Publishing and Viewing Data with a Mediawiki Solution and MVC Architecture

Tiago José Ribeiro Teixeira

Instituto Superior Técnico de Lisboa, Portugal

Keywords: Music · Musicians · Musical Art · Information Systems · Ontologies · Web

Abstract. This project aims to develop an information system that allows to manage all the information about musicians and professionals of this type of art, who lived in Portugal, between 1750 and 1985. Currently, this information is spread over several digital platforms and also handwritten ones, which does not turn easy the user's access to it. For this reason, the primary goal of this project is to create an information system on the Web that aggregates all the available dataset but preserving all the existing information in its original shape. In order to create this information system, an ontology will be built using Wikibase and Mediawiki technologies. While creating and defining the ontology, two web applications will be developed, one of which will be assigned to the management of the ontology and the other to view the data, the task that will be developed by me as the main focus of my dissertation.

1 Introduction

The historical context of music in Portugal focuses particularly on the pieces created and developed by portuguese or foreign composers who passed through Portugal, at some point in their lives. However, there is an evident gap in the general knowledge regarding musical institutions and the musicians themselves, both in terms of their personal life and in the way that this personal context may be related to Portugal. In concrete terms, the motivation for this project represents a continuous collection over time of a collection of both existing and new data, about people who had music as their main activity or profession, in Portugal between the years 1750 and 1985. Such system will bring enormous social and cultural advantages as it will converge all the dispersed information in one location only. It will also allow self-correction (the biggest public exposure will allow a greater and more accurate way of crossing information) and the creation of a more complete profile for each element, since each database may contain different and complementary information about the same person.

2 Related Work

2.1 Wikidata,Wikimedia and Mediawiki

Mediawiki[1] will be the basis technology of our project. It is a free technology prepared to install on an online server and to receive millions of accesses per day. It was developed by the Wikimedia Foundation[2]. Uses PHP[3] technology to process and display data stored in its MySQL[4] database. Mediawiki has been optimized to efficiently handle large-scale projects in the order of terabytes and with hundreds of thousands of accesses per second. This technology is available in more than 300 languages, with more than 900 different configuration settings and with the possibility of incorporating more than 1900 extensions that allows the activation of new functions as also some changes to the existing functions. One of the extensions that we will use in our project will be Wikidata[5]. Wikidata is essentially characterized by items, each of which is described with a label, description and one or more alternative names. Statements describe the characteristics of an element through a property and a value. For example, a person can add a property to indicate where he was educated and set as value the place where he was educated. In relation to a phsyical location, it's possible to create properties for the geographic coordinates, specifying the values of longitude and latitude. In the domain of our project, we can define a "date of birth" property that will have as value the date of birth of the musician in question. A property that links to an external database is called an identifier.

2.2 Mediawiki API

When MediaWiki is installed, it has a REST API that allows managing information from the repository through several endpoints. For the development of the data visualization interface, the API will assume one of the most important roles in the system, being responsible for all types of information extraction. The client's side makes a request for information to the server using the HTTP protocol and receives the response in a standard format, usually in JSON[6] format. An order consists of an endpoint and a set of parameters that will be used to filter the information. There are two types of orders that can be placed: GET and POST. For a GET request, a parameter would be a query string present in the URL. For a POST order, the parameters are formatted in JSON format and included in the order. In the query string present in the URL, the "action" parameter is added. This parameter tells the API which action to take. The most popular action is the query (the URL must then contain action = query), which allows you to extract data from a wiki system. At last, the format parameter is

[1] https://www.mediawiki.org/
[2] https://www.wikimedia.org
[3] https://www.php.net
[4] https://www.mysql.com
[5] https://www.wikidata.org/
[6] https://www.json.org/json-en.html

included, which informs the API in which format we want to obtain the results. The recommended format is JSON.

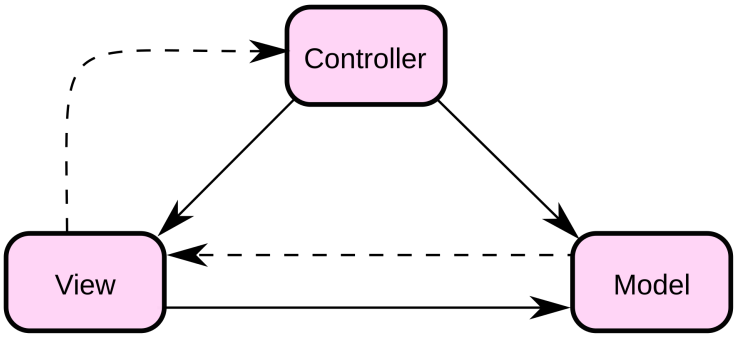

Fig. 1. MVC Architecture model diagram

2.3 Django: Web Development framework

Django is a framework for web development using the python programming language and was created in order to accelerate the development of websites based on databases. As mentioned in the previous section, "inside" Django is the MVC architecture and a set of open source and python programmed libraries. The motto of this technology is "Don't repeat yourself". Like Python, Django focuses on inefficiency, allowing the programmer to do almost all the tasks with minimal code effort. In addition to the advantages already mentioned, Django is also scalable, robust and fast growing because of the large community of developers and a robust set of integrated components. You can either access or create JSON and / or XML data files and handle relational database systems such as Oracle, MySQL, SQLite and PostgreSQL.[1] In a traditional website, a web application waits for HTTP requests from the browser or another "client". When a request is received, the application calculates what needs to be done based on the URL and, possibly, the information present in the POST or GET data. Depending on what is needed, the application can read or write information from a database or perform other tasks to respond to the request. The application will then return a response to the browser, usually dynamically creating an HTML page for the browser to display, inserting the data generated in the right tags of the HTML page.

2.4 Data Management Application

As referred in the previous sections, a data management application was developed by other student also in his thesis. This aplication aimed to allow to manage

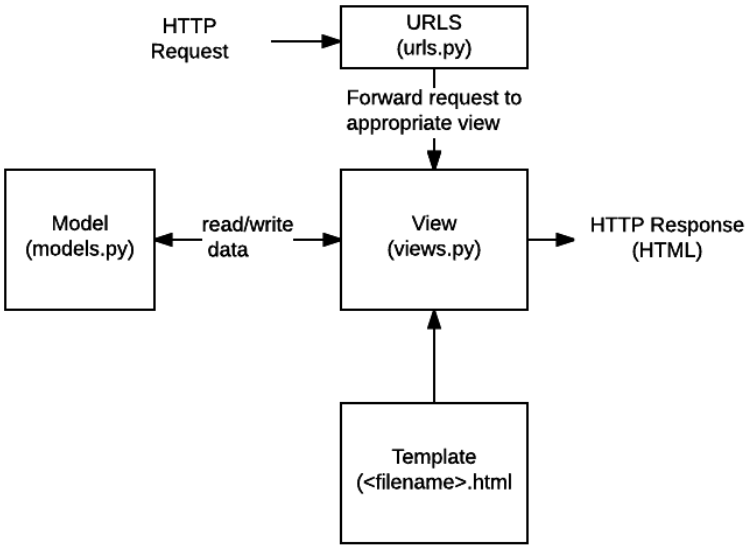

Fig. 2. Django application architecture

the ontology information (Mediawiki). As it was an application developed in the same project, there was an obvious collaboration between us. In the end, this application could: - Create account and login; - Import item set; - Export item set; - Create normalized form; - Create View; - Export view; - Manage items (Delete, Restore and Edit);

3 Problem Analysis and Conceptual Solution

3.1 Requirements

The application will mostly be used by users with few technical skills, so all of its operations must have a low level of difficulty and complexity to be executed. After conversations with the researchers involved in the project, they showed interest in the application projecting the data in attractive graphic elements such as dynamic geographic maps where all the information with geographic references can be shown, dynamic tables where the information shown can be chosen, timelines where dates corresponding to the selected properties can be shown, and other dynamic ways of visualizing the data. We then assumed that the application would have to provide enormous versatility with regard to the visual and aesthetic configuration of the final interface for viewing the views.

4 Constraints

The application will extract all the information through the Mediawiki API, which represents some difficulties since the referring documentation is not the most complete and some theoretically simple operations represent more complex solutions. The most effective solution for extracting information would be through a queries execution system directly in our repository, something that is possible, for example, in the repository that supports Wikipedia [7], but to configure in a local repository like ours there is no concrete documentation on how to do it. As such, we are limited to the capabilities and operations available through the API. For operations to which the API cannot correspond, alternative and more complex solutions, naturally slower and less efficient, had to be developed.

5 Conceptual Solution

A view is a subset of all the objects in the system, where each object is made up of a set of properties and their values. For example, we can consider as property "Date of Birth" and its respective value a date that represents the date of birth of the corresponding object. Views can be created and managed by users of the system, plus they can be published meaning that each view will have its own website, which can be shared with anyone. So, if a user wants a website with all the Italian musicians who arrived in Portugal in a certain period of time, they can create a View with all the items that match that criterion and then publish it. When the "owner" of a view decides to publish it, he can define a number of options regarding its aesthetic presentation on the site, selecting a wide range of options that ultimately result in the view's "official site". The developed application aims precisely to serve the configuration of the view publication by defining a visual interface for viewing it. The chosen and idealized solution would have to correspond to the requirements indicated by the users implementing the identified use cases. Considering that the total configurability of the application was a central necessity, the entire solution was developed around this aspect. As mentioned in the previous chapters, the data is stored in a Mediawiki repository and all operations and interactions with the data are performed through the API built into this system. The creation of "views", for later publication, is done in another application also developed within the scope of this project. It is also in this data management application that the "connection" to the view visualization application is made. Given the level of technical skills of users, this process must be as easy and accessible as possible.

6 Use Cases

With the specified requirements, the following relevant study cases were defined for further implementation in the application. All of them are focused on data

[7] https: // www. wikipedia.org

visualization elements and their flexibility. They will be accessible to the administrators of their sub datasets (views).

- **Edit names, titles and sub-titles**: The view administrator can edit the title of the view, the sub-title and also the titles of each of the information sections (Information, Table, Map and Time Graph).
- **Choose the properties to be displayed in the table**: One of the application's sections is a table where the various properties related to each object are represented, where each line represents the information related to an object. The administrator has the possibility to define what properties he wants to see in the table.
- **Edit the display information order**: The view administrator can define the order in which visual elements can appear. For example, you can define that the table should appear first than the time graph and first than the map.
- **Choose the geographical properties to be projected on a map**: Out of all the properties that are able to be projected on the map, the administrator can choose the ones that are visible on the map.
- **Choose the objects to be projected on the map**: Out of all objects with properties that can be projected on the map, this definition allows you to choose which ones will actually be shown.
- **Choose the color of the geograpgic marker corresponding to each geographic property of a specific object**: From the properties that the administrator chose to project on the map, he can for each object define a marker with a different color for each one.
- **Choose the objects to be shown on the timeline**: This definition allows you to choose the objects whose related dates will be able to be consulted in the timeline.

7 Architecture

In our architecture, the Object Controller is made by an implementation of Mediawiki that will manage and support the Object Repository. All kinds of additional information such as the views elements or the aesthetic definitions of each view will be saved in a SQLite database. These elements are coordinated by the Application Controller. This controller is a Django application that connects to Mediawiki via HTTP and to the database via an internal API. The application Controller is also responsible for the interface shown to the user through an Apache [8] web server.

7.1 Application Controller

The application controller contains all the application logic for both front-end and back-end. It also contains the graphical interface with which users will interact. This interface is generated dynamically using the information stored in

[8] https://www.apache.org

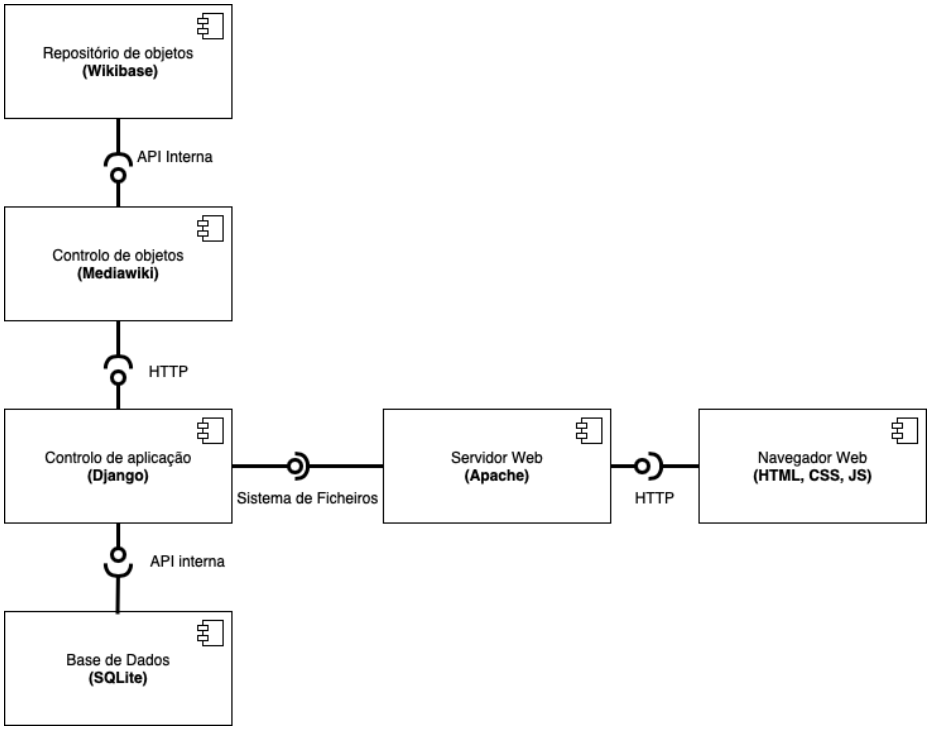

Fig. 3. Application Conceptual Architecture

the database and using the Mediawiki repository to extract information about the view objects.

7.2 Database

The SQLite database is linked to Application Controller via an internal API and stores all the information about views, from the objects of each view, to the aesthetic settings of each view. In Django, a model is a Python class where each column in the database is represented as an attribute of the class. We opted for the SQLite technology as it is a lightweight database and represented through a single file, which allows mobility.

7.3 Object Controller and Object Repository

The Object Controller represents a Mediawiki implementation and it's where all the properties and values of all the objects are stored. It also allows data management, but in a more complex way from the user's perspective. In this way, this type of management will be more suitable for a system administrator and not so much for the average user. The main way of interacting with the object

repository is through the Mediawiki API which provides a set of endpoints to filter the extracted information.

8 Implementation

In this section, will be described the entire application implementation process. The main components to be addressed will be the views configuration and the final interface construction for viewing the views.

8.1 Views configuration

When a user creates a view in the data management application, he is automatically the administrator of the view. Being an administrator, allows the user to publish the view. When publishing is done, the administrator can define a wide range of visual options. To store all the information about the style of the view, a new table was created in the database called *ViewStyle* in which each column represents an aesthetic field of the view visualization. As mentioned earlier, in the Django architecture each table in the database is represented by a class *Model* in which its attributes represent the columns of the database. After the publication, the administrator face a series of choices that he can make in order to configure the view. He can immediately configure the titles and sub-titles for both the view and each section of the page.

8.2 Interface Construction

For the construction of the final view visualization interface, the Application Controller extracts from the database all the settings saved by the administrator, and applies them to the base HTML page, which serves as the basis for all views. As a web framework, Django needs a convenient way to generate HTML code dynamically. The most common approach depends on templates. A template contains the static parts of the desired HTML output, as well as some special syntax that describes how dynamic content will be inserted. A Django template is a text document or Python string marked using Django template language. Some constructs are recognized and interpreted by the templates mechanism. The main ones are variables and tags. A model is rendered with a context, in the form of a Python dictionary, which contains the information that will replace the variables present in the template with the values due. The first step taken by the application to build the interface, starts by extracting an array from the database with the id's of the objects belonging to the view in question. After extracting the id's, the application will make a request HTTP to the Mediawiki API and receive a response in JSON format that contains all the properties and the respectives values, as well as possible qualifiers if they exist. The next step is to iterate the JSON response and for each object keep the values of each main property in Python dictionaries and if there are qualifiers, they are also saved in the same way. In the process of creating the dictionaries, there is a special

filtering if the data is a date or a geographical coordinate. For these two types of data, the set <Property >: <Value >is stored in a separate structure so that later in the template these two dictionaries are displayed in specific structures as they are a map and a time graph.

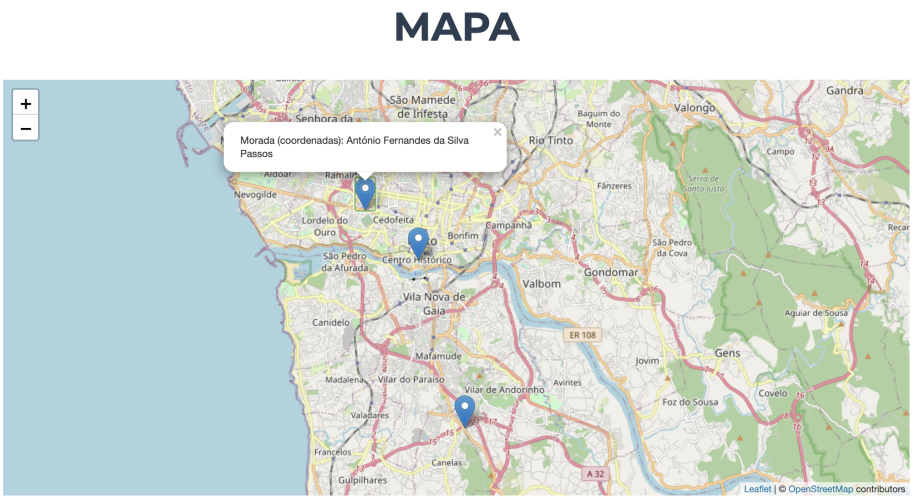

Fig. 4. Example of the map in the application.

8.3 MediaWiki and Wikibase Integration

To communicate with the Mediawiki API we mostly used the Python library Wikibase-api [9]. By default this library makes requests to Wikidata's endpoint but allows us to indicate the URL of our repository as a default. For each type of information, there is a specific format in Wikibase systems. Through the official documentation, it was possible to understand which formats the Wikibase system supports, mainly in the more abstract data types such as dates and geographical coordinates. Ordinary *strings* do not need any special treatment.

9 Stabilization of the Data Management Application

I will present in this section the fixes that were made in the data management application.

[9] https://wikibase-api.readthedocs.io//downloads/en/latest/pdf/

Fig. 5. Example of the timeline in the application.

9.1 Exporting data by any kind of form

One of the features that was implemented was the option for users to be able to export data using any form that already exists in the system. Until now they could only export using the same standard format that the system allowed, with no flexibility for the user to export using another form (preferably imported by you) present in the system.

9.2 Fixing data import action

In the first attempts to import real data, some errors prevented the right importation to succeed. All errors were associated with the type of data present in the xlsx file. If a column in the table referred to a date, excel automatically changed the data type of that column to "date" and with integer numbers the same thing happened, but in this case it changed to the type "number". All data that was not in "text" or "general" format, the application could not read them, resulting in the application malfunctioning. Bearing in mind that excel assumes these types of data automatically, it did not make sense to instill in users the responsibility of constantly checking the types of data present in the file, for each column. Thus, the changes to be made would have to be at the application level so that the import process was as robust as possible. In this sense, after a complex analysis of the functional code responsible for importing the data, and after several debugs, it was possible to make this process independent of the type of data imported and, consequently, it was possible to import a first block of the actual data provided.

10 Evaluation

10.1 Methodology

The tests consisted of the following steps: **Introduction** - In this step, users were explained the purpose of the session and in what context it was inserted; **Performing tasks** - In this step, users performed the tasks described in the form and also answer three simple questions about the task they just completed; **General Appreciation** - In this step, after the execution of all the tasks, the users were invited to write a general appreciation about the application.

10.2 Results

In this section, the results of the tests with the users will be presented. The evaluation had 7 users, mostly people involved in the Profmus project. At the end of each task, each user replied if he completed the task successfully and indicated the level of difficulty in executing the task where: 1 – Extremely Hard, 2 - Hard, 3 – Relatively Hard, 4 - Moderate, 5 – Relatively Easy, 6 - Easy and 7 - Extremely Easy. The following table will show, for each user, the level of difficulty they reported for each task.

Table 1. User evaluation for each task

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13
User 1	7	7	7	7	7	7	7	7	7	7	7	7	7
User 2	7	7	7	7	7	7	7	7	7	7	7	7	7
User 3	7	7	7	7	7	7	7	6	6	7	5	6	7
User 4	6	7	6	7	7	7	6	6	6	5	7	4	7
User 5	7	7	7	7	7	7	7	7	7	7	7	7	7
User 6	7	7	7	7	7	7	7	7	7	7	7	7	7
Average	6.83	7	6.83	7	7	7	6.83	6.66	6.66	6.66	6.66	6.33	7

10.3 Discussion

Analyzing the results, it is easily noticeable that in general users performed all tasks easily. Tasks 8,9,10,11 and 12 were the ones that offered the less easiest execution in the users' perspective. The others were globally identified as extremely easy to execute. Next, the general comments made by all users at the end of the test sessions will be presented. Each of these comments includes opinions and suggestions about each task. It also includes a general opinion on the application and recommendations for the future. In general, the results of the tests were quite positive, resulting only in a few suggestions that aim to further

improve the functionality of the application. It is important to report that none of the users suggested significant changes to the application manual. Overall, everyone found the application very intuitive and easy to use, and some of the difficulties that arose were quickly mitigated with a quick consultation of the user manual.

11 Conclusions

Being part of the Profmus project allows a real perception of the void that exists in the search and maintenance of the historical domain of music in Portugal. At the same time that we realize the cultural wealth that our country contains, we also realize that this wealth is not always well used and esteemed. The ideal solution was to group all the information collected in a single information system. The first steps were taken with the collaboration of another student, in the elaboration of his final master's thesis. He was responsible for coordinating the collection of information even though this process is progressive over time and for developing an application responsible for managing the information collected. At the end of his thesis, the application he developed was responsible for all direct actions on the data, such as importing, exporting, deleting, etc. One of the actions also available is the creation and management of views, which are subgroups of information. For example, a user can create a view that contains all the musicians who were born in 1800. The final objective of these views is their publication, which involves creating a website for viewing the data contained in it. One of the requirements initially defined is that the administrator of a view (who created it) can define the visual and graphic characteristics of the "site" of his view. This thesis focuses precisely on the development of the application responsible for publishing these elements. During a first stable version of the application, several tests were done with some of the project researchers, where they performed a set of tasks and in the end were able to give a general opinion of the application, giving (or not) suggestions for improving the application. The users feedback was very positive. All tasks were performed successfully and in general most tasks were considered "easy" or "very easy" to do. The application was also considered intuitive and simple to handle. Of all the comments made in the scope of the tests, I highlight the importance of two in specific, made by two of the main coordinators of the Profmus project. The compliments and suggestions included in these two comments, conveyed a great degree of satisfaction and a sense of accomplishment.

References

1. Indonesian Association for Pattern Recognition. International Conference (1st : 2018 : Tangerang, I., Universitas Bina Nusantara, Institute of Electrical and Electronics Engineers. Indonesia Section., Institute of Electrical and Electronics Engineers. Indonesia Section. Computer Society Chapter., Institute of Electrical and Electronics Engineers: The 1st 2018 Indonesian Association for Pattern Recognition International Conference (INAPR) (1), 218–222 (2018)