

POLITECNICO DI TORINO

Corso di Laurea Magistrale in  
INGEGNERIA INFORMATICA (COMPUTER ENGINEERING)

# **Medieval municipal buildings: development of a digital atlas to support historical research**



## **Supervisor**

prof. Alessandro Fiori

## **co-Supervisor**

prof. Silvia Chiusano

prof. Andrea Longhi

## **Candidate**

Andrian Garaba

March-April 2020



# Contents

<b>List of figures</b> .....	v
<b>List of tables</b> .....	vii
<b>1. Introduction</b>	1
<b>2. Related works</b>	3
2.1 “Tecniche murarie tradizionali: conoscenza per la conservazione ed il miglioramento prestazionale”.....	4
2.2 Carta di Rischio.....	5
2.3 THEMAS.....	6
2.4 ResCult.....	6
2.5 HERMeS.....	7
2.6 Arches.....	7
2.7 Conclusions.....	8
<b>3. Information system technologies</b>	9
3.1 Deployment System.....	9
3.2 Database Management System.....	14
3.3 Frontend development framework.....	18
3.4 Backend development framework.....	22
<b>4. Information System Architecture</b>	29
4.1 Database System.....	31
4.2 Software infrastructure.....	36
4.3 Platform functionalities.....	37
<b>5. Use cases</b>	39
5.1 Configuration.....	40
5.2 Insert/edit of a new source.....	43
5.3 Insert/edit of a locality.....	45
5.4 Insert/edit of the researched building or city building.....	47
5.5 Insert/edit of a new descriptive element.....	49
5.6 Translation process.....	53
5.7 Users management.....	54
5.8 Statistics dashboard.....	56
<b>6. Conclusions</b>	59
<b>Bibliography</b>	61



# List of figures

3.1 Containerized applications.....	13
3.2 General structure of MongoDB collection.....	16
3.3 Aggregation pipeline.....	18
3.4 MVC structure and interactions between components.....	23
3.5 Django Architecture.....	26
4.1 Docker container distribution of our platform.....	30
4.2 Conceptual Data Model of “Atlante dei palazzi comunali”.....	32
4.3 Navigation bar menu – contained in private_base.html.....	36
4.4 Sitemap of our web application.....	37
5.1 Use Case UML Diagram.....	40
5.2 Configuration workflow.....	41
5.3 Entering in configuration page for inserting new type of source.....	41
5.4 Inserting new type of source form.....	42
5.5 List of already inserted Source types.....	42
5.6 Sources management workflow.....	43
5.7 Entering in source page.....	44
5.8 Inserting new source form.....	44
5.9 Form management buttons.....	45
5.10 List of introduced sources.....	45
5.11 Form for inserting new locality.....	46
5.12 The list of already inserted localities.....	47
5.13 Entering in building or city building pages.....	48
5.14 Form for inserting new building.....	48
5.15 Entering in the building’s short description page.....	49
5.16 Form for inserting the short description of the building.....	49
5.17 Entering into descriptive elements insertion/display pages.....	50
5.18 Descriptive elements management workflow.....	50
5.19 Descriptive elements insertion form.....	51
5.20 Descriptive elements view page.....	51
5.21 Block for defining descriptive element characteristics.....	52
5.22 How to enter the translation module.....	52
5.23 Insertion form for new translations.....	53

5.24 The result of the search.....	53
5.25 Differences between the same piece of form displayed in French and Italian languages...	54
5.26 Reset password form.....	54
5.27 Login form.....	55
5.28 User profile page.....	55
5.29 Change password page.....	55
5.30 Homepage dashboard.....	57

# List of tables

2.1 Comparison of functionalities between platforms .....	8
3.1 Overview on aggregate operations.....	17
3.2 The possible stages of aggregation pipeline.....	17
3.3 Main characteristics of the compared frontend frameworks.....	20
3.4 MVC components description.....	23
3.5 The strengths of Django.....	27
4.1 Location collection schema.....	33
4.2 Building collection schema.....	33
4.3 City_building collection schema.....	34
4.4 Translation collection schema.....	34
4.5 Source_type/Elements_type collection schema.....	34
4.6 Source_type example.....	35
5.1 Example of aggregate method.....	56



# Chapter 1

## Introduction

Nowadays, governments, business structures, and the scientific community face the task of filling the registers and information systems with the necessary data to automate, accelerate, simplify interaction, implement analytics and forecasting, while in parallel eliminating human labor and the human factor from these processes. The digitalization of paper-based documents is a significant part of this global process. But why paper information is a bad thing? Paper archives require much space for themselves, special storage conditions; besides, documents may be damaged, stolen, or destroyed due to natural or human-made disasters.

