditions of the project and to achieve flexibility and immediacy in the response to adjust the project and its development to the specific circumstances of the environment. In other words, they allow a quick adaptation to change and continuous refinement of requirements, as well as error detection.

This type of methodology is also characterized by dividing the development of the project into iterations, phases whose final objective is to obtain functional products by performing analysis, design, implementation and testing tasks, which constitute the typical phases of a traditional development cycle. With each of these iterations the product is refined until the desired final product is obtained. For this, before starting an iteration, the work to be done in that period is planned. This is an advantage since it is easier to estimate and planning than in the traditional methodologies, where the whole process of the project was planned in advance, which prevents adapting the planning to unexpected events.

The agile methodology that has been used for the development process is the one known as SCRUM [22], a framework that is used to manage complex projects in a simple way. The characteristics of this methodology are that it is applied in small work groups where they collaborate. Since this is an individual project, it has been necessary to adapt this work method to the characteristics of the end of degree project, so it has not been possible to apply all the good practices that it proposes.

In this methodology the requirements are described as User Stories in the product backlog. Unlike in traditional methodologies where use cases are only used and these ones are more elaborated descriptions of an action or activity accompanied by a use case diagram, where you can see the activities related to the actors who will perform them.

User stories [23] are the basic element for the creation of products in agile methodologies. They consist of short and simple descriptions of a project feature told from the perspective of the person who needs that functionality; usually from the point of view of a user or customer of the system. This type of tool follows a template phrase: As <role> I want <event> for <functionality>.

To show how this methodology was adapted it is necessary to first explain the main elements that form it: the artifacts, the work roles and the events and then comment on how they have been used in the project [24].

3.1.1 Artifacts

Artifacts are a set of physical elements that represent work and value in different forms, that is, they are the resources that establish the productivity and quality of any project. The main Scrum artifacts are:

- **Product Backlog:** It is the list of all the requirements that the final product must meet in order to achieve the desired objectives. These requirements are represented by user stories and are ordered according to their priority over others. These priorities are not fixed and may change throughout the development or even new requirements may appear.
- **Sprint backlog:** These are the user stories of the backlog product that have been planned during a sprint.
- **Increment:** Result of a sprint, it is the sum of all the elements finished during a sprint plus the value of the previous sprint increments.

The artifacts described above have all been used in this project, the list of the backlog product is described in section 4.1.2 and the use of the sprint backlog and increments in section 3.2.3.

3.1.2 Roles on the Scrum Team

The scrum team refers to all team members working on the project. They are self-organizing teams that do not depend on people who are not part of the team, and are designed to optimize flexibility, creativity and productivity.

- **Product owner:** He is responsible for maximizing the value of the product to be implemented by the development team. He is the client's representative within the team and the one in charge of identifying the project's requirements, so he is the one in charge of adding and sorting the backlog product's user stories.
- **Scrum master:** This person ensures that the Scrum framework is developed and applied correctly, is the leader who is responsible for removing obstacles and ensuring that the team meets the time and objectives set.
- **Development team:** A structured team that organizes its own work, they are in charge of transforming the tasks into development increments.

In this project the roles of scrum master and product owner was performed by the mentors as they were the ones who knew best the client's requirements. The figure of the development team has been played only by the author of the work, organizing the sprint backlog of each iteration and developing the project. As part of the learning process, the author of the work has taken also part on the roles of product owner and scrum master.

3.1.3 Events

Milestones of the scrum process that allow the development of the product.

- **Sprint:** These are the iterations in which the project is divided, normally they are of short duration, in this case the sprints lasted two weeks and in total, seven sprints were made, obtaining in each one a functional product.
- **Sprint planning:** In this meeting the objectives of the next sprint are defined, and the questions are answered: What is going to be done? and How is it going to be done?
- **Daily scrum:** Daily meetings of the development team where they explain what they have done the previous day, the problems they have had and what they are going to do that day. In this case, the team is formed by only one person, so this task has not been carried out as a meeting, but as a daily planning.
- **Sprint review:** In these meetings, the product obtained in the sprint is shown to the product owner and the development team shows how it works. The product owner must provide feedback on the assessment of the changes made and whether it meets the expected objectives. This kind of meetings were held together with the sprint meetings, in order to reduce time.

3.1.4 Methodology support tools

To perform the different tasks related to the project, the following tools have been used:

- **GitLab:** [25] Distributed version control and software development tool. This web service has been used mainly to apply the SCRUM methodology. Three projects were created, one for documentation, one for the client, and finally one for the server. This separation allowed better management of the user stories represented in GitLab by Issues, since the same requirement was created as two issues, one for the client project and another for the server. For the overall project which includes the three mentioned above, a milestone was created for each sprint, grouping the completed issues and the time spent on them. When developing the project, we worked with a workflow based in GitFlow, this means that the work is organized in branches. The main branch is called the Master branch where everything that is uploaded in it must work correctly, and then we have the secondary branches created from each issue, they are the ones where the code will be modified to add or modify features to get the desired result.

The common procedure of this method consists in opening the issue, creating a branch, making the necessary changes and when the implemented functionality is finished and works correctly, making a merge request that consists in merging the code of the branch with the master code and finally closing the issue. Therefore, the work done for each feature will go into its own branch. This allows to have a clear separation of the code during the development and if there is an error to know in which moment it occurred.

- **Toggle:** [26] Web application that allows to time the duration of the project tasks and generate a calendar with the time dedicated to each one of them.
- **Maven:** [27] Tool for the management and construction of Java projects. It uses a Project Object Model (POM) to describe the project to be built, its dependencies with other modules and external components.
- **Draw.io:** [28] Web application that allows to make a great variety of diagrams, as well as mockups.
- **Eclipse:** [29] Integrated Development Environment (IDE) used for the server's Java development.
- **Visual Studio Code:** [30] Code editor used for client development in Vue.
- **Latex:** [31] Text composition system oriented to the creation of written documents used to write the end-of-degree project report.
- **starUML:** [32] UML diagramming software.

- **Web browsers:** that allowed the execution of the web client (especially Google Chrome).

3.2 Planning and monitoring

In this section, we will show the initial planning and the project estimations. A comparison with the final result will be made to find out the deviations that have occurred and the cost of the project.

3.2.1 Resources

The resources used to carry out this project are divided into two groups: human resources and material resources:

Human resources

This project has been developed by a team of 3 people who, in accordance with the SCRUM development methodology, played the roles of Product Owner, Scrum master and Development Team. In a more detailed way we can say that the roles of Product owner and Scrum master were performed by the mentors; and the roles of developer and analyst by the end-of-degree author.

Material resources

- Dell personal laptop: Device where the development of the whole project was done.
- Software resources: correspond to those mentioned in section 3.1.4

3.2.2 Initial Planing

At the beginning of this project a preliminary analysis was made to know the objectives of the project, for this purpose the following tasks were carried out:

- A **study of the technology** was carried out to know which tools would be best suited for the development of the project. Once the technologies were selected, we proceeded to prepare the work environment by installing the selected tools and to study them in depth in order to know how to use them once the implementation of the project started.
- The initial **user stories** that would form the initial product stack were decided upon based on the application requirements that were refined throughout the project.
- The **data model** was designed to define what information would be stored and in what form.

- The **prototypes** of the application were developed based on the user stories, in order to define the web interface and to have a better understanding of how the project should work.

Also in this phase, the duration of the sprints was decided taking into account the delivery date. The final decision was that the duration should be two weeks, as it was considered sufficient time to be able to implement the new features and get feedback on it, since sprints of shorter duration would not allow to implement enough material to test it in the review meetings and longer iterations would cause the number of implemented features to increase but at the same time it would increase the risk of not being able to have enough time to correct a bug or a bad implementation that had been detected in the sprint reviews.

So, in the end the development of the work would be done in seven sprints of two weeks each, allowing to have two weeks to write the report.

3.2.3 Schedule tracking

In this section, we comment on the work done for each sprint detailing the tasks performed and the deviations that have occurred throughout the development over the initial planning. At the beginning of each sprint, a planning meeting was held, so the results of the previous iteration were known, allowing to organize and act accordingly. The agreed time of dedication for the work in each sprint would be as follows:

- Development and analysis work done by the author: 5 days a week with 5 hours for each.
- Dedication of 1 hour of the entire project team in each sprint review.

The individual work done for each sprint with the corresponding tasks will be detailed below. Following the applied Scrum methodology, the plan for each iteration was made in the planning sprint meetings, which were done at the end of the previous sprint and before the start of the next one, so the result of the previous iteration was known and could be acted consequently.

SPRINT 1

In this sprint, the skeleton of the project was defined and created, to which new functionalities were later added throughout the development. We also started with the implementation of the basic operations for the management of the information of the editors. The tasks carried out during this time were:

- Implementation of authentication and logout operations.

- Implementation on the server of the operations to create new editors, modify their information and be able to delete them.
- Implementation of the necessary operations to be able to obtain a list of all the editors stored in the database.

In this iteration it was possible to complete in time all the tasks related to the user stories that had been planned to be implemented.

SPRINT 2

During this time, the implementation of the basic functionalities continued but this time, those in charge of managing the information on the locations:

- Implementation in the server of the operations to create and delete places.
- Implementation of the necessary operations to obtain a list of all the places stored in the database.

In this case, it was not possible to implement the functionality of modifying the information of the places, since it was not taken into account that at this stage the leaflet library was going to be used for the first time in the project and the duration of the activities was not well estimated since, as the technology was not very well known, the development of the activities increased.

SPRINT 3

The following activities were carried out during this sprint:

- Implementation of the operation to modify places.
- Implementation of the necessary operations to obtain a list of all the gazettes stored in the database.

During this period of time it was possible to finish the functionality that was planned to be done in the previous one by increasing the time dedicated to the project.

SPRINT 4

During this time, the basic operations on gazette information management were completed.

- Implementation of operations to create, modify, delete gazettes.
- Implementation of operations to create, modify and delete news.
- Implementation of the operation to modify places.
- Implementation of the operations to be able to visualize the information in detail of a gazette and its news.
- To filter the gazettes by place of publication, edition and publishers.
- Two functionalities that were implemented previously were modified, those of eliminating places and editors. Now if they have a gazette or news associated can not be removed, because it does not interest to delete that information in the gazettes, as it is important to know the places where the actions happened.

As you can see, the basic operations on information management have been completed and some functionalities previously implemented have been corrected so that they offer a correct operation and avoid errors. In this sprint all the planned work could be done in time.

SPRINT 5

In this increment of the project we managed to add the main functionality of the application which is to be able to visualize the information on the map.

- Confirmation window when you want to delete some information.
- Implementation of the map and the markers that indicate the places where some kind of activity has taken place.
- Implementation of the pop-up that shows the number of references of each type in the places (events, announcements, editions or publications).

SPRINT 6

New map functionalities continued to be implemented:

- Implementation of the map legend.
- Implementation of filtering by years on the map.

During this increase all planned activities were carried out, although their planning was more complicated because it coincided with the preparation of a university exam, but even so it was possible to maintain the same hours of dedication.

SPRINT 7

The following activities were implemented in sprint 7:

- Implementation of the flow between locations.
- Implementation of the animation where you can see the evolution in time of the activities in the places.
- Implementation of search filters in the list of editors and the list of places.

With these last implementations the project development was finished.

At the end of a sprint, as mentioned above, reviews of the new features were made; as a result of these meetings, requests were made to change some of the implemented functionality in order to improve the interface and functionality of the application. These changes are explained in the testing section 6.2.

3.3 Cost evaluation

In order to obtain the total costs of a project, the costs of human resources and material resources must be considered.

In this case, the material costs are not taken into account, since the equipment used is personal and no software tool that required a license has been used.

As for the human costs, it was necessary to take into consideration the weekly hours dedicated to the project, the duration of the project and the costs of each of these resources. The table 3.1 shows the salaries/hours of each worker.

Employee	Costs
Miguel Luaces	27€/h
Alejandro Cortiñas	27€/h
Carmen Corrales	17€/h

Table 3.1: human resources cost table

If this project was actually carried out project by a company, indirect costs would have been taken into account, such as the cost of the company's office rent, the Internet connection

cost, the electricity cost, or the support staff cost. However, as the value of these costs is specific to the company in which the project is developed, we do not take them into consideration in this project.

After knowing all these facts, it was possible to calculate the planned costs of the project and the actual costs, to know the deviations that occurred.

Employee	Hours	Cost
Miguel Luaces	10	270€
Alejandro Cortiñas	14	378€
Carmen Corrales	400	6800€
Total	424	7475€

Table 3.2: Planned project costs

Employee	Real hours	Cost
Miguel Luaces	10	270€
Alejandro Cortiñas	14	378€
Carmen Corrales	450	7650€
Total	474	8296€

Table 3.3: Actual project costs

In this section, we explain the analysis of the application, describing the requirements and actors, the system architecture, the user interface, and the data model by a class diagram.

4.1 Requirements

The requirements are properties that a product must fulfil in order to satisfy the customer's need. They will describe the behaviour of the system. There are two type of them:

- **Functional requirement:** this type of requirement defines a function of a system; they refer to the description of the activities and services that a system must provide. Normally these types of requirements are linked to the inputs, outputs of the processes and the data that has to be stored in the system.
 - **User authentication:** users will be able to log in to the web application using their credentials.
 - **Display information:** the application offers the possibility of being able to see the information about the gazettes, places and editors stored in the system.
 - **Manage information:** New data can be added, changed or deleted about the gazettes, places and publishers.
 - **Display a map:** An interactive map will be shown with the information mentioned above and the users will be able to manage it to visualize the data in a more intuitive way.
- **Non-functional requirement:** the non-functional requirements are those that refer to the characteristics that the program should have to improve the user experience such as performance, ease of use, security...

- **Easy use:** The application must be intuitive to make it easy for users to use so they do not get frustrated using the application.
- **User-friendly interface:** The application must be attractive so that it pleases the users and improves the user experience.
- **Security:** The application must provide a user authentication system for ensure that the data cannot be modified by anyone.

4.1.1 Actors

An actor is any entity external to the system that interacts with it and that demands functionality from it. In this project there are two of them:

- **Unauthenticated user** can visualize the data by interacting with the map, this type of user cannot access or modify the information. They also can access to the login page.
- **Authenticated user** manage the application data, they can add, modify or delete the information about the gazettes, editors or places.

4.1.2 Product backlog

Once the analysis of the application started, the product backlog was created with the general requirements described as user stories. These were refined throughout the development at the scrum meetings and the result is the list below.

1. As a user I want to be able to see a list with all the Gazettes that I have stored.
2. As a user I want to be able to filter the list of the Gazettes by editor.
3. As a user I want to be able to filter the list of Gazettes by place.
4. As a user I want to add new Gazettes and at the same time add the news that the Gazette contains.
5. As a user I want to be able to view the information of a specific Gazette and also its news.
6. As a user I want to be able to eliminate the information of a Gazette and also its news.
7. As a user I want to be able to edit the information of a Gazette.
8. As a user I want to be able to see a list with the information of the editors that the app has stored.

9. As a user I want to be able to add new editors.
10. As a user I want to be able to modify the data of the editors.
11. As a user I want to be able to eliminate the information of an editor.
12. As a user I want to be able to modify the data of the news.
13. As a user I want to be able to remove one specific news.
14. As a user I want to be able to see a list with all the places.
15. As a user I want to be able to add new places.
16. As a user I want to be able to modify the information of the places.
17. As a user I want to be able to eliminate places.
18. As a user I want to view the home page.
19. As a user I want to be able to log in.
20. As a user I want to logout.
21. As a user I want to be able to see a general map of locations in which the size and colour of the marker of each location is the number of times that the location is referenced.
22. As a user I want to see on the screen a map legend indicating the meaning of the size and colour of each location.
23. As a user I want to be able to see a popup when I click on a location. The popup will show the number of references of each type in the location (events, announcements, editions or publications).
24. As a user I want to be able to filter the locations that are shown on the map and display only those that are referenced as a specific type (events, announcements, editions or publications).
25. As a user I want to be able to select a range of years by which to filter the locations shown on the map.
26. As a user I want to be able to see the news flow, i.e. the system will show the number of times the location on which I have the mouse is connected to other locations. That is, for each time the location Madrid appears referenced, I want to discover and add which other locations appear referenced in the same news and in the gazette that contains

it. The number of times has to be shown as an arc connecting the locations whose color and thickness depends on the number of times. If you filter the locations by type (history filter the locations) all the locations that are the target of the news flow should still appear, not only those of the type indicated in the filter. That is, the filter only applies to the source of the flow, not the target.

27. As a user I want to be able to filter the news flow by selecting how a location should be referenced to be counted as the source or destination of the news flow.
28. As a user I want to be able to display an automatic animation of the map status for each year of the selected interval.

4.2 System architecture

This section will describe the system architecture with a high level of abstraction by performing a high-level decomposition of the system components but without yet mentioning the technologies that support them.

As we can see in [Figure 4.1](#) the model used is the one known as client-server. This type of architecture is characterized by being formed by two main components; one known as server (back-end) and the other as client (front-end). The client is in charge of making requests to the server and the server is in charge of satisfying the demands of the clients and processing and storing the data. Thanks to this architecture the tasks can be distributed between the server and the client, which allows the system to be scalable, that is, the capacity of clients and servers can be increased separately, with easy maintenance.

The client (front-end) is the software in charge of showing the user interface that allows the visualization of the information and the interaction with the client. It is responsible for collecting the input data introduced by users, transforming it into the appropriate format required by the server and sending the data to it. It also receives the responses from the server and displays them to the user in an understandable way by updating the interface itself. All this is possible because the client knows only the Rest interface that the server offers, this allows the application to have a low coupling architecture, in other words, it is scalable and easy to maintain.

The server (back-end) is the part of the software that provides a set of services to the client through the REST interface. It makes sure that the application works correctly and for that purpose it processes the requests that come from the front-end interface and sends back to the client answers for each one of them with the necessary information. He is also the one who makes the requests to the database.

The structure of this component is in turn made up of layers. This type of organization

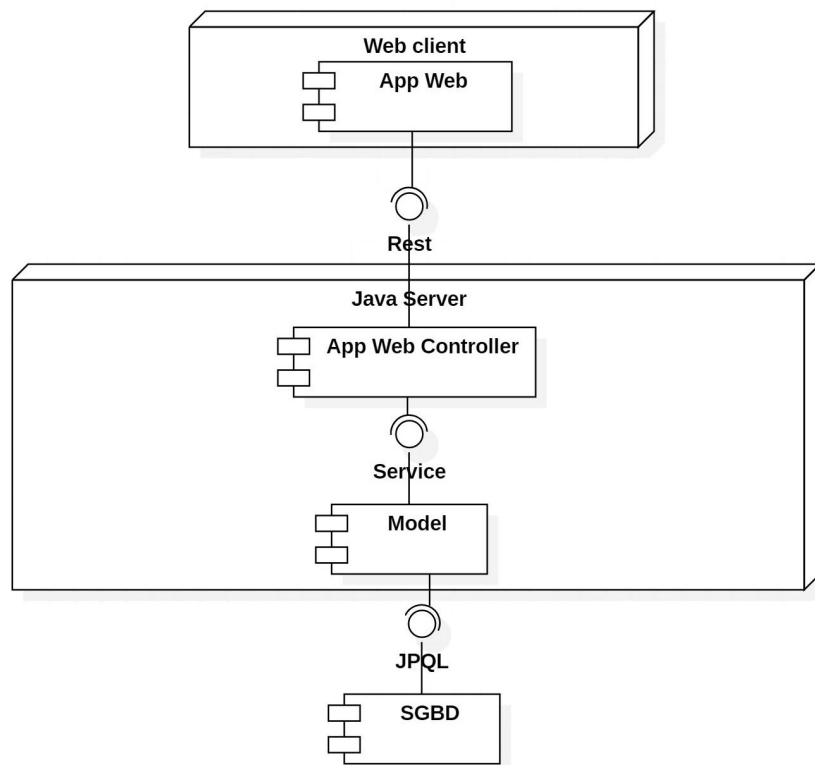


Figure 4.1: System architecture diagram

allows each of the levels to focus on one functionality without considering how the others are implemented. This has an advantage because changes in any one layer do not affect the others. Each of the lower layers provides a service to the upper one, i.e. each layer only communicates with the adjacent ones.

The back-end as mentioned above can be broken down into two other layers:

- **Controller Layer:** It exposes a set of access points (endpoints) that can be invoked by external processes. Endpoints can be defined as operations that receive parameters and return a result. Therefore, the services layer is the one that receives the requests from the client layer and delegates them to the model layer in order to perform them. In this case the layer implements a Rest service, an HTTP-based style of architecture. It uses the basic methods of the HTTP protocol (GET, POST, PUT and DELETE) to send requests. The exposed interface is detailed in section 5.2

- **Model Layer:** It is the set of classes that implement the business logic of the use cases, usually divided into two sub-layers.
 - **Business Logic Layer:** Implement business logic by using the data access layer to read/write the data.
 - **Data Access Layer:** It enables access to databases and provides the top layer with operations to read and store data.

4.3 User interface

In this section, the user interface will be described at a high level by discussing the general structure of the application's screens and the navigation structure between them.

For the design of the interface was obtained by a design oriented to computer screens (Desktop-first), since the use of this web application will be mostly in this type of devices with large screens.

4.3.1 Components and navigation

This section will explain the important components that compose the front-end and the navigation between them.

Component	Description
Home page	This is the main component where the interactive map is displayed, showing the places where a gazette was published or edited and where a news event or notice occurred. The map has some filters with which you can adjust the display preferences of users.
Login	Component used for the login. It is accessed by pressing the “login” button in the top menu.
EditorList	This component displays a table with the information of the editors. If you press the button add a new editor or modify the information of an editor, the “EditorForm” component will be loaded.
EditorForm	This component allows you to add a new editor or modify the information of the existing ones in the database. In this component if you press the accept or cancel buttons you will return to the “EditorList” page.
PlaceList	This component displays a table with the information of all the places. If you press the button add a new place or modify the information of an place, the “PlaceForm” component will be loaded.
PlaceForm	This component allows you to add a new place or modify the information of the existing ones in the database. In this component if you press the accept or cancel buttons you will return to the “PlaceList” page.
GazetteList	Component that shows a list with the information of all the gazettes. If you want to add new gazettes or modify any of them, you can press the button and it will change to the “GazetteForm” component, if you press the display button it will change to the “GazetteDetail” component.
GazetteDetail	Shows the information in detail of a certain gazette and its news.If you click on the button to modify the data of a gazette, the “GazetteForm” component will be loaded, if you click on the button to modify or add a new news item, it will be changed to the “NewsForm” component.
NewsForm	Component that allows the creation of new articles or the modification of the existing ones. If you press the accept or cancel button you will return to the “GazetteDetail” component.

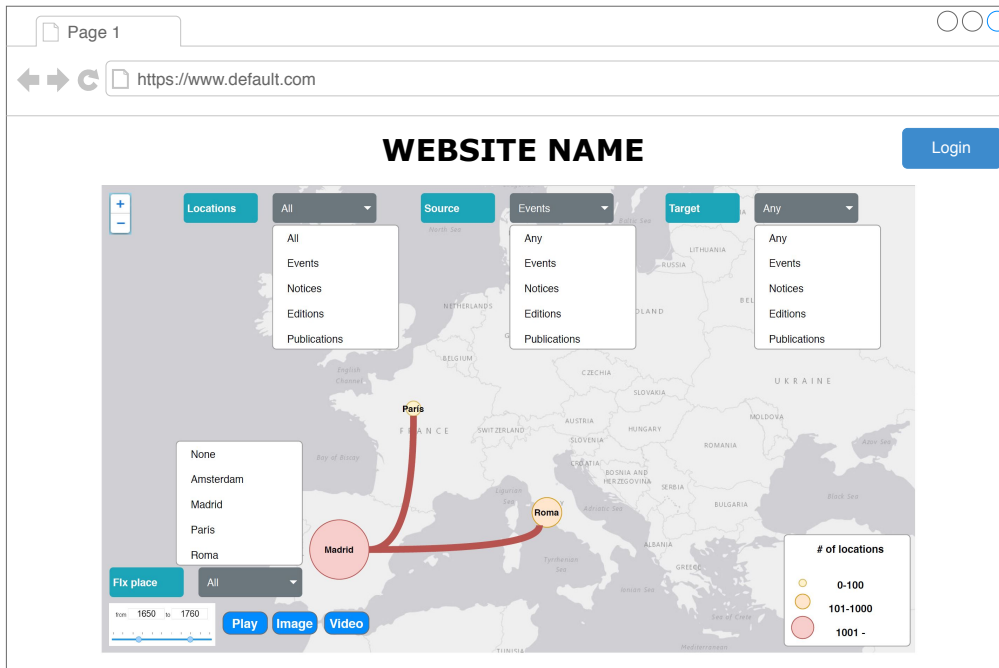


Figure 4.2: Home prototype

4.3.2 Component Mockups

This section will show the screen prototypes designed for the most important components. The mockups served to have an approximation of how the user interface had to be and thus, facilitate its implementation; also allowed to define new requirements that had not been considered and refine the existing ones.

The Figure 4.2 belongs to the "Home" component. This component will be the main page of the web application since it will contain the map that will allow the philologists to study and visualize the information about the relationship between places and gazettes. The map will show the places where gazettes have been published and edited and where news events and announcements have taken place. To be able to control the elements that are displayed on the map, a filter has been added that allows you to select whether you want to see only the places where one of the previous actions has taken place, so that it does not always show all the places together and allows a better view of the information.

In order to indicate the places, the map shows some circles over them. These circles will change in size depending on how many times an action or a set of them has happened in a

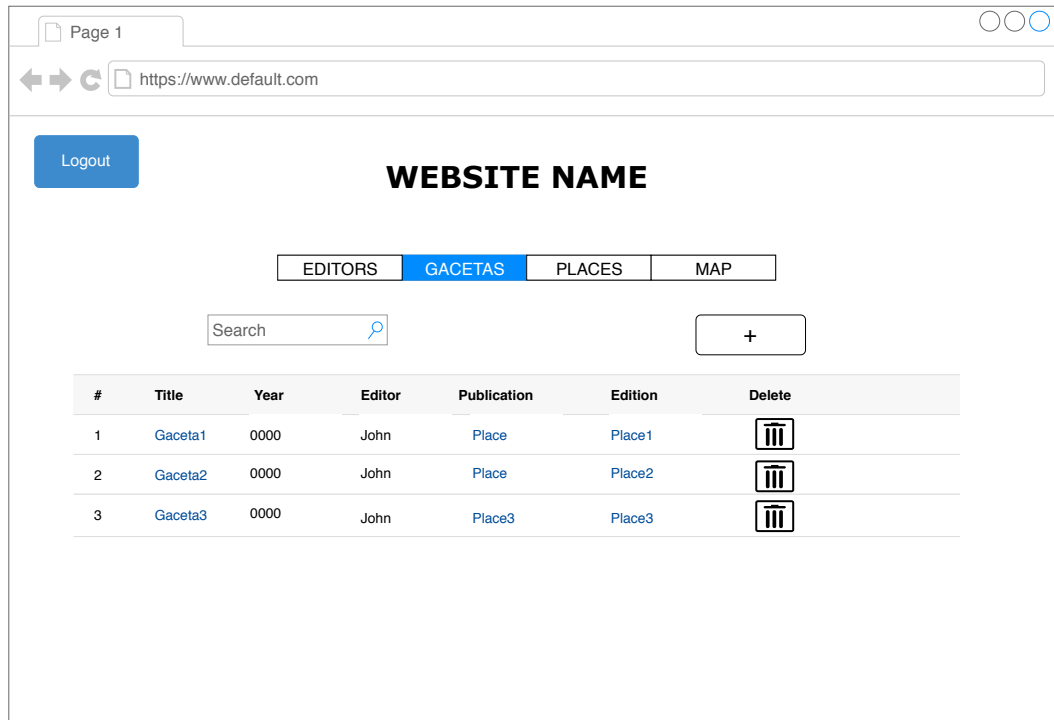


Figure 4.3: Gazettes list prototype

location, those that have experienced more activity will be reflected on the map with larger diameter circles. If you want to see how many times an activity has happened in a certain place you should click on the circle and a pop up will be shown showing the times it has been published, edited, noticed and occurred an event in that place.

One of the requirements of this web application was to be able to see the connections that would exist between the places, that means to see a place where a gazette had been published, the place where it was edited, and those places where the news was produced and where it was notified, because as it had been mentioned before, these places could be different. So in order to see the relations, you will have to choose in the filters the source action (publication, edition, notice or event) and the target action. In this way the two types of places will be shown in the map and if you position the mouse over one of the circles you will see some lines that will connect the places. These lines will change in thickness depending on how many times these two places have been communicated. Also, at the same time you can filter by a range of years or just one year to see in more detail what happened during the chosen time period.

The image shows a web browser window with a single tab labeled 'Page 1'. The address bar contains 'https://www.default.com'. The main content area displays a website with the title 'WEBSITE NAME'. Below the title is a horizontal navigation menu with four items: 'EDITORS', 'GACETAS', 'PLACES', and 'MAP'. The 'GACETAS' item is highlighted in blue. Below the navigation menu is a form titled 'New Gaceta' with a blue header. The form contains several input fields: a text field for 'Title' with the value 'Gaceta 4', a dropdown for 'Year' with '1927', a dropdown for 'Editor' with 'Editor1', a dropdown for 'Edition' with 'Madrid [Madrid (Spain)]', and a dropdown for 'Publication' with 'Madrid [Madrid (Spain)]'. At the bottom of the form are two blue buttons: 'NEXT' and 'CANCEL'.

Figure 4.4: Gazette form prototype

The [Figure 4.3](#) represents the “GazetteList” component, which shows the list of all gazettes. The information you will see for each one is: the title, the year of publication, the name of the person who wrote the gazette, the name of the place of publication and the place of edition. In this view you can add new gazettes by clicking on the add button and you will be taken to the “GazetteForm” component, which is the form where you can add the new information (see [Figure 4.4](#)). Also from this component you can modify the information of any gazette, for this you must press the edit button and it will take you to the “GazetteForm” component, but in this case it will contain the information in editable mode of the selected newspaper. Other actions that can be done in this view will be to delete the gazettes and to be able to see them in detail.

The detail view of a gazette corresponds to the “GazetteDetail” component and is where you can see all the information including the news of each newspaper (see [Figure 4.5](#)). This view will allow you to delete a gazette or modify its information and you will also be able to manage the information of the news, that is, you will be able to add new news, modify the existing ones or delete them. In order to add a new news item, you must press the add button

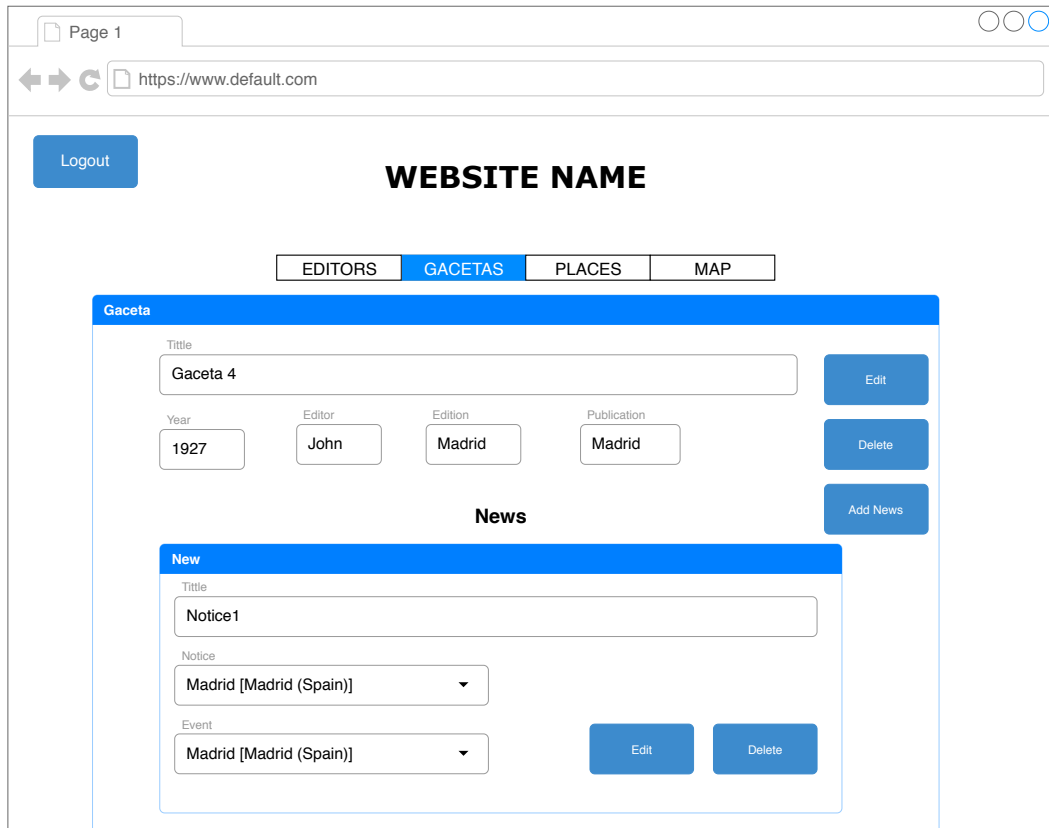


Figure 4.5: Gazettes detail prototype

and the “NewsForm” component will be loaded. This same component will be used to modify the information of the news.

4.4 Conceptual data model

During the analysis of the project, the database was also designed in order to subsequently facilitate the implementation of the project. To this end, the conceptual model of the database was developed, specifying the entities, their attributes and the relationships that exist between them.

- **Editor:** Stores information about the editors of the gazettes. There may be editors that are stored in the database that have not yet been related to any gazette, so the Editor-Gazette relationship has a cardinality of 1..1/0..* indicating that an editor may have written multiple gazettes but a gazette only has one editor.

– *Id:* editor ID.

- *Name*: editor name.
- **Place**: Stores information about the places
 - *Id*: place identifier.
 - *Name*: name by which the place was known in the 17th century.
 - *CurrentName*: current location name.
 - *Country*: name of the country of the place.
 - *Location*: coordinates of the place.
- **Gazette**: Stores the information about the gazettes
 - *Id*: gazette ID.
 - *Title*: title of the gazette.
 - *Year*: year in which it was published.
 - *Country*: name of the country of the place.
 - *Location*: coordinates of the place.
- **News**: Stores information about the news items. The gazettes store a group of news, but there are gazettes that may not have news stored so the relationship established between the two has a cardinality of 1..1/0..*, since a gazette can contain multiple news but a news only belongs to one gazette.
 - *Id*: news identifier.
 - *Title*: title of the news item.

In reference to the publication, edition, event and notice relations say that the gazettes contain a newsgroup and each of these news items describes events that may have occurred in one place but could have been reported in a different one. The same is true for the place of publication and editing of newspapers.

Therefore, each of these relations explains:

- **Publication**: the place where a gazette was published.
- **Edition**: the place where a gazette was published.
- **Event**: Place where a news item occurred
- **Notice**: Place where notice was given

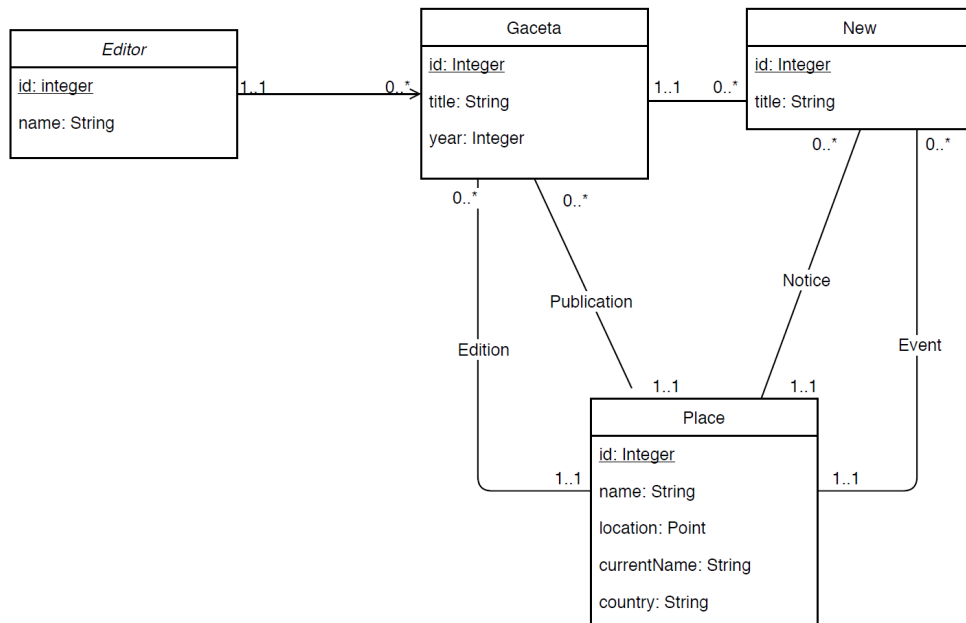


Figure 4.6: class diagram

All these relations have a cardinality of 1..1/0..*, since all of them must refer to only one place, but in one place multiple publications, events, editions or news may have occurred. At the same time there may be places that do not refer to any gazette or news, for example when you create a new place it is not related to any gazette or news.

Chapter 5

Design

In this chapter, we show the architecture of the system along with the technologies used and we give a detailed description of the design of the architecture to complement what is set out in chapter 4.

5.1 Technological architecture of the system

In this section, we describe and explain the technologies that were used for the development of the project. The following [Figure 5.1](#) is the enhanced diagram of the section 4.1

As we can see in the lower layer of the diagram, the database will use the PostgreSQL application, an open-source, object-oriented relational database management system, to store the information. As in this project, it is necessary to store the geographic information of the places so a module called PostGIS has been added to PostgreSQL that adds support of geographic objects in order to convert the database into a spatial one. For the connection to the DDBB with the upper layer, JPA is used which is an API that allows the execution of operations on the database and to maintain the persistence of the objects using JPQL (java persistence query language) as a query language based on SQL.

The model layer is formed by the repositories and services. The DAOs (Data Access Object) are used for the connection between the model layer and the server. A DAO is a design pattern that allows encapsulating the data access logic to the rest of the application, in other words, it separates the business logic from the logic to access the data by providing an interface with the necessary methods for the service to interact with the data source. The services implement the business logic thanks to the persistent data read and write operations offered by the repositories. These services are implemented using Hibernate, a framework that acts as an object-relational mapper [hibernate].

In order for the server to connect to the Web client, a Rest service was implemented, which will communicate with the client by sending the requested information in JSON format. The

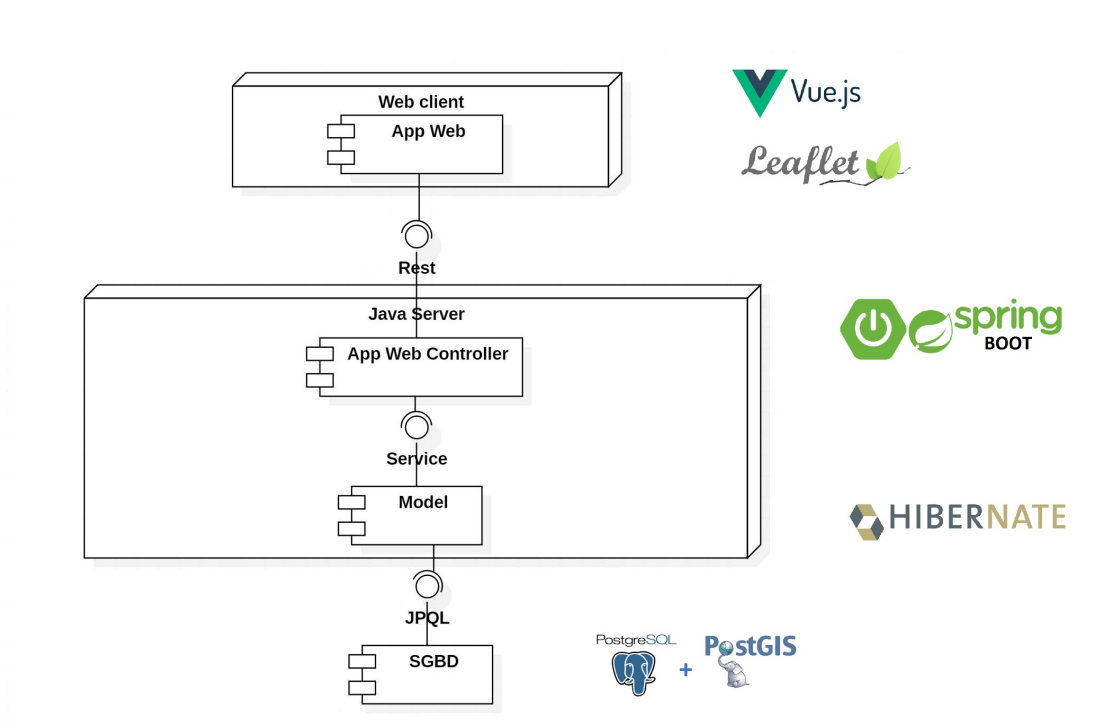


Figure 5.1: Diagram of the architecture with the used technologies

Rest interface has been implemented by using the annotations offered by spring to map the URLs with the HTTP methods and thus know what operation should be processed with each request. This client is implemented with Vue.js and the Leaflet library to create the interface and the interactive maps.

5.2 Back-end

This section detail the internal structure of the back-end. This is the part of the application that is not visible to the client. The back-end implements the functionality of the application and is composed of the persistence layer, the data access layer, the services layer and the controllers. The project contains the `src/main/java` package which is where the Java classes are located. All of them together provide the server-side functionality. In turn, it contains the following hierarchical package structure:

- **Domain:** Package where the application entities are stored.
- **Repository:** Package that includes the DAOs of the project.
- **Security:** Package that contains the classes in charge of the security of the application, it is also in charge of the users' authentication.

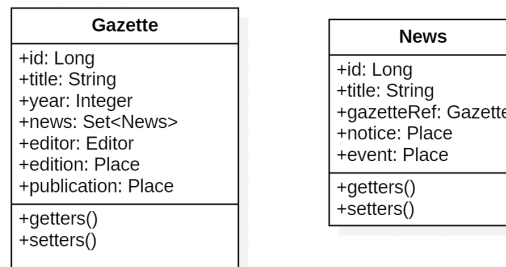


Figure 5.2: Gazette and news entities

- **Service:** Package that contains the services that the model exposes.
- **Web:** Package containing the REST controllers that expose the services.

5.2.1 Entities (Domain)

These are the persistent classes representing the data model entities defined in section 4.4. These classes define the attributes of the tables and for each of them, the getters and setters methods. These are access methods that allow recovering or assigning the value of the attributes. In each one of these classes the *@Entity* annotation is set to specify that they are persistent classes and that they must be mapped to the database.

To specify the primary key of an entity the *@Id* annotations are placed and to specify the strategy for generating values of the primary key the *@GeneratedValue* annotation is used.

The [Figure 5.2](#) and [Figure 5.3](#) show the entities of the project.

5.2.2 Repositories

The Data Access model as mentioned above is a software component that allows you to separate the access logic from the data and business logic. The DAOs provide the basic methods for creating, reading, updating and deleting (CRUD) information and other methods needed to perform other functions such as returning the number of times that gazettes have been published or edited in a given location, so that the business layer will use the repositories to access and interact with the data source. The *@Repository* notation is used to indicate that the class is a DAO.

In [Figure 5.4](#) you can see the project repositories.

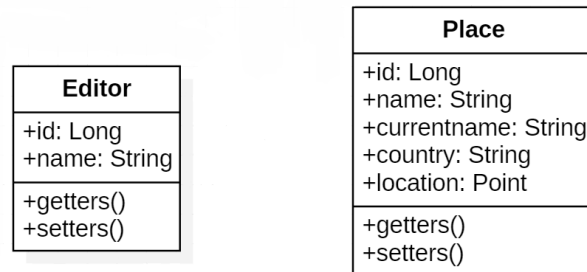


Figure 5.3: Place and editor entities

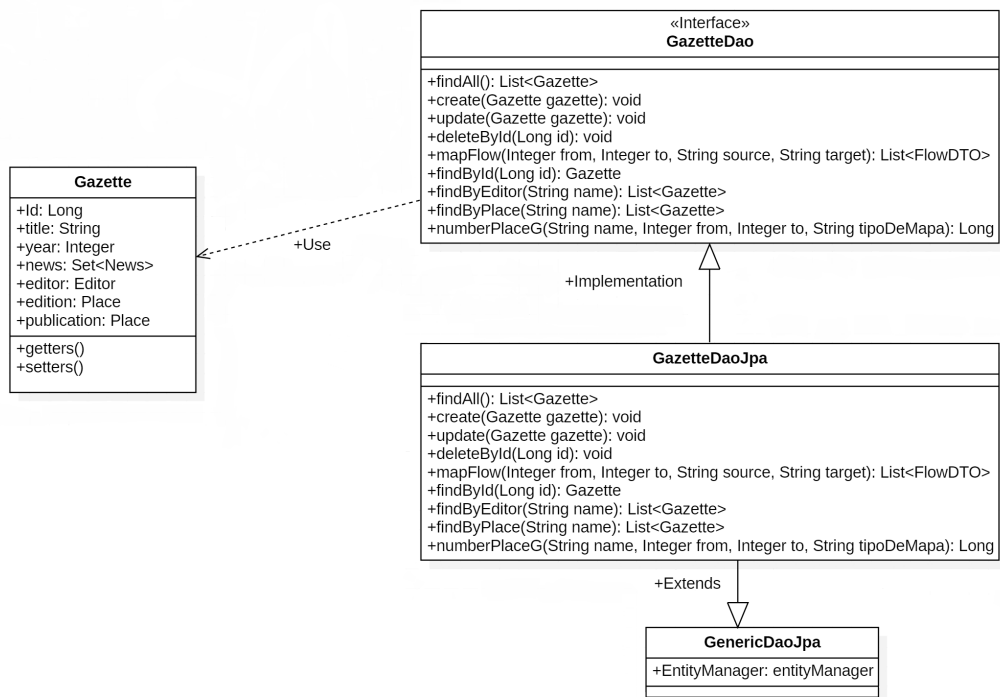


Figure 5.4: Gazette DAO

EditorService
+editorDAO: EditorDao +gazetteDAO: GazetteDao
+findAll(): List<EditorDTO> +create(editor: EditorDTO): EditorDTO +update(editor: EditorDTO): EditorDTO +findById(id: Long): EditorDTO +deleteById(id: Long): void

Figure 5.5: Editor service

PlaceService
+placeDAO: PlaceDao +gazetteDAO: GazetteDao +newsDAO: NewsDao
+findAll(): List<PlaceDTO> +listAll(from: Integer, to: Integer): List<PlaceDTO> +create(place: PlaceDTO): PlaceDTO +update(place: PlaceDTO): PlaceDTO +findById(id: Long): PlaceDTO +deleteById(id: Long): void

Figure 5.6: Place service

5.2.3 Services

The service layer implements the logic of the application, i.e. the functionalities necessary to fulfill the requirements of the application. In order to provide them, the service needs to access the stored information so it will be the operations of this layer that will call the DAOs that are the ones in charge of accessing the data. In our project, the *@Service* notation is used to indicate that the class is a service and that it is maintaining the business logic.

The following diagrams (Figure 5.5, Figure 5.6, Figure 5.7 and Figure 5.8) show the services of the project.

5.2.4 Controllers

It is the layer responsible for exposing the services through a REST API, it acts as an intermediate layer between the client and the server, processing the methods required by the incoming requests and transforming the data returned to DTOs (Data transfer objects) objects that encapsulate the information required by the request. Each method in the layer must make use

GazetteService
+gazetteDAO: GazetteDao +placeDAO: PlaceDao +editorDAO: EditorDao +newsDAO: NewsDao
+findAll(): List<GazetteDTO> +listAll(): List<FlowDTO> +create(gazette: GazetteDTO): GazetteDTO +findByld(id: Long): GazetteDTO +update(gazette: GazetteDTO): GazetteDTO +deleteByld(id: Long): void

Figure 5.7: Gazette service

NewsService
+gazetteDAO: GazetteDao +newsDAO: NewsDao +placeDAO: PlaceDao
+create(news: NewsDTO): NewsDTO +update(news: NewsDTO): NewsDTO +findByld(id: Long): NewsDTO +deleteByld(id: Long): void

Figure 5.8: News service

of the service exposed in the lower layer in order to carry out its functions and, to be called by the client, each one will be mapped with a URL.

The *@RestController* annotation is used to indicate that the class is a controller and for the mapping of URLs of the service with the methods. The *@GetMapping* (HTTP GET), *@PostMapping* (HTTP POST), *@PutMapping* (HTTP PUT) and *@DeleteMapping* (HTTP DELETE) annotations are used next to the URL to be called. If you want to group requests that refer to the same URL, you can use the *@RequestMapping* annotation along with the piece of URL they share and have in common, and then in the method annotations (get, post, put, delete) you indicate the part of the URL that is missing if needed. An example would be: The controller class *@RequestMapping("/api/editors")* and a *@GetMapping("/id")* method.

The following list shows the operations and URLs used in the project:

1. AccountResource

- (a) **POST** */auth/authenticate*: allows a user to log in with the credentials that are sent in the body of the request

2. EditorResource

- (a) **GET** */api/editors/*: returns the information of all the editors.
- (b) **GET** */api/editors/id*: returns only one editor's information.
- (c) **POST** */api/editors/*: creates a new editor.
- (d) **PUT** */api/editors/id*: modifies the data of an editor.
- (e) **DELETE** */api/editors/id*: removes an editor.

3. GazetteResource

- (a) **GET** */api/gazettes/*: returns the information of all the gazettes.
- (b) **GET** */api/gazettes/map*: returns the necessary information about the relationship between the gazettes and its news items locations to display it on the map.
- (c) **GET** */api/gazettes/id*: returns only the information of a single gazette.
- (d) **POST** */api/gazettes/*: creates a new gazette.
- (e) **POST** */api/gazettes/id/news*: creates a new news item for a gazette.
- (f) **PUT** */api/gazettes/id*: modifies the data of a gazette.
- (g) **DELETE** */api/gazettes/id*: removes a gazette.

4. PlaceResource

- (a) **GET** */api/places/*: returns the information of all the places.

- (b) **GET** `/api/places/map`: returns the necessary information for the map of all the places.
- (c) **GET** `/api/places/id`: returns only one place's information.
- (d) **POST** `/api/places/`: creates a new place.
- (e) **PUT** `/api/places/id`: modifies the data of a place.
- (f) **DELETE** `/api/places/id`: removes a place.

5. NewsResource

- (a) **GET** `/api/news/id`: returns the information of a single news item.
- (b) **PUT** `/api/news/id`: modifies the data of a news item.
- (c) **DELETE** `/api/news/id`: removes a news item.

5.3 Front-end

This section explain the internal structure of the front-end. For the development of this part it was decided to implement it with Vue.js a progressive framework that is characterized by working with the MVVM (Model-View-ViewModel) pattern that simplifies the development of event-driven interfaces and tries to decouple the user interface from the application logic as much as possible.

5.3.1 Components

The client side is composed of components which are reusable and configurable Vue.js instances. In other words, they are customized HTML elements that encapsulate code so that it can be reused within other components, by creating a new instance of it and using a hierarchical structure, this means that parents can modify their children but not the other way around. For a component to be reusable it must be registered globally or locally with a tag. At a high level you could say that components are custom elements to which the Vue.js compiler would associate a specific behavior.

A Vue.js component consists of the following parts (see [Figure 5.9](#)):

- **Template:** part where the HTML code is used to structure the web page and define the view.
- **Script:** This part uses the JavaScript language that allows the web page to be dynamic by defining the behaviour of the component.

```

</template>
<script>
import RepositoryFactory from "@/repositories/RepositoryFactory";
import Swal from "sweetalert2";
const EditorsRepository = RepositoryFactory.get("editors");

export default {
  data: () => ({
    editors: [],
    search: '',
    editedIndex: -1,
    headers: [
      { text: "Name", value: "name" },
      { text: "Actions", value: "action", sortable: false }
    ]
  }),
  async created() {
    this.editors = await EditorsRepository.getAll();
  },
  methods: { ...
}];
};
</script>
<style scoped>
.clear {

```

Figure 5.9: Structure of a Vue.js component

- **Style:** part where the language of CSS styles is used to modify the presentation of the website, allows us to organize the elements that are defined in the template, change the background colours, fonts...

The following [Figure 5.10](#) shows how the components are divided depending on the permissions, i.e. the type of user who can access them. On the one hand, we have the registered users, who are the system administrators since they are in charge of handling the data, and on the other hand, the unregistered users who can visualize and interact with the map.

The “NotFound.vue” and “App.vue” components are not shown in this diagram because they do not implement important functionalities.

5.3.2 Routing

In order to access the components, a file was defined with VueRouter which relates a path for each component. In this case, the components are grouped by entities (editor, place and gazette) and have their own file router.js where the paths for these components are defined and then, we have the file Router.js that keeps the paths of the components both those that can be accessed by all users and those defined in the files mentioned above by importing their paths.

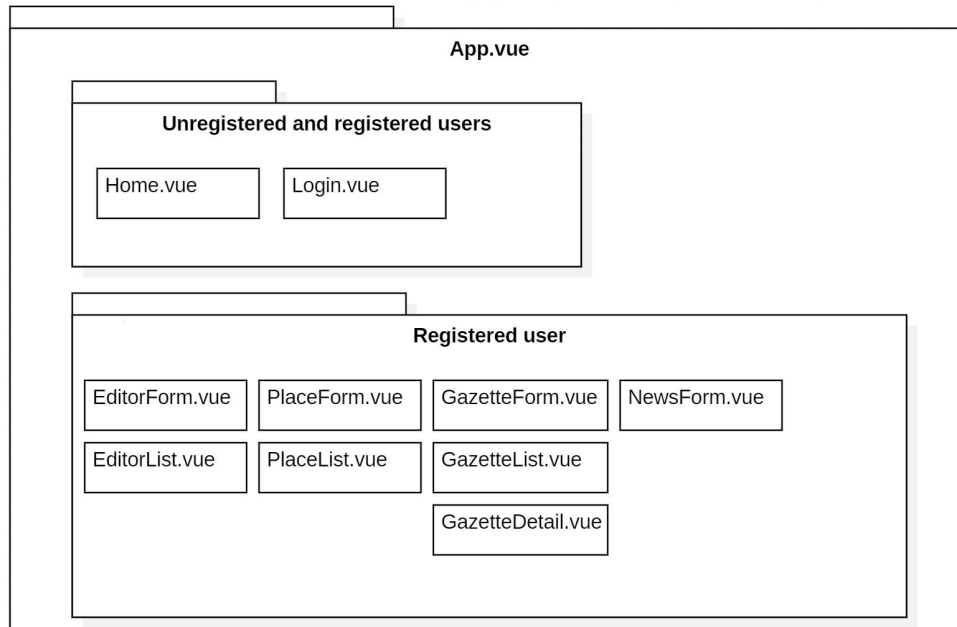


Figure 5.10: Component diagram at the front-end

Name	Component	Route
Home	Home.vue:	/
Login	Login.vue:	/login
EditorForm	EditorForm.vue	/editor/new
EditorEdit	EditorForm.vue	/editor/:id/edit
EditorList	EditorList.vue:	/editors
PlaceForm	PlaceForm.vue	/place/new
PlaceEdit	PlaceForm.vue	/place/:id/edit
PlaceList	PlaceList.vue	/places
GazetteForm	GazetteForm.vue	/gazette/new
GazetteEdit	GazetteForm.vue	/gacette/:id/edit
GazetteList	GazetteList.vue	/gazettes
GazetteDetail	GazetteDetail.vue	/gazette/:id
NewsForm	NewsForm.vue	/gazette/:id/news
NewsEdit	NewsForm.vue	/gazette/:id/news/:idNews/edit

Table 5.1: Component path table

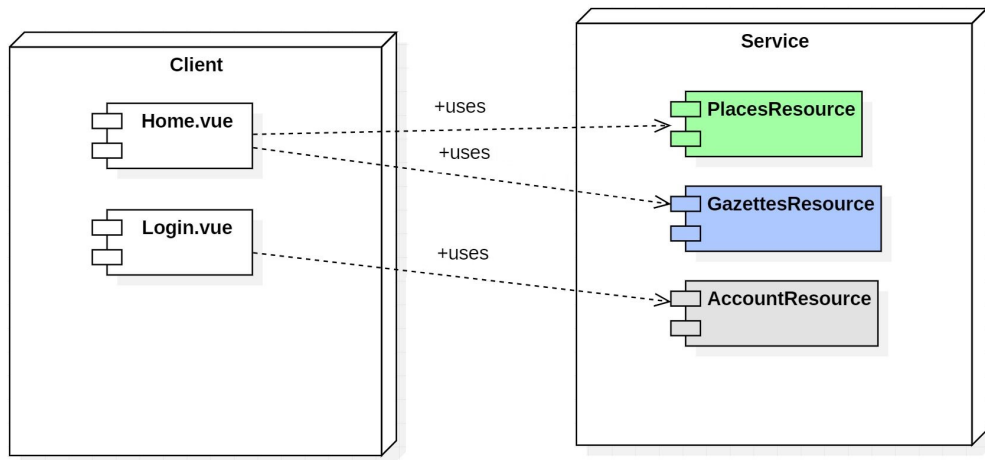


Figure 5.11: Communication of the common components from the Client to the Server

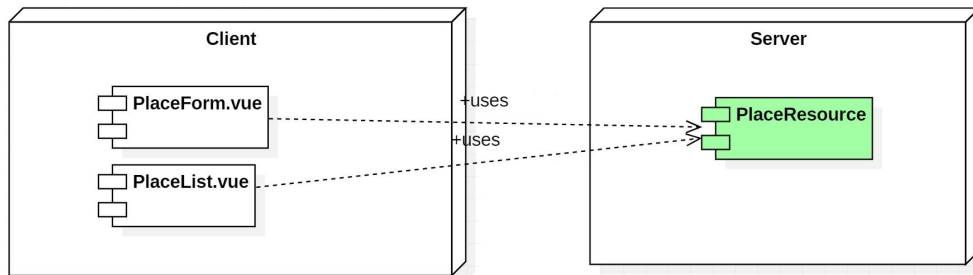


Figure 5.12: Communication of the user components from the Client to the Server (1)

5.3.3 Communication with the server

In this section, we show the communications that take place between the components on the client side and the server controllers through HTTP requests.

The [Figure 5.11](#) shows the communication of the components that can be accessed by all users, both registered and unregistered. The requests that are made are for authentication and those to retrieve information about the places shown on the map.

The [figures 5.12](#), [5.13](#), [5.14](#) and [5.15](#) show the communication of those components that only registered users can access, so the requests that are made are to retrieve, create, edit or delete data from any of the gazettes, editors or places.

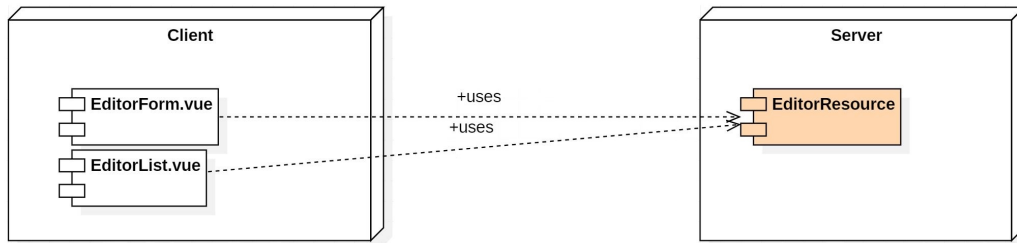


Figure 5.13: Communication of the user components from the Client to the Server (2)

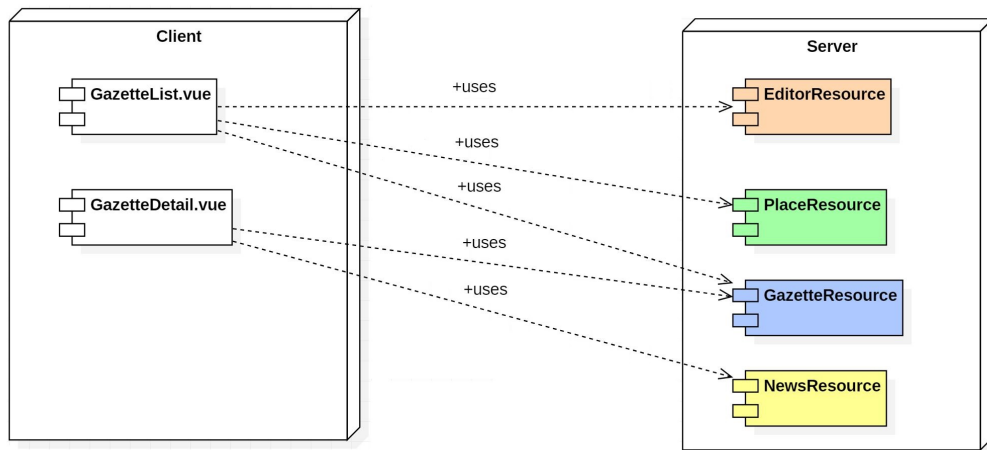


Figure 5.14: Communication of the user components from the Client to the Server(3)

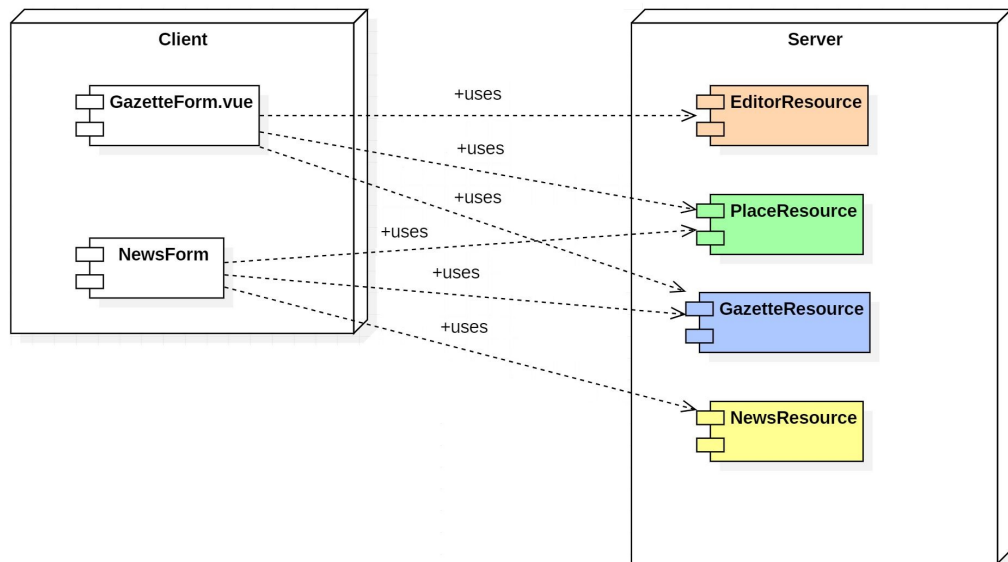


Figure 5.15: Communication of the user components from the Client to the Server(4)

Implementation and tests

In this chapter, we describe some complex algorithm that implements an important functionality of the application and we also explain the tests made to the project.

6.1 Implementation

6.1.1 Get the number of times a place is referenced

One of the requirements of this project is to be able to see the number of times an event has been published, edited, noticed, or occurred in a place. The following paragraphs will explain how this algorithm works.

From the component "Home" of the client side a GET request will be sent to the PlaceResource controller with the method *getList(from, to)* defined in PlacesRepository (see [Figure 6.1](#)), this method is used to obtain the necessary information of the places to be able to show them in the map, among this information are the data that we need. As the map can be filtered by years, the request can be made with up to two parameters.

- Request without parameters: the information of all the places is retrieved.
- Request with parameter *from*: the information of only that year is retrieved.
- Request with *from* and *to* parameters: the information of the places whose activities are in that range of years is recovered.

Once the answer is received, the *drawPlace()* method is called, which will draw on the map the circles over the places that meet the requested conditions depending on the filters selected by the user, that is, if he has decided to show the publications, editions, events, news or a combination of them.

From the server side, once the GET request has reached the PlaceResource controller, the PlaceService *listAll(Integer from, Integer to)* function is called (see [Figure 6.2](#)). This function

```

async getList(from, to) {
  if (from == undefined && to == undefined){
    return (await HTTP.get(`${resource}/map`)).data;
  }else if (to == undefined){
    return (await HTTP.get(`${resource}/map?from=${from}`)).data;
  } else {
    return (await HTTP.get(`${resource}/map?from=${from}&to=${to}`)).data;
  }
},

```

Figure 6.1: getList() function in PlacesRepository

```

public List<PlaceDTO> listAll(Integer from, Integer to) {
  return placeDAO.findAll().stream()
    .map(place -> new PlaceDTO(place,
      gazetteDAO.numberPlaceG(place.getName(), from, to, "publication"),
      gazetteDAO.numberPlaceG(place.getName(), from, to, "edition"),
      newsDAO.numberPlaceN(place.getName(), from, to, "notice"),
      newsDAO.numberPlaceN(place.getName(), from, to, "event")))
    .collect(Collectors.toList());
}

```

Figure 6.2: ListAll() function in PlaceService

retrieves a list of all places and adds the information about the number of times each action has occurred. For each place on the list obtained, four database queries are made, one for each action that could have occurred at that place. Two functions have been implemented for this purpose, one in GazetteDao and another in NewsDao, which are the ones that will perform the query to the database in order to obtain that information.

In both repositories a function was implemented, one is called *numberplaceG* for GazetteDao (Figure 6.3) and another one called *numberplaceN* for NewsDao. These functions are the ones that launch the query to the database and, get as answer for each action the number of times that place is referenced in the gazettes. To launch the query, it is necessary to formulate it first, for this reason, we use the function *getSQLQuery* (see Figure 6.4) that prepares the query depending on the parameters received. Among these parameters you can find the action (publication, edition, notice, or event), for which you want to know its quantity in the places, and the range of years. The action is introduced as a parameter because as mentioned before, four queries are made for each place, exactly one for each action, which are: in *numberplaceG* two queries are made, one to know the number of publications and a second one to know the number of editions, and the same thing happens in *numberplaceN* but with the events and notices.

Later this data are added in the DTO of each place. This DTO is returned by the server function so that it can be sent later as an answer to the client's request.

```
@Override
public Long numberPlaceG(String name, Integer from, Integer to, String tipoDeMapa) {
    String query;
    if (from == null && to == null) {
        query = getSQLQuery(name, null, null, tipoDeMapa);
        return (Long) entityManager.createQuery(query).setParameter("name", name)
            .getSingleResult();
    } else if (from != null && to == null) {
        query = getSQLQuery(name, from, null, tipoDeMapa);
        return (Long) entityManager.createQuery(query).setParameter("name", name)
            .setParameter("year1", from).getSingleResult();
    } else {
        query = getSQLQuery(name, from, to, tipoDeMapa);
        return (Long) entityManager.createQuery(query).setParameter("name", name)
            .setParameter("year1", from).setParameter("year2", to).getSingleResult();
    }
}
```

Figure 6.3: NumberplaceG function in GazetteDao

```
private String getSQLQuery(String name, Integer from, Integer to, String tipoDeMapa) {
    String query = "Select count(*) from Gazette u where ";
    if (tipoDeMapa == "publication") {
        query = query + "u.publication.name = :name ";
    } else if (tipoDeMapa == "edition") {
        query = query + "u.edition.name = :name ";
    }
    if (from != null && to == null) {
        query = query + "and u.year = :year1 ";
    } else if (from != null && to != null) {
        query = query + "and u.year between :year1 and :year2 ";
    }
    return query;
}
```

Figure 6.4: getSQLQuery function in GazetteDao

6.1.2 Visualization of the places

In order to display on the web interface, the circles over those places that satisfy the conditions entered by the client, the following methods have been implemented in the front-end.

The client must first select the map types from the filters. Continuing with what was explained in the previous case of obtaining the number of times that an action had occurred in a place, when the client receives the response to the request, the method that implements the *drawplace* function is called. In this method, named *mapSource()* (Figure 6.5 what it does is:

- First, it obtains the maximum value of the number of publications, editions... depending on what the client has selected to show on the map. Since in reference to this number, the diameter of the circles will be calculated, that is, the one with the highest value will be the largest because it is the one with the highest activity and the diameter of the rest will be calculated in relation to the difference between its value and the maximum.
- Next, the *drawPlace* function will be called to create the circles whose characteristics will be obtained according to the properties of the map and the place (see Figure 6.6).

The main properties are:

- Colour of the line of the circle: it depends on the type of map that has been decided to show.
- Fill colour: depends on the type of map that has been decided to display.
- Radius: parameter that will be calculated by multiplying the base value they can have as a radius by the number of times the action has occurred in that place and dividing that result by the maximum value, mentioned above. The number mentioned above will depend on the number of map types the user has decided to display. If it is a map where two types of actions are shown, the value of both is added.

6.1.3 Flow between locations

To represent the communication between the places, the same thing is done as in the previous step, but apart from calling *mapSource()*, *mapTarget()* is called which is the one that shows the circles in the places defined as target.

To know the connections between source and target places, it is necessary to make a GET request to the GazetteResource controller with the *getList(from, to, source, target)* method defined in GacettesRepository. As parameters are sent:

```

async mapSource() {

function drawPlace(place, tipoDeMapa, maximun) {
  const auxCircle = L.circle(
    place.location.coordinates,
    _getPlaceProperties(place, tipoDeMapa, maximun)
  ).addTo(mymap);
  auxCircle.bindPopup(_getPlacePopup(place));
  return auxCircle;
}

if (this.filterP != undefined) {
  var maxValue = this.getMax(this.filterP);
  this.filterPlaces.forEach(place => {
    const circleAux = drawPlace(place, this.filterP, maxValue);
    this.layerCircle.push(circleAux);
    if(this.filterT != undefined){
      circleAux.on('mouseover', (e) => {
        this.mapFlow(e);
      })
      circleAux.on('mouseout', ()=>{
        this.clearFlow();
      })
    }
  })
});
}
},

```

Figure 6.5: MapSource() function in Home component

```

function _getPlaceProperties(place, tipoDeMapa, maximun) {
  return {
    color: colorDeTipoDemapa[_getTipoMapaUnico(tipoDeMapa)],
    fillColor: rellenoDeMapa[_getTipoMapaUnico(tipoDeMapa)],
    fillOpacity: 0.5,
    radius: _getRadius(_auxRadius(place, tipoDeMapa), maximun),
    _feature: place
  };
}

```

Figure 6.6: Getproperties function in Home component


```

async mapFlow(e){
  var prueba = e.target.options._feature.id;
  var max = this.getMaxWeight();
  this.placesFlow.forEach(e =>{
    var source = this.filterPlaces.find(place => place.id == e[0]);
    var target = this.filterTarget.find(place => place.id == e[1]);
    if (prueba== source.id || prueba ==target.id){
      var number = e[2];
      var multipolyline = L.polyline([source.location.coordinates,
        target.location.coordinates], _multiPolyLineOptions(number, max));
      multipolyline.addTo(mymap);
      this.layerLinesFlow.push(multipolyline);
    }
  })
}

```

Figure 6.7: MapFlow() function in Home component

- The range of years for which you want to obtain the answer (parameters *from* and *to*).
- One of the actions (publication, edition, notice or event) taken from the target filter.
- One of the actions from the source filter.

This is done because in this way, for example, you can see the communication between the places that published gazettes and the places where the news events took place.

With this request, a list will be obtained with the locations where the first action took place (source), the locations where the second action (target) occurred and the number of times those two locations are connected.

Once the answer is obtained with the data, in the function *mapSource()* and *mapTarget()* for each of the places incorporated in the map, the function *mapFlow()* (Figure 6.7) is called. This one is in charge of adding the lines between the places to the map, thus representing the connections. These lines will only be seen if the mouse is positioned over the circles generated by the functions mentioned above. To create the lines, this function first obtains the characteristics of the place to which you want to know its relations and then calculates the maximum value of all the connections. This value is taken from the list that we obtain as a response to the *getList()* request explained in the previous paragraph. The thickness of the lines will vary depending on this value, as it is done with the diameters of the circles explained above. Later, it adds to the map all the relations that the place has, allowing to see them later in the interface.

6.2 Testing

In this section we comment on the type of tests that have been carried out to ensure quality and compliance with customer requirements.

The tests that have been carried out in this project are known as functional tests, which are specific, concrete and exhaustive tests to prove and validate that the software meets the requirements that have been specified throughout the project. There are several types of functional tests, but those that have been performed in this project are known as acceptance software quality tests.

These tests allow to know if an application complies with the expected performance and they focus on the users who are in charge of validating the functionality and performance of the application.

These evaluations have been performed throughout the entire development process, first by the development team, checking the correct functioning of the new functionalities, added to the project during the implementation of each of the iterations, and then, in each of the sprint review meetings, by the entire team checking if the new user stories achieved the expected objectives and if the interface was adequate, intuitive and easy to use.

The requested changes as a result of these sprint review were:

- **Sprint 1** [17/03/2020]: it was decided not to change anything.
- **Sprint 2** [31/03/2020]: it was decided not to change anything.
- **Sprint 3** [14/04/2020]: The list of places initially showed the coordinates, so it was asked to be changed so that they would not be shown in the table, as it was not interesting information. The show, delete and edit buttons of all the tables were separated in different columns and it was asked to change them so that there was only one column of actions containing all the buttons.
- **Sprint 4** [28/04/2020]: At the beginning to modify editors and places, a dialog had been designed so that when the edit button was pressed it would be shown and you could modify the data there without changing the component. But when that same function was implemented for the gazettes, as that functionality could be accessed from two components, the table view and the detail view, it was decided to reuse the “GazetteForm” component for editing. And as it was unusual to have two different ways to modify the application information, it was also asked for change the dialog of places and editors to reuse the form component of each one. It was also decided in this meeting to implement confirmation windows for when you want to delete some information, so you do not allow the user to delete easily things he did not really want. The changes suggested in the previous sprint were implemented and tested.
- **Sprint 5** [12/05/2020]: The map had been implemented at the beginning without the possibility of seeing all the types together, so it had to be modified to be able to show this option and to allow combining different actions at the same time. It was decided

to change also the style of the confirmation windows to make them more visually appealing. The changes suggested in the previous sprint were implemented and tested.

- **Sprint 6** [26/05/2020]: The changes suggested in the previous sprint were implemented and tested and in this review was decided not to change anything.
- **Sprint 7** [09/06/2020]: The layout of the filters in the "Home" component was ask to be changed, to allow the map to have more space and avoid the filters to take too much space.

Solution developed

The aim of this chapter is to show the developed application resulting from this project. To this end, an explanation will be provided on the main functionalities of the application.

7.1 Use of the map

To access this view and interact with the map it is not necessary to be authenticated in the application, that is, all types of users can view the information displayed on the map.

7.1.1 Displaying the information

The source filter is a multiple selection filter, this means that you can select all the options that you want from the filter. The filter offers to show the places where a gazette has been published or edited or an event has been notified or occurred. As mentioned above, it allows you to select several options, thus allowing you to display a combination of several activities. In order for the map to show information about the types of activities that have occurred there, it is necessary to choose at least one option in the filter. The [Figure 7.1](#) and [Figure 7.2](#) show how the information is displayed.

7.1.2 Display the number of references.

To see the number of references of the activities that have taken place in a location it is necessary to click on one of the circles shown on the map so that a pop up with the desired information is displayed. The [\(Figure 7.3\)](#) shows a pop-up displaying the number of references.

7.1.3 Visualize the communications

One of the main requirements of this project is to be able to see the communications between the places, for this the source filter should have chosen only one of the possible options,

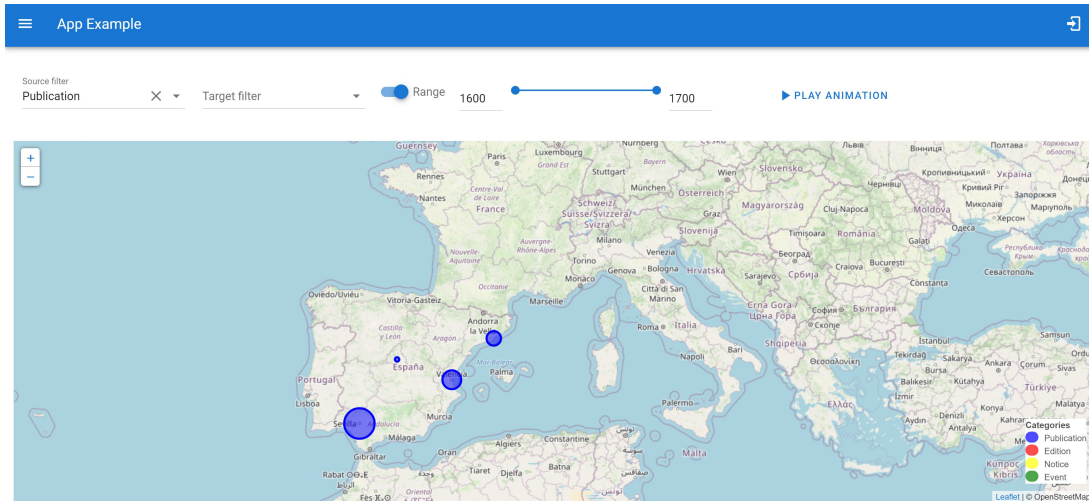


Figure 7.1: Map page

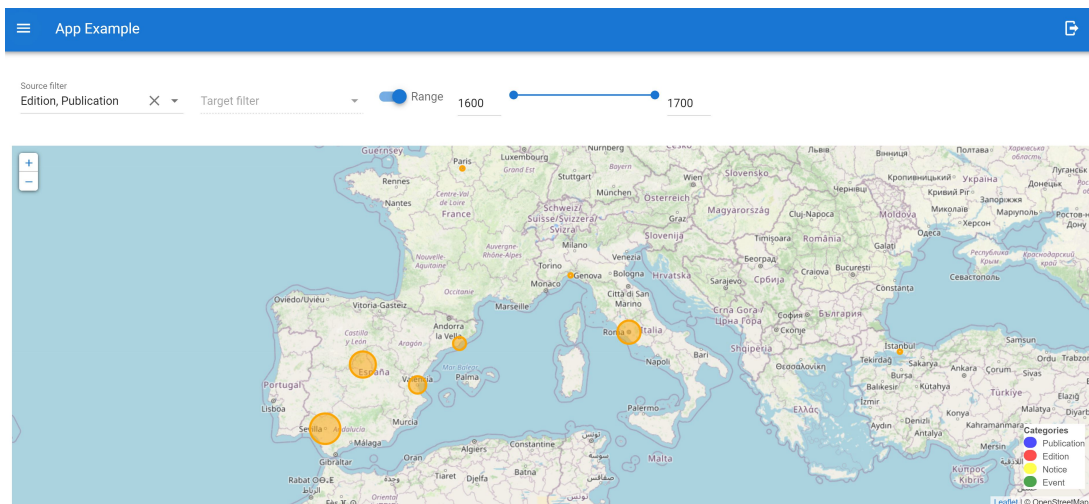


Figure 7.2: Map page combination of actions

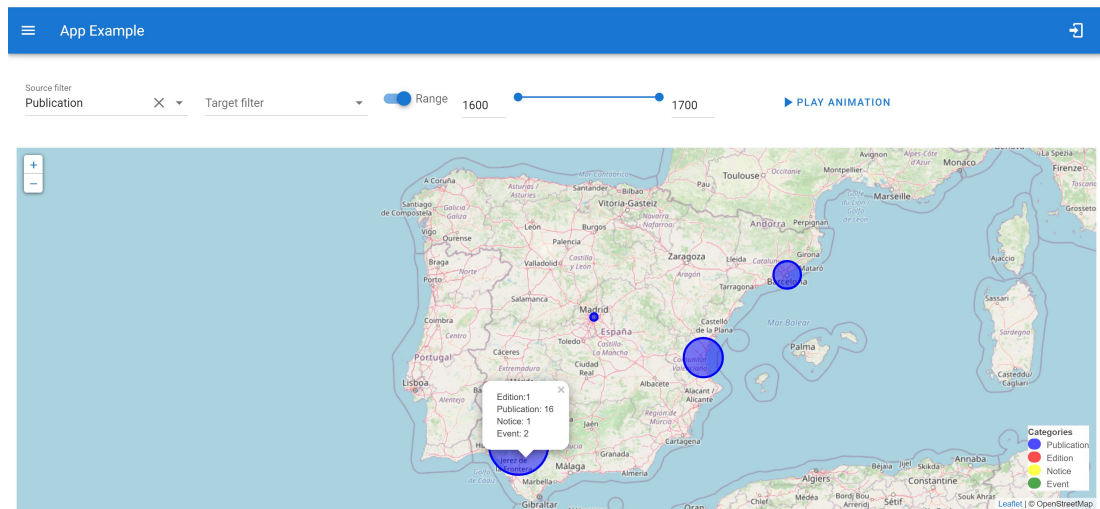


Figure 7.3: Map pop-up

only in this way the target filter will be enabled. So, we choose on option of source filter and another one of target filter and we should position the mouse over the place we want to know the connections. The [Figure 7.4](#) and [Figure 7.5](#) show an example of the connections that could be seen on the map.

7.1.4 Filter by a year or a range of years

This map also allows you to filter by years:

- By range of years: you have to modify the slider or overwrite some of the values on the sides.
- By a certain year (see [Figure 7.6](#)): you must move the slider in the year you want or overwrite the value in the side.

7.1.5 View an animation

Another feature that has been implemented is the possibility to see an animation of the evolution of the selected activities over a range of years. This option can be seen only if the target filter has only one option selected and if the range of years is activated so that the play button appears which will start the animation. You can also see the animation with the source filter selected to see the modification of the two selected actions. In the ([Figure 7.7](#)) we can see an example of how the screen looks when showing an animation..

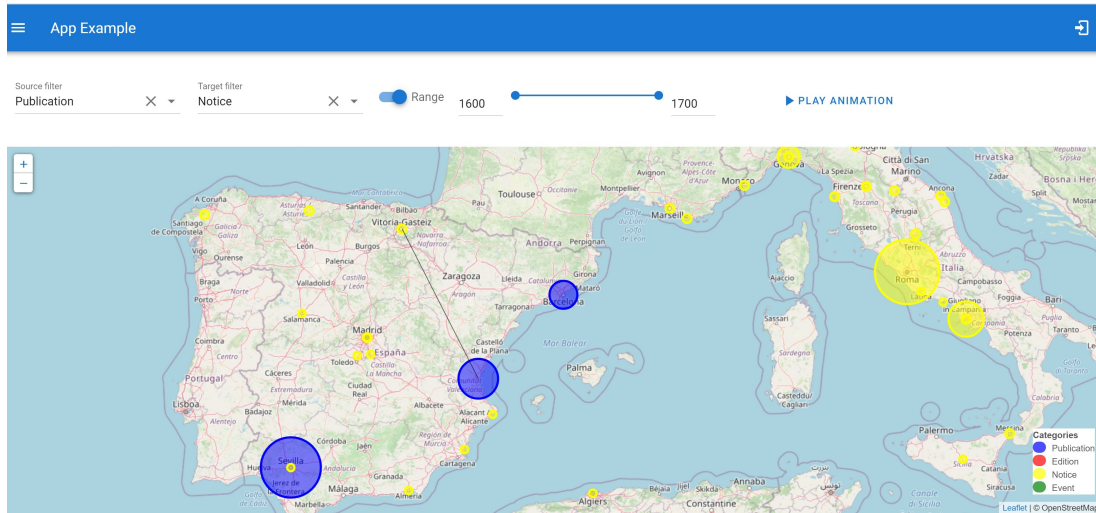


Figure 7.4: Map flow

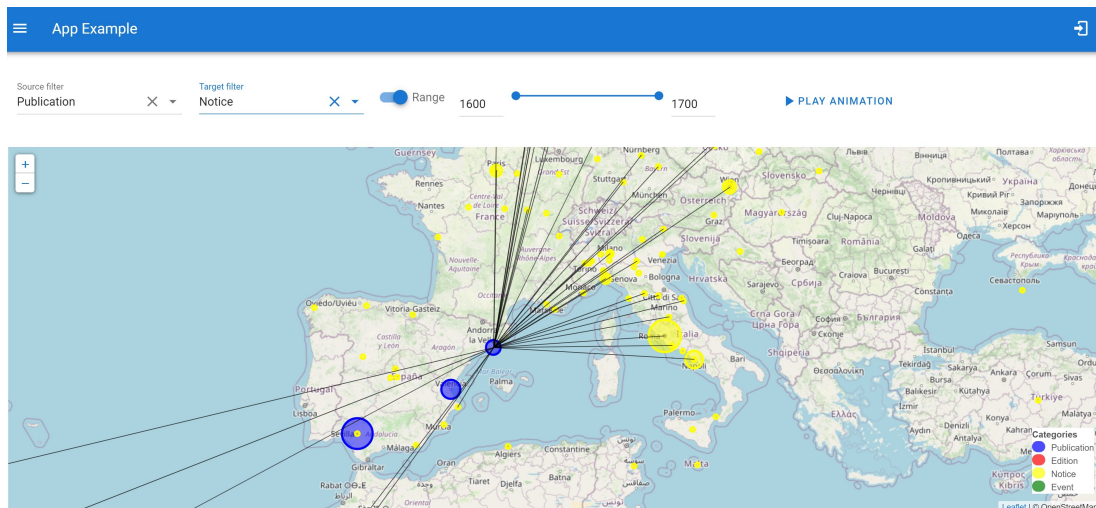


Figure 7.5: Map flow (2)

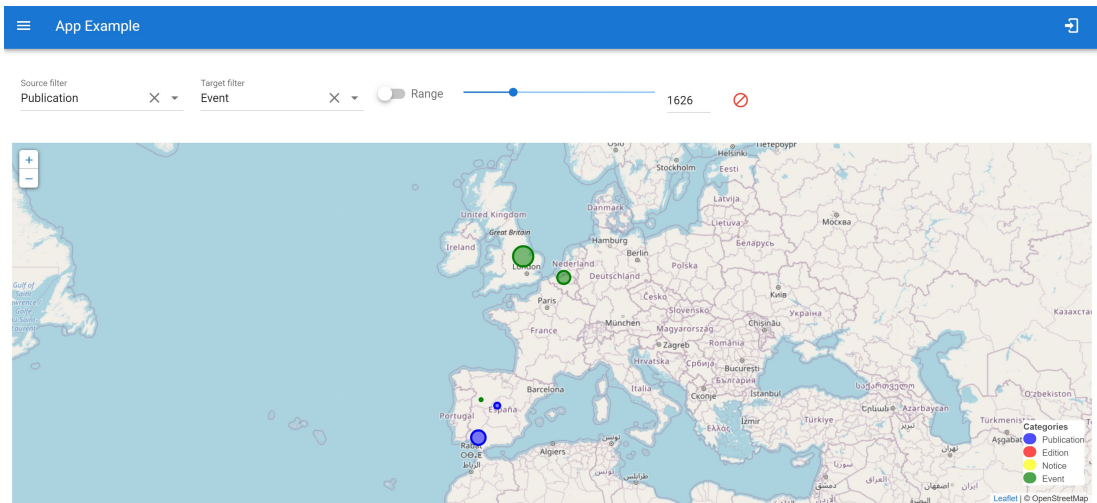


Figure 7.6: Map filtered by a year

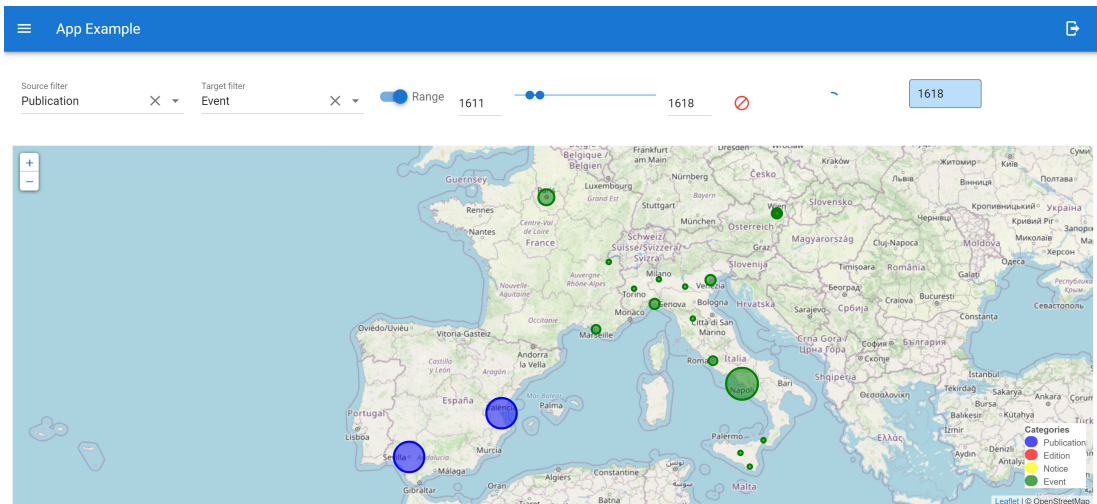


Figure 7.7: Map animation

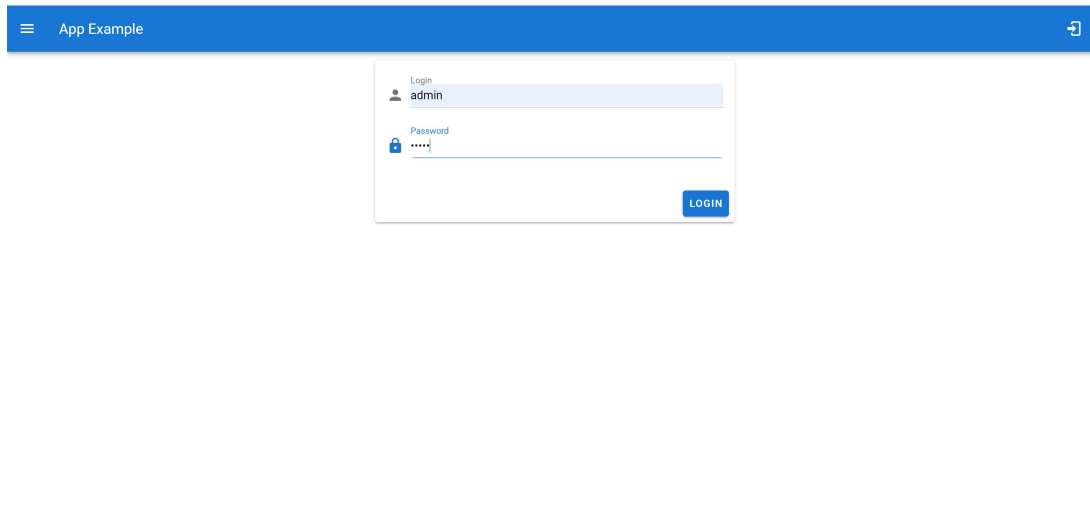


Figure 7.8: Login page

7.2 Authentication

To manage the data, i.e. add, modify or delete it, it is necessary to be authenticated since only logged-in users have access to this information.

To do this you must click on the login button and enter your credentials and password (see [Figure 7.8](#)).

7.3 Data Management

Once authenticated we will have access to the screens that show the lists with the information of the editors as shown in [Figure 7.9](#), of the places as shown in [Figure 7.10](#), and of the gazettes as shown in [Figure 7.11](#).

On these pages (editors, places and gazettes) new information can be added by clicking on the “New” button. Once it is pressed, you will be moved to the form page where you can fill in the information of the new entity that you need to add. An example is shown in ([Figure 7.12](#))

It is also possible to change the information of any editor, place or gazette, for this you must press the modify button. Next, the entity’s form will be shown with the information in editable mode as it is shown in [Figure 7.13](#)

In order to see the news of each gazette it is necessary to enter in its view in detail (see [Figure 7.14](#) and [Figure 7.15](#)). In this page you can delete the news, edit them or add new ones. A form will be shown, as the one commented for the previous entities, for the actions of adding or editing.

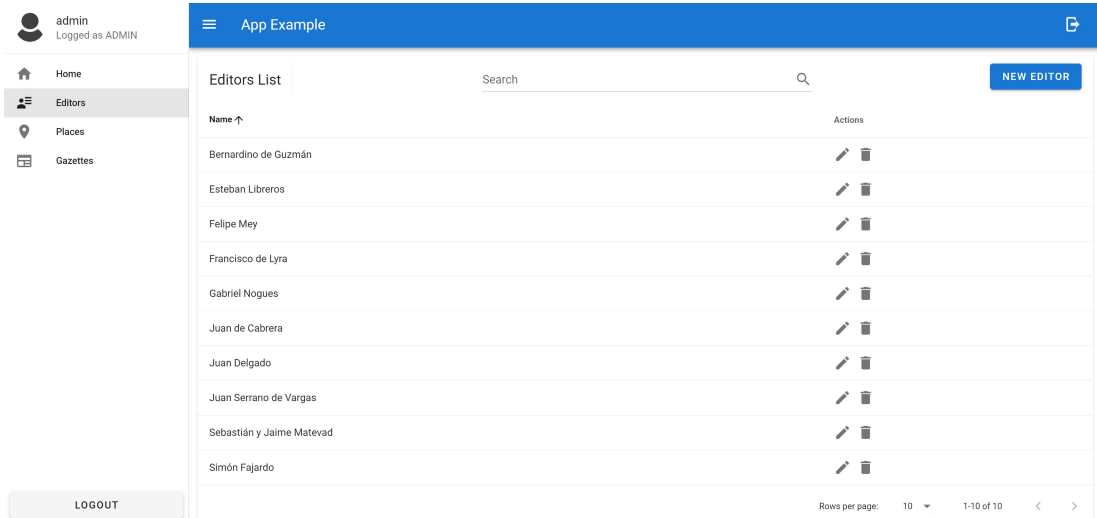


Figure 7.9: Editors list page

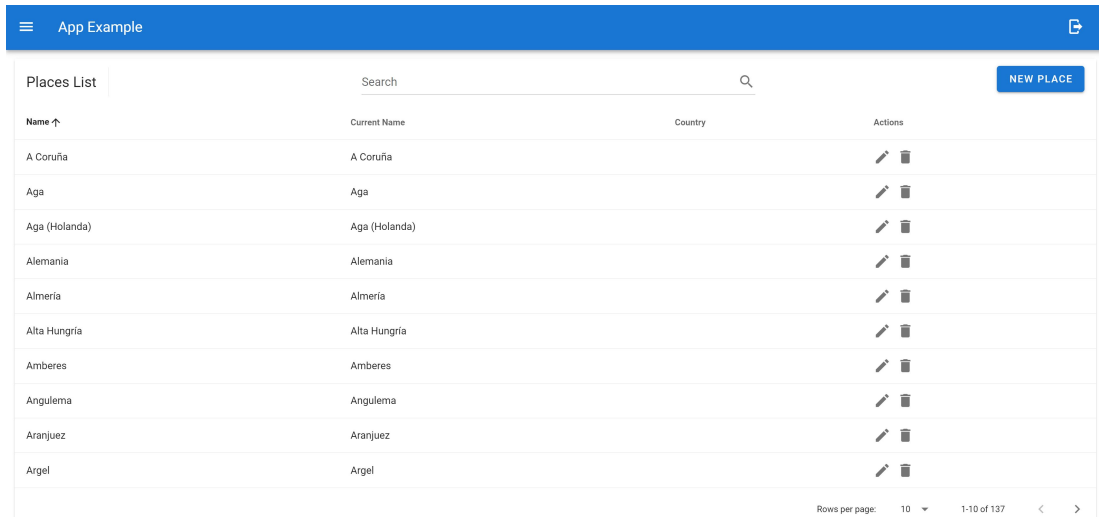
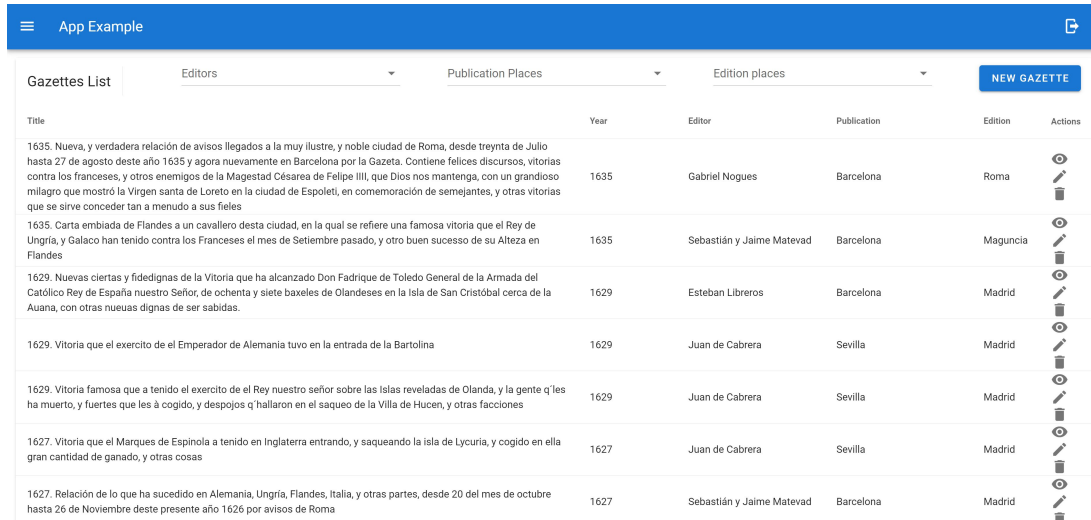


Figure 7.10: Places list page
























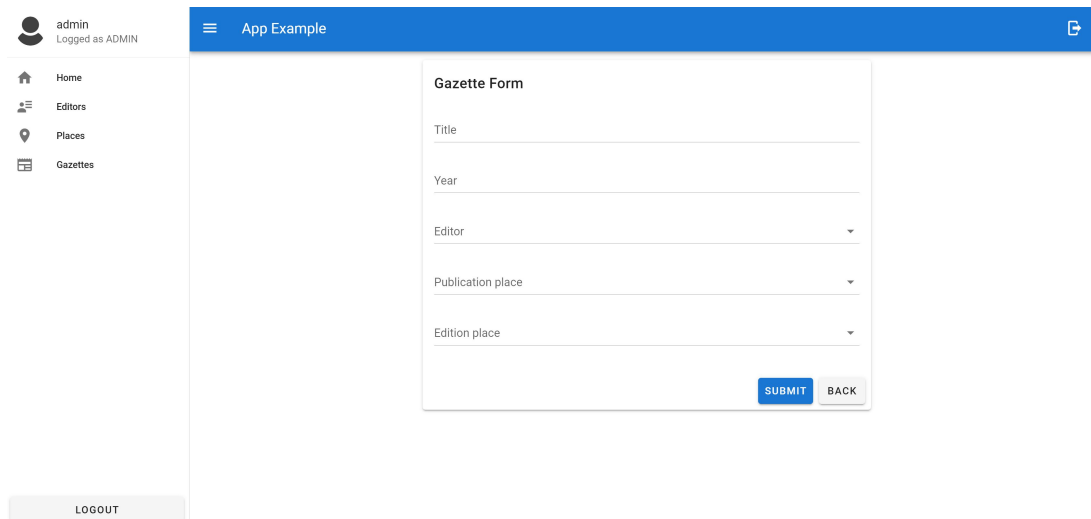
Title	Year	Editor	Publication	Edition	Actions
1635. Nueva, y verdadera relación de avisos llegados a la muy ilustre, y noble ciudad de Roma, desde treynta de Julio hasta 27 de agosto deste año 1635 y agora nuevamente en Barcelona por la Gazeta. Contiene felices discursos, victorias contra los franceses, y otros enemigos de la Magestad Césarea de Felipe III, que Dios nos mantenga, con un grandioso milagro que mostró la Virgen santa de Loreto en la ciudad de Espoleti, en conmemoración de semejantes, y otras victorias que se sirve conceder tan a menudo a sus fieles	1635	Gabriel Nogues	Barcelona	Roma	  
1635. Carta embiada de Flandes a un cavallero desta ciudad, en la qual se refiere una famosa vitoria que el Rey de Ungría, y Galaco han tenido contra los Franceses el mes de Setiembre pasado, y otro buen successo de su Alteza en Flandes	1635	Sebastián y Jaime Matevad	Barcelona	Maguncia	  
1629. Nuevas ciertas y fidedignas de la Vitoria que ha alcanzado Don Fadrique de Toledo General de la Armada del Católico Rey de España nuestro Señor, de ochenta y siete baxeles de Olandeses en la Isla de San Cristóbal cerca de la Auna, con otras nuevas dignas de ser sabidas.	1629	Esteban Libreros	Barcelona	Madrid	  
1629. Vitoria que el exercito de el Emperador de Alemania tuvo en la entrada de la Bartolina	1629	Juan de Cabrera	Sevilla	Madrid	  
1629. Vitoria famosa que a tenido el exercito de el Rey nuestro señor sobre las islas reveladas de Olanda, y la gente q'les ha muerto, y fuertes que les à cogido, y despojos q' hallaron en el saqueo de la Villa de Hucen, y otras facciones	1629	Juan de Cabrera	Sevilla	Madrid	  
1627. Vitoria que el Marques de Espinola a tenido en Inglaterra entrando, y saqueando la isla de Lycuria, y cogido en ella gran cantidad de ganado, y otras cosas	1627	Juan de Cabrera	Sevilla	Madrid	  
1627. Relación de lo que ha sucedido en Alemania, Ungría, Flandes, Italia, y otras partes, desde 20 del mes de octubre hasta 26 de Noviembre deste presente año 1626 por avisos de Roma	1627	Sebastián y Jaime Matevad	Barcelona	Madrid	  

Figure 7.11: Gazettes list page



admin
Logged as ADMIN

Home

Editors

Places

Gazettes

LOGOUT

App Example

Gazette Form

Title

Year

Editor

Publication place

Edition place

Figure 7.12: Gazette form page

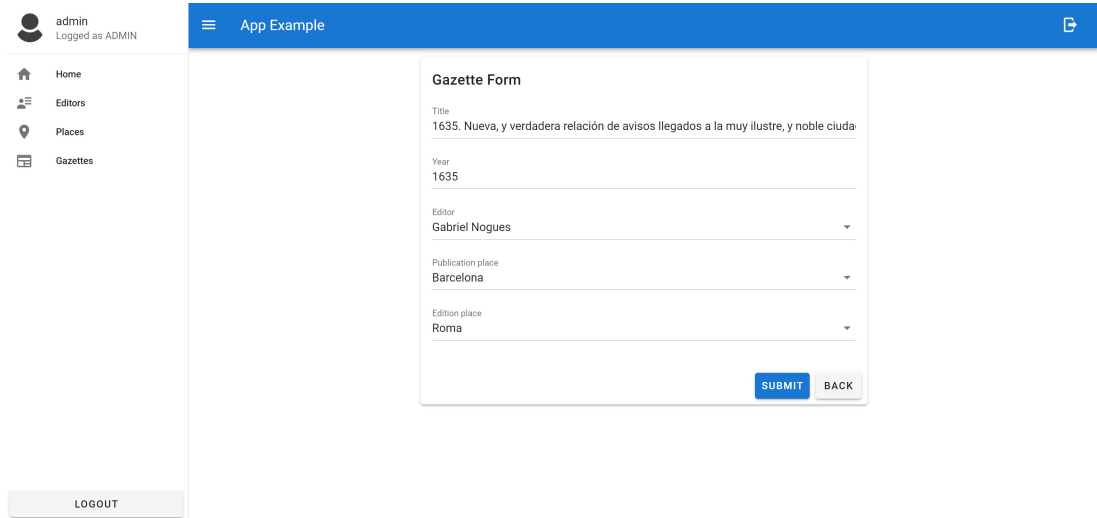


Figure 7.13: Gazette modification page

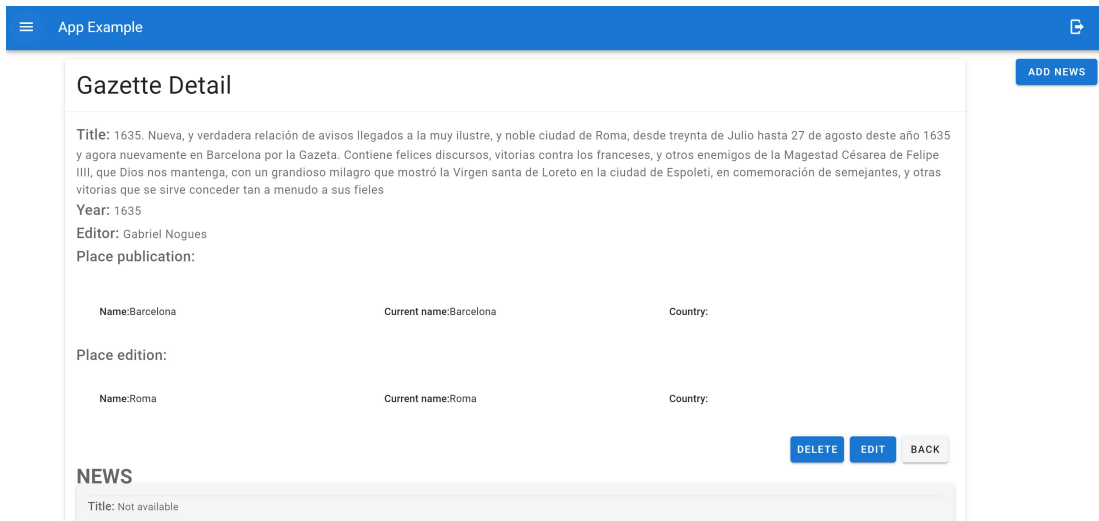


Figure 7.14: Gazette detail page

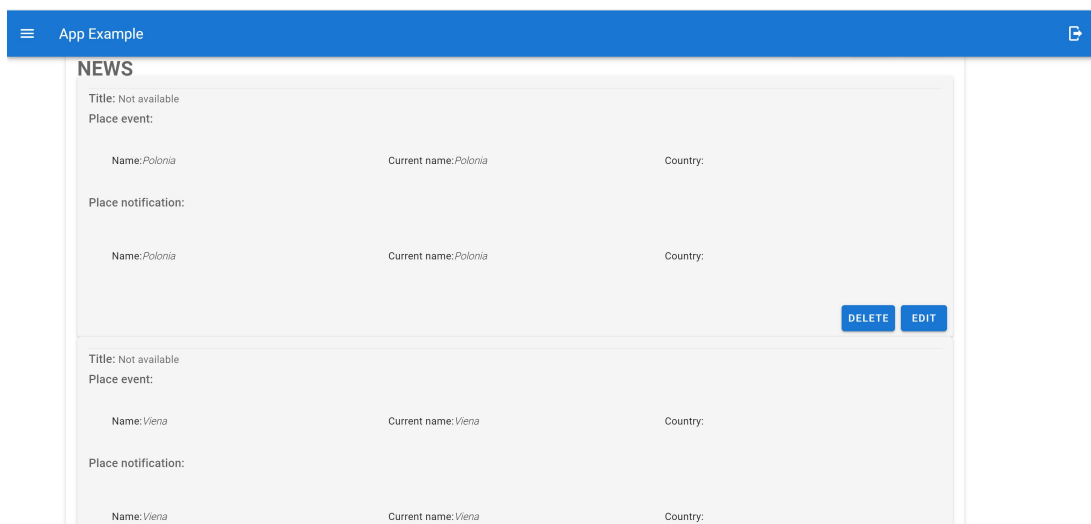


Figure 7.15: Gazette detail page (2)

Conclusions and future work

This chapter will discuss the degree to which the objectives set at the beginning of the project have been achieved, the lessons learned and future implementations that can enhance the functioning of the application.

8.1 Objectives

- A web application has been fully developed that fulfills the needs of the client and will allow to improve the study of the gazettes of the 17Th century.
- A map has been implemented that allows users to interact with it in order to control the output of the information allowing to show what meets the user's demands.
- A program has been developed to manage information about publishers, places and gazettes.
- Software has been implemented that performs the necessary functions for the application to function properly.
- The application has a friendly, intuitive, and easy-to-use web interface to improve the user experience.
- Ability to face new problems and to solve them by using the knowledge obtained throughout the university degree

8.2 Lessons learned

- It has been possible to get to know and become aware of the planning and workload of a real project. As well as being able to apply a development methodology in a project,

getting to know first-hand the importance of the good use of them, in order to ensure the quality of the final product.

- Existing knowledge about back-end technologies has been improved through their use during the project: Sprint, Hibernate, Rest servers...
- Knowledge has been acquired about the leaflet library that has allowed the implementation of the map and about how to manage the spatial data in the database.
- It has been possible to get some more experience on how to improve the user experience by offering a pleasant and intuitive web interface using Vue and CSS styles.

8.3 Future work

- Internationalization of the application so that the texts can be translated easily.
- Allow exporting as video the animation that shows the evolution of the publication and edition of the gazettes, in order to make the information more accessible.
- Data import and export, using a standard (e.g. CSV) to allow a user to easily enter large sets of data into the application so that they do not have to be entered one by one.

Appendices

Appendix A

Mockups

This section will show all the mockups made in the initial phase, explained in Section 3.2.2. These screens are a preview of what was wanted to be able to plan the project work and have an idea to start with, so there were changes to the screens of the final application.

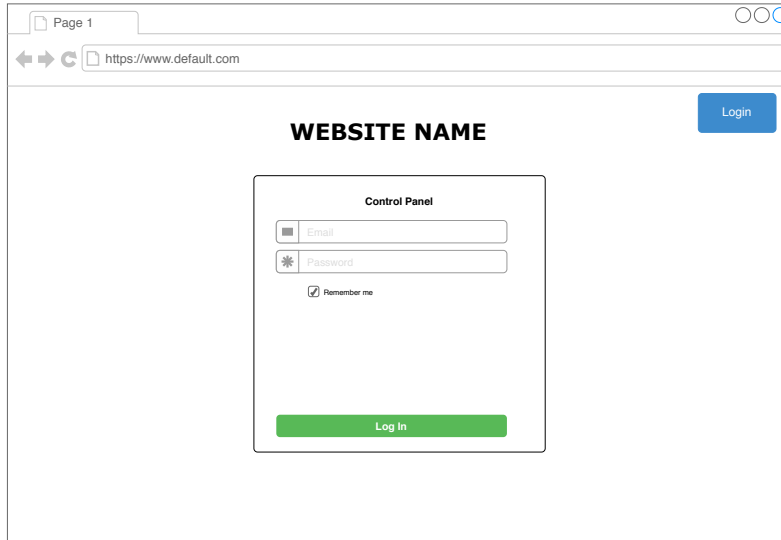


Figure A.1: Login page prototype

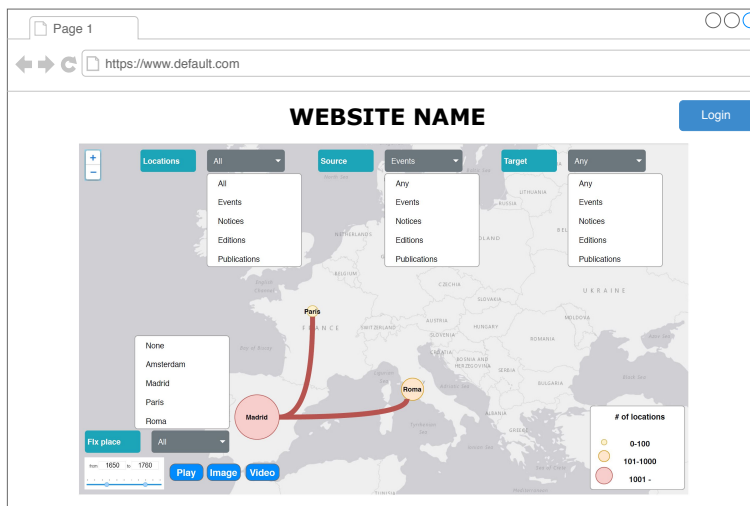


Figure A.2: Main page prototype

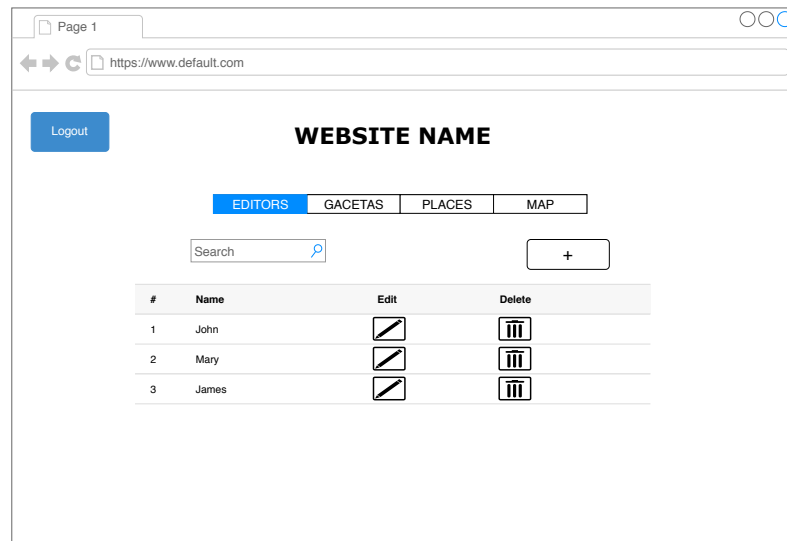


Figure A.3: Editors list prototype

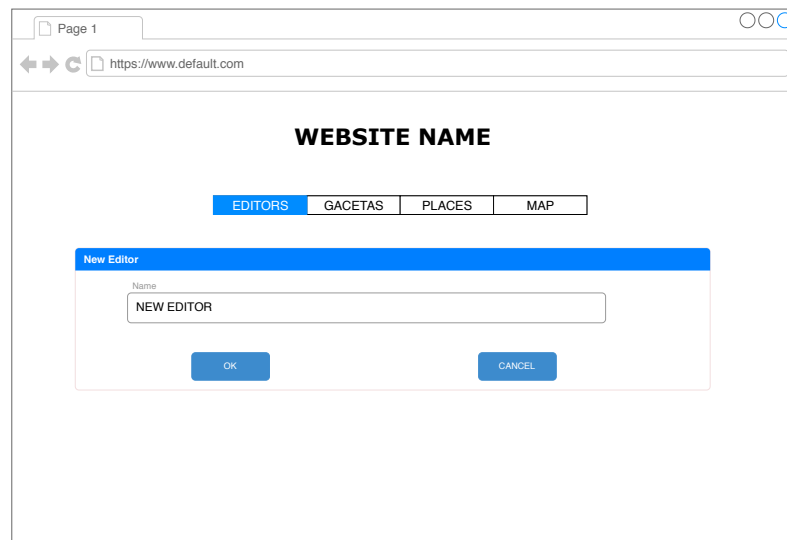


Figure A.4: Editor form prototype

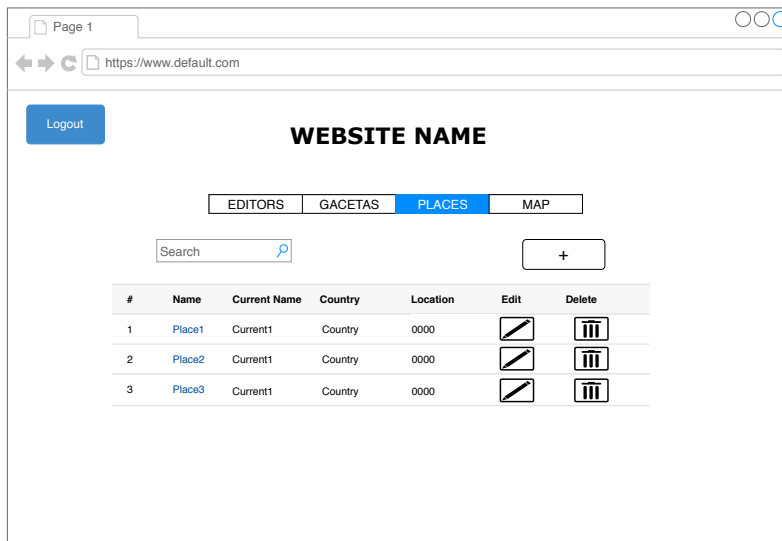


Figure A.5: Places list prototype

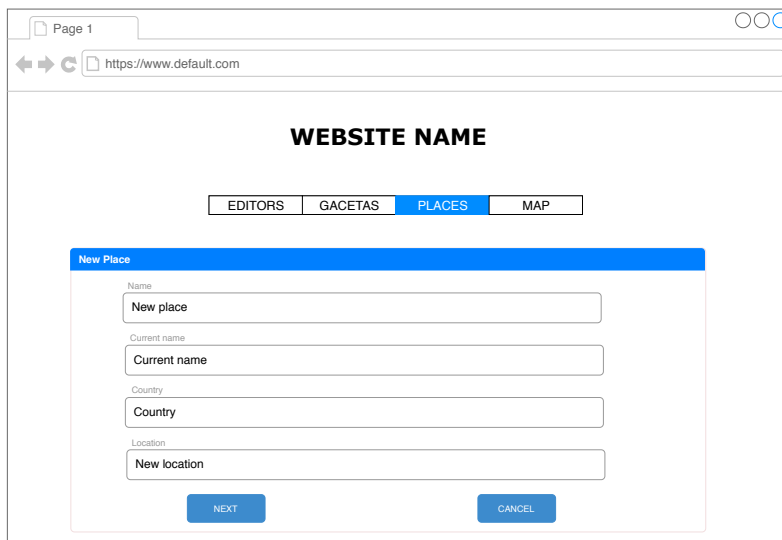


Figure A.6: Place form prototype

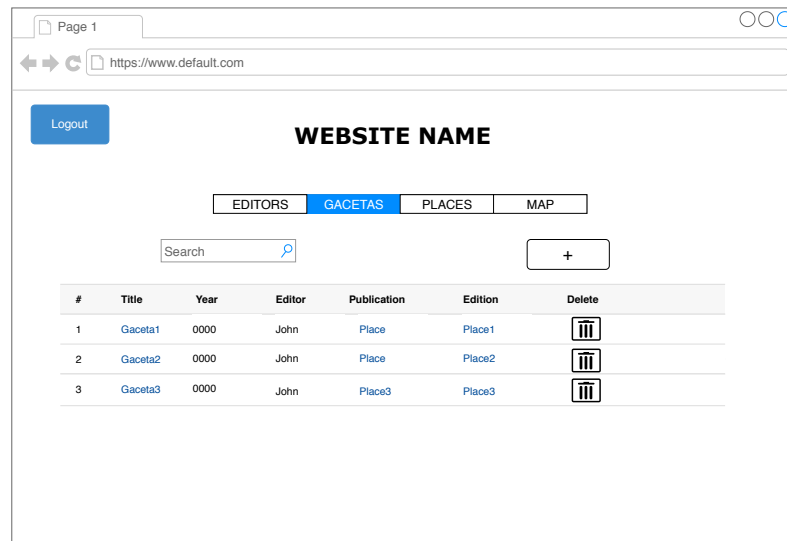


Figure A.7: Gazettes list prototype

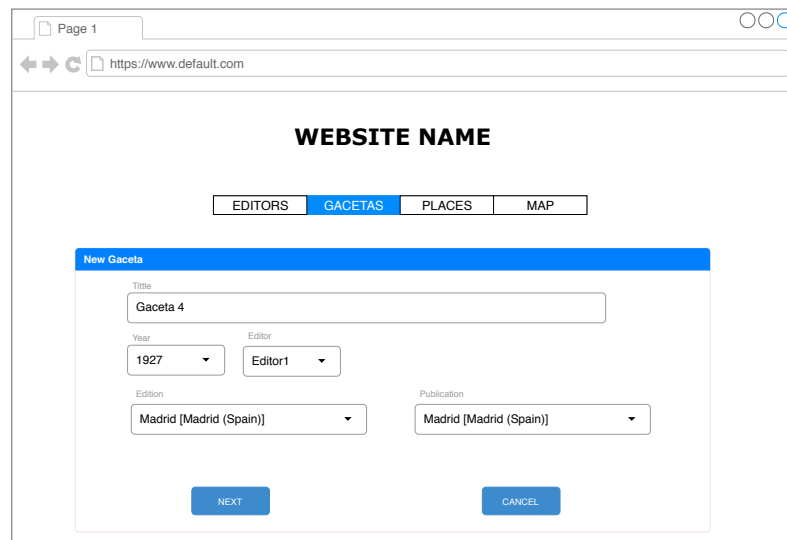


Figure A.8: Gazette form prototype

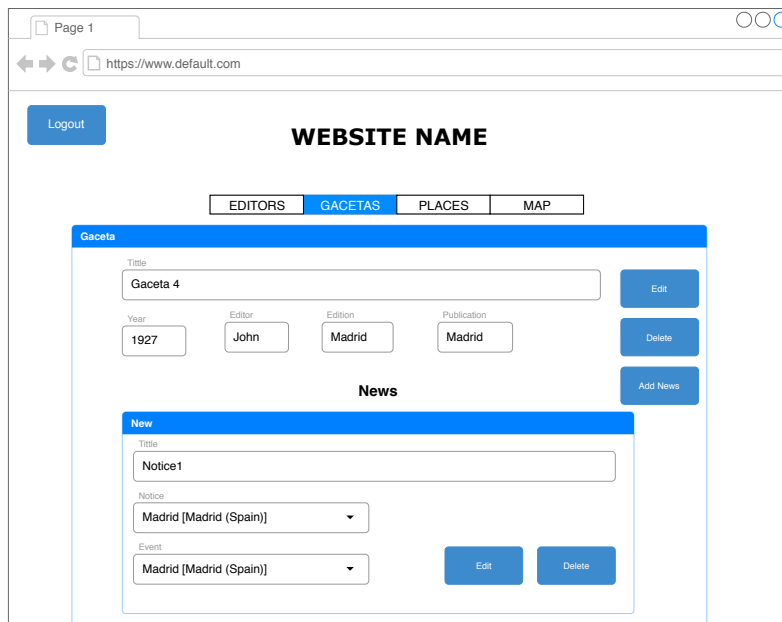


Figure A.9: Gazette detail prototype

Glossary of Acronyms

- API** *Application Programming Interface.*
- CRUD** *Create, Read, Update and Delete.*
- CSS** *Cascading StyleSheets.*
- DAO** *Data Access Object.*
- DBMS** *Data Base Management System.*
- DDBB** *Data base.*
- DTO** *Data Transfer Object..*
- HTML** *HyperText Markup Language.*
- HTTP** *HyperText Transfer Protocol.*
- IDE** *Integrated Development Environment.*
- JDBC** *Java Database Connectivity.*
- JPA** *Java Persistence API.*
- JPQL** *Java Persistence Query Language.*
- JSON** *JavaScript Object Notation.*
- MVCC** *Multi Version Concurrency Control.*
- MVVM** *Model–View–Viewmodel.*
- REST** *Representational State Transfer.*

SQL *Structured Query Language.*

UML *Unified Modeling Language.*

URL *Uniform Resource Locator.*

Glossary of terms

Back-end Part of a computer system that is not directly accessed by the user and is responsible for storing and manipulating system data.

Feedback information or statements of opinion about something, such as a new product, that can tell you if it is successful or liked.

Framework Reusable software environment that provides particular functionality to facilitate the development of a computer application.

Front-end Part of a computer system with which the user interacts directly.

Issue Unit of work to complete an improvement in a system.

Scrum Set of practices used in agile project management that emphasize daily communication and the flexible reassessment of plans that are carried out in short, iterative phases of work.

Sprint In Scrum it is a prefixed interval during which a "Done or Finished" product increment is created.

Bibliography

- [1] “Die fuggerzeitungen website,” (consulted in June 2020). [Online]. Available: <https://fuggerzeitungen.univie.ac.at>
- [2] “Six degrees of francis bacon website,” (consulted in June 2020). [Online]. Available: <http://sixdegreesoffrancisbacon.com>
- [3] “Resource trade.earth map website,” (consulted in June 2020). [Online]. Available: <https://resourcetrade.earth.com>
- [4] “Resource trade.earth information website,” (consulted in June 2020). [Online]. Available: <https://resourcetrade.earth/about#section-5>
- [5] “Vue website,” (consulted in June 2020). [Online]. Available: <https://vuejs.org/>
- [6] “documentation about vue router,” (consulted in June 2020). [Online]. Available: <https://router.vuejs.org>
- [7] “Vue-guide website,” (consulted in June 2020). [Online]. Available: <https://vuejs.org/v2/guide/>
- [8] “Node.js website,” (consulted in June 2020). [Online]. Available: <https://nodejs.org/es/docs/guides/getting-started-guide/>
- [9] “Css definition,” (consulted in June 2020). [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets
- [10] “Html definition,” (consulted in June 2020). [Online]. Available: <https://en.wikipedia.org/wiki/HTML>
- [11] “Javascript definition,” (consulted in June 2020). [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- [12] V. Agafonkin, "Leaflet website," 2019, (consulted in June 2020). [Online]. Available: <https://leafletjs.com/>
- [13] "Java website," (consulted in June 2020). [Online]. Available: <https://www.java.com/es/>
- [14] "Spring website," (consulted in June 2020). [Online]. Available: <https://spring.io>
- [15] "Spring boot website," (consulted in June 2020). [Online]. Available: <https://spring.io/projects/spring-boot>
- [16] "Hibernate website," (consulted in June 2020). [Online]. Available: <https://hibernate.org/>
- [17] "Hibernate spatial website," (consulted in June 2020). [Online]. Available: <http://www.hibernatespatial.org/>
- [18] "Geotools website," (consulted in June 2020). [Online]. Available: <https://geotools.org/>
- [19] "Postgresql website," (consulted in June 2020). [Online]. Available: <https://www.postgresql.org/>
- [20] "Postgis website," (consulted in June 2020). [Online]. Available: <https://postgis.net/>
- [21] "Agile development definition," (consulted in June 2020). [Online]. Available: https://en.wikipedia.org/wiki/Agile_software_development
- [22] "Scrum website," (consulted in June 2020). [Online]. Available: <https://www.scrum.org/>
- [23] "User stories explication," (consulted in June 2020). [Online]. Available: <https://www.atlassian.com/es/agile/project-management/user-stories>
- [24] "Scrum-guide website," (consulted in June 2020). [Online]. Available: <https://www.scrumguides.org/scrum-guide.html>
- [25] "Gitlab website," (consulted in June 2020). [Online]. Available: <https://about.gitlab.com/>
- [26] "Toggle website," (consulted in June 2020). [Online]. Available: <https://toggl.com/>
- [27] "Maven website," (consulted in June 2020). [Online]. Available: <https://maven.apache.org/>
- [28] "Draw.io website," (consulted in June 2020). [Online]. Available: <https://www.draw.io>
- [29] "Eclipse website," (consulted in June 2020). [Online]. Available: <https://www.eclipse.org>
- [30] "Visual studio code website," (consulted in June 2020). [Online]. Available: <https://code.visualstudio.com/>

BIBLIOGRAPHY

- [31] “Latex website,” (consulted in June 2020). [Online]. Available: <https://www.latex-project.org>
- [32] “Staruml website,” (consulted in June 2020). [Online]. Available: <http://staruml.io/>