Digitizing of paper archives provides access to the necessary information from anywhere where it is possible to reach the Internet and to several users at the same time. Translation of documents into electronic form is done with their simultaneous systematization, structuring, and backup creation. The availability of backup copies of documents reduces the probability of their destruction or loss. Also, by implementing security and logging policies, documents with limited access can be viewed only by people with the appropriate authority, and all actions on the information system are recorded. As well digitalization process affects paper archives by themselves by the possibility of releasing the occupied space or by optimizing the human activity in it.

After the translation of paper documents into digital form and creating search fields, they can be loaded into specialized systems - electronic archives. This approach provides several improvements comparative with physical storages: fast documents search, secure documents moving, protection against loss or damage to documents.

Paper storages are becoming more expensive and less efficient every year. Searching documents takes an unreasonably long time. Sometimes they can not be found due to the absence, or the location is different than the place where the search is done. In contrast, a search

in electronic archives takes a matter of seconds.

When the organization's structural units are physically removed from each other, this gives rise to a long wait in line for viewing the necessary documentation: first, find it in the archive, arrange paperwork for moving the originals, and then wait for the logistics process to complete. There is no need to go anywhere for digital copies, and all interested parties get access to them at the same time. And in the case that paper originals are required, finding them in the repository and arranging a move is very easy.

Long-term practice shows that not a single paper archive can be considered a 100% reliable place of storage. It doesn't matter if the document was accidentally thrown away, lost in piles of paper or destroyed - it all the same means that it needs to be restored, spending time and money. Electronic archives are based on the fact that documents always have backup copies, their restoration eliminates paperwork and is quickly performed.

Such an electronic archive is also the subject of these thesis. The information system "Atlante dei palazzi comunali" serves to collect heterogeneous information on historical buildings that are the object of study, and to provide researchers with a digital tool for the analysis and sharing of the collected material. In general, the platform structure can be divided in two parts: public site and private site.

The public part of the portal is used to present the project of the "Atlante dei palazzi comunali", and to allow the visitor to perform some predefined searches, view statistics on the collection of buildings or their characteristics, and acquire some information of detail on each listed building.

The private part, which has the most complex requirements of the thesis, is described a little more in detail. It has a multifunctional user interface, which includes management (insert/edit/view) forms of all platform entities such as sources, locations, buildings, municipal buildings, descriptive elements, and users. Some of them can be dynamic, based on the selection of information to be inserted/displayed.

In the next Chapter (2), it will be briefly described similar platforms and Information Systems, focusing in particular on the similarities and differences with the "Atlante dei palazzi comunali" and advantages and disadvantages of each over. In Chapter 3, it can be found information about technologies used to create the information system "Atlante dei palazzi comunali", their main features, pluses and minuses, and the reasons why each of it was chosen. Chapter 4, instead, will contain a description of the architecture of the platform: detailed information about the data model and Django architecture. In Chapter 5, we will present some use-case scenarios to illustrate how the created product, meets the requirements and provides the desired results. The last Chapter (6) conveys some final conclusions about the project and its future evolving.

## Chapter 2

### Related works

In general, given the history of humankind, the culture reproduces, improves, translates, projects a range of things that correspond to familiar or innovative patterns and which ensures the connection between past and future for present generations. It is believed that the structure of culture is the most complex in the world. On the one hand, these are material and spiritual values already accumulated by society: a stratification of eras, times and peoples, fused together. On the other hand, it is today's human activity influencing, modifying, and transmitting the heritage of the past generations to those who will replace the existing ones. To identify tangible cultural features, the introduction of certain concepts is necessary. Such a concept is the cultural heritage and its contents elements as cultural memory, cultural monument, geographical environment, etc. The official content of the concept of cultural heritage is recorded in UNESCO's "Convention concerning the protection of the world cultural and natural heritage" [1]. UNESCO was founded in France in 1945, and one of its primary functions until today remains the unification of states and nationalities to preserve and protect the cultural and natural values of humanity.

"Cultural heritage" is a relatively young term and is used today in national legislations and international documents as a confirmation of the process of formation in modern society of a systematic approach to world culture, cultural values, and environmental protection. In general, the conceptual plan for the protection of cultural heritage is influenced by the development of scientific ideas about monuments and by the changes in the political and ideological situation in the country (government policy in the field of culture, expressed primarily in legislative acts on the protection, restoration, and use of monuments). Until reaching the modern concept of cultural heritage, over time, many cultural monuments were destroyed or damaged due to

ignorance, utilitarian attitude, interest in the material value of the goods, lack of national and international organizations in the field, the imperfection of laws.

There are many ways of preserving and transmitting the experience of the past. It is oral language, writing, all types of art without exception. Also, as a part of this process are libraries, museums, art galleries, exhibition halls, architectural structures, temples, monasteries, palaces, garden and park ensembles, meaning, everything that keeps the history of the people and the memory of the past. Thus, the translation of social memory occurs with the help of specific systems, which we can call monuments. A monument is an object that is part of the cultural heritage of the country, people, humanity, and it is created to immortalize the memory of certain events and people.

Today, thanks to the significant implication of international and national organizations for in-depth studies and development of the classification of monuments, the concept of cultural heritage is rebuilt and sounds like a combination of material and spiritual monuments. It includes stationary monuments (monuments of urban planning, architecture, history, archeology, monumental art, nature and so on), movable monuments (objects of pictorial art, manuscripts, archives and so on) and the so-called spiritual monuments (specific forms of housekeeping, beliefs, traditions, technologies, and so on).

For the academic environment in the field of architecture and culture, for organizations responsible for the protection of cultural heritage sites, for the state institutions responsible for culture and even for the enthusiasts who care about the cultural environment of the region where they work or live inventories of architectural and cultural objects are crucial tools. They serve as an everyday information resource to analyze the risks of damage of some historical buildings, to allow studying in more detail the history of a region based on its culture, to intervene in time with the application of the rules and laws in the field of cultural heritage or even to draw public attention to valuable cultural items created by humankind.

Digital inventory platforms that use the capabilities of modern information technologies have the potential to provide different needs and uses. While some put key on managing internal processes for storing, checking, editing, and deleting data, others focus on the user experience and granting controlled and secure access to information. Some platforms combine the two approaches described above. In the majority of cases, there is not a single software that can meet all project requirements. To choose the right platform is useful to consider all aspects of the project and analyze all available tools thoroughly.

Lately, a lot of advanced open-source software (a free license that allows developers and users to study, modify, improve, and distribute software) has been created dedicated to cultural heritage recordings. In the following sections we will describe the most relevant works that represent the state-of-art in the cultural heritage field.

## **2.1 “Tecniche murarie tradizionali: conoscenza per la conservazione ed il miglioramento prestazionale”**

The project [2] contains a spatial information system that aims to study building technologies of the 13-19 centuries in the region of Sardinia (Italy) for their further knowledge, preservation, and promotion. This architectural heritage is vast and heterogeneous in typology and function and is characterized by a deep stratification of the transformations that have occurred over time.

Managing the massive amount of data obtained during the study requires the development and implementation of a system for archiving, comparing data, and maintaining a relationship between information and territory, such as allowing cross-reading of geographical distribution.

The project was designed by the Department of Civil and Environmental Engineering and Architecture, University of Cagliari. The implemented information system is a database connected to the GIS and a WebGIS interface through the web map service (WMS). The entire infrastructure is developed and implemented using open-source software components. The database is created in PostgreSQL and PostGIS (PostgreSQL spatial extension), which allows storing and managing object geometries and spatial data. Data entry is carried out through HTML and PHP forms. Access to the database is done with Quantum GIS (QGIS), and WebGIS was built using the open Leaflet Javascript libraries.

Even though this project enfold a narrow spectrum of research (building techniques) is very similar to our platform, where its researched objects match with the descriptive elements in ours. Other common features could be (i) determining the position and shape of the studied object on the map using GIS and (ii) a detailed description of it.

But having small purposes, this project also has some limitations compared to the requirements of our platform. The sources which describe studied elements are missing from it, and it is not possible to configure other types of descriptive elements for further research. The possibility to view the buildings in 3D, direct use of latitude and longitude to determine the position of the object and the simultaneous display on the map of all the neighboring buildings are the main advantages over our platform. But all these improvements can be added later in our project if necessary. Unfortunately, the system has not been published on the web until this moment.

## **2.2 Carta di Rischio**

The “Risk Map” [3] is a territorial information system of scientific and administrative support to state and regional bodies responsible for the protection of cultural heritage. It was developed by the “Istituto Superiore per la Conservazione” (formerly ICR - Istituto Centrale per il Restauro), and it is a system of experimentation and research on the territory, for knowledge on the risk of damage to real estate. In fact, it is a system of alphanumeric and cartographic databases capable of exploring, overlapping, and processing information around the potential risk factors that can affect the cultural heritage.

As the primary purpose of the “Risk Map” project differs from our project goals, they have in common only the position on the map module of the study object and its detailed description. In the rest, our attention was drawn only by the fact that it was one of the first projects in Italy (the first version was elaborated between 1992 and 1996), which placed the buildings with historical importance on the map. In the following years, some developments of the project followed, by modifying the interface, by absorbing the information on the real estate fund from other sources and databases, by creating similar regional projects, by diversifying the study categories.

## 2.3 THEMAS

**Thesaurus Management System** – "THEMAS" [4] is an open-source web-based system for creating, managing, and administering multi-faceted multilingual thesauri. It was designed by FORTH-ICS (Foundation for Research and Technology Hellas - Institute of Computer Science), one of the six institutes of the major national research center in Greece.

The main features of the system include the processing and management of semantic relations of treasure terms, easy navigation between interlinked terms, extensive search capabilities, and multiple presentation capabilities. "THEMAS" can be adapted to the requirements and needs of any research area as it has customizable configuration settings (e.g., Languages management – at the interface level and entity level with possibility directly to translate it in the platform, customization of the graphical representation and others). The workflow of the system uses well-defined user roles. It allows the collection of information from a large pool of users, with the responsibility of accepting, correcting, or rejecting any records by smaller expert groups specialized in the discipline of each studied entity, thereby ensuring the accuracy and consistency of the inserted materials.

The technical aspect of the project is as follows: the graphical user interface is a web application implemented with JavaServlets technology. The data storage and management are done using the open-source graph database Neo4j community edition. At the same time, for the designing of the studied objects, the principles of the TELOS [5] representation language was utilized. The Neo4j-sisapi library was used for communication between the web application and the database. The whole system is based on Java technologies and, therefore, can be installed on any operating system (Windows, Linux, Apple, etc.).

Analyzing this project and comparing it with the requirements of our project, as a similarity, drew our attention to the possibility of configuring in several languages the studied entities. Also, the existence of the categories like researched elements and the sources where these elements are mentioned can be treated as a typical particularity of both projects. As it is a project dedicated only to the description of cultural elements, there is no positioning in the building and the location of the studied object, and this can be treated as a big difference.

## 2.4 ResCult

ResCult [6] – is a project coordinated by Siti - Istituto Superiore sui Sistemi Territoriali, in which PoliTo was one of the partners. It had as objective to create a tool called EID (European Interoperable Database) for Cultural Heritage. It was designed to create a unique platform for Civil Protection, European Union, national ministers of Cultural Heritage, local authorities. Specifically, ResCult is working on the following three overall objectives:

1. Improvement of the Disaster Risk Reduction strategy (for prevention and resilience), according to the principles of Sendai Framework.
2. Increase cooperation and interoperability between EU member states for the sake of protecting Cultural Heritage (information sharing, interoperable protocols, best practices dissemination, alignment with EU policies/standards)
3. Enhancement of the capability of Civil Protection Bodies to understand/prevent/mitigate disasters impacts on Cultural Heritage

As it is described above, the ResCult platform is more related to the protection of architectural buildings, but in this project was realized some features which can be found in our requirements. Geolocation of the buildings, descriptive summary of the building, 3D-models (future enhancement for "Atlante dei palazzi comunali"), possibility to insert information by different researchers are the common elements of these two projects. Of course, comparative with ResCult, "Atlante dei palazzi comunali" should contain more detailed information about historical sources and descriptive elements of architectural monuments.

## 2.5 HERMeS

HERMeS [7] – is a Digital Collection of Historic Buildings, appeared as a result of long research conducted by **the National Technical University of Athens and Municipality of Hermoupolis**. It was a local project (only within the borders of town), with the objective to analyze and evaluate the vulnerabilities of historic buildings. A universal database was designed to record information about historic buildings and their actual or possible damages in order to achieve these goals. Apart from deteriorations, historic buildings are labelled in terms of type, architecture style, historical value, and geolocation. It is based on Omeka Project – a web publishing platform for sharing digital collections.

As with the previous project, HERMeS practically has the same goals and objectives: to prevent the deterioration process of the historic buildings and to focus on the restoration of which needs more immediate attention. As it initially had a small range of functionalities, it is hard to compare with "Atlante dei palazzi comunali". But some similarities can be found. Again, it is about geolocation and descriptive summaries for the buildings, and as well as ResCult Project, it is not possible to record details about architectural elements of the buildings.

## 2.6 Arches

The closest to the "Atlante dei palazzi comunali" are the Arches Project [8]. It is a platform for cultural patrimony inventory and management. It is open-source and was developed by Getty Conservation Institute and World Monuments Fund. As an open-source project, Arches is free to be used by any organization, and it can be customized without any constraints to face the requirements of the final users. Mainly the Arches Project can be used for the identification and inventory of heritage places and for the research and analysis of the physical structure and cultural impact of the historical places with identification, monitoring, and mapping of risks. Also, it can be useful in the process of identification of needs and priorities for investigation, research, management, conservation, and detailed planning of further actions in these directions. Therefore, it is an ideal instrument for raising attention and informing on a large scale (citizens, scientific environment, governments, decision-makers) about heritage patrimony, risks related to their physical damages, and possible conservation and management modes.

The default version of the platform is configured to record details about all types of historical buildings, cultural landscapes, heritage areas, and archaeological sites. It includes six types of resources which can be related to each other, as follows: Heritage Resource (building, archaeological site), Heritage Group (district), Activity (survey, partial repair), Historical Event

(earthquake, war), Actor (architect, a celebrity who has some links with the building, or organization which repaired the building), Information Resource (3d Model, photo, report). All the resources can be renamed in the process of customization. Arches allow creating a network of relationships between all Arches resource types, which permits, for example, to track several roles of people or organizations. Meaning, a person could be the architect and the owner of the researched building. This approach allows distinguishing such roles in the working process clearly. Besides, Arches can store location information in several formats (e.g., text description, address, cadastral data) and in different geolocation shapes (any combination of point, line, and polygon data) at the same time.

One of the main principles of Arches project design was the compliance with standards of IT, heritage inventory, and management, which leads to the creation of advanced, configurable, and content system. Arches' system architecture follows modern trends, using RESTful interfaces and an MVC pattern to separate representation of the data and the way it is getting from, or it is presenting to the user. Arches platform is built on python and Django and implements the following core libraries: Require.js, Backbone.js, jQuery, and Bootstrap. Also, it enables geospatial technology standards, like OGC standards, to support geospatial data interoperability, like GeoJSON, KML, and traditional GIS data format for representation of geolocation information.

The Arches platform has the most common entities with the "Atlante dei palazzi comunali" project specifications, as it also contains info about researched buildings (geolocation and building summary especially) and some info about sources (not detailed one). As well as other analysed platforms, Arches can't record information about elements of the researched historical building and cannot link them to sources that mentioned them.

## 2.7 Conclusions

Taking into account the above, we realize that there are a lot of projects that touch on the topic of research and conservation of cultural objects. But none can be considered standard one because it does not include all the possible functionalities that heritage protection organizations or scientific community may need.

	Sources management	Descriptive elements management	Geolocation	3D models	Descriptive summaries of buildings	Multilingual	User control management
<i>Tecniche murarie tradizionali</i>			×		×		
<i>Carta di Rischio</i>			×		×		
<i>THEMAS</i>	×	×			×	×	×
<i>ResCult</i>			×	×	×		
<i>HERMeS</i>			×		×		
<i>Arches</i>	×		×	×	×		
<i>Atlante dei palazzi comunali</i>	×	×	×		×	×	×

Tab. 2.1 Comparison of functionalities between platforms

In Tab. 2.1 we can see the distribution of functionalities between the presented platforms and the requirements of our project. Even if some similar projects are very close, we can conclude with certainty that none of them meet the requirements of our project. None of them has tools to configure researched entities directly by the user, very few support multiple languages and user management. And that is why the design and implementation from scratch of the *Atlante dei palazzi comunali* project is fully justified.



# Chapter 3

## Information system technologies

Today, in the age of informatization and computerization, the basis for solving many problems is information processing. Under the processing of information is understood any transformation of data from one type to another, carried out according to strict formal rules. In fact, in most cases, this process is carried out by information systems. An information system is a software package whose functions are to support reliable storage of information in the computer's memory, perform information-specific transformations and calculations specific to a given application, and provide the user with a convenient and friendly interface. Usually, the volumes of information that such systems have to deal with are quite large, and the information itself has a rather complex structure. Typical examples of information systems are banking and accounting systems, airline or train ticket systems, tax service systems, statistical systems, hotel reservation systems, and others.

In the process of developing an information system, after collecting the requirements from the client and until the design of the system itself, it is necessary to analyze and choose the technical tools with which the project will be developed. It involves choosing the programming languages, Database Management System, and platforms and technologies based on which the system will be designed, developed, and maintained. This chapter contains detailed information about the features of the chosen technologies and the motivations behind the decisions.

### 3.1 Deployment System

Deployment is a very important stage in the process of creating a software product. It includes all actions that make a software system ready to use. In other words, deployment is a process

(one way or another organized) of translating project code into working conditions on a specific machine. If it is to remember the beginnings of IT, all these operations were done manually by the programmers. However, fortunately, at the moment, we can benefit from systems that support automatic deployment, and because of this, it seems strange that many companies continue to do it manually. One of the solutions to have automatic deployment is containerization. It is a process of distributing and deploying applications in a portable and predictable way. It is achieved by packing the components and their dependencies into small, stand-alone, and lightweight execution environments called containers. They run in isolation from one another but share OS kernel.

Before using containers, virtual machines were one of the solutions to keep applications on the same host or cluster, isolated from each other, so they don't unnecessarily interfere with each other's operation or maintenance. But they have some drawbacks as each requires his specific OS, usually needs Gigabytes size on HW space and are difficult to maintain and upgrade. On the other hand, containers need few Megabytes on HW, uses much fewer resources, and start-up almost instantly.

Container's concepts have similar features with virtual machines, as independence and self-sufficiency. The container can be moved to any OS with a docker service on board, and the container works. It can perform its functions anywhere wherever it is launched. But containers have some particular features as well. Inside the container is the minimum necessary set of software used for the operation of the running process. This minimal software is not a full OS and doesn't need to be monitored or keep track of the free space on hard drives. Also, a different approach to virtualization is used - containers virtualize the operating system instead of hardware.

Docker [9] is a software platform for building applications based on containers (Fig. 3.1). It's straightforward in management, upgrading, migration, and is suitable for a wide range of tasks (from application development to testing). While containers as a concept have been known for some time, Docker, an open-source project launched in 2013, led to the promotion of the respective technology. Also, it contributed to the development of so-called cloud-native development (use on large scale of containerization and microservices in software development).

Before the creation of a Docker-containerized application, it is necessary to introduce some Docker components: Docker File, Docker Image, Docker Engine, and Docker Registry.

Docker File is a text file, which contains all the dependencies it needs to download and install to run the application. A Dockerfile specifies the operating system that underlies the future container. Also, it contains the languages, environmental variables, file locations, network ports, and other components it needs—and, of course, what the container does once we run it.

Docker Image is some set of layers. Each layer is the result of the work of the command in the Dockerfile. In other words, image is a template based on which the container is launched. From one image, it can be run several identical copies of this image, and as a result, several containers are launched. Also, after some manipulations, can be created a template from a container - a new image is obtained. In other words, a Docker image is an executable package that contains all the range of information needed to run an application: the code, runtime, libraries, environment variables, and configuration files. Docker container is a runtime instance of an image. In fact, it is what the image becomes in memory when it is executed (an image with the state, or a user process).

Docker Engine is the core of Docker, and it is based on client-server technology. Its main role is to create and run the containers.

Docker Registry is a public or private repository where Images are stored. The most popular one is Docker Hub – where can be downloaded containers from open-source projects and vendors, or unofficial images form users.

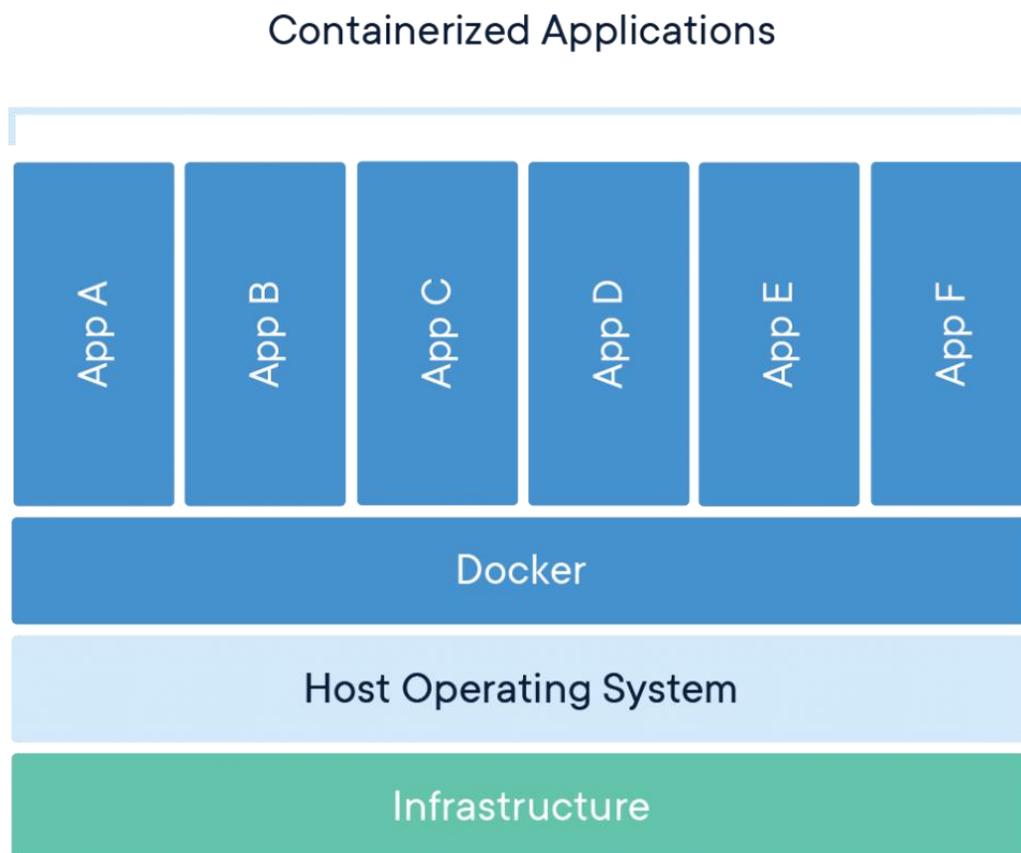


Fig. 3.1 Containerized applications

Creating and running containers is fundamental to modern software infrastructure, development, testing, and deployment, but this is not the end of the story. It is obvious that there is a need to coordinate containers somehow or other in order to work together in a single system. Docker-compose was created by Docker to simplify the process of developing and testing multi-container applications. It's a command-line tool that uses a specially formatted file to compile apps from different containers and run them in a linked way on the host. Docker Swarm and Kubernetes [10] offers more advanced behaviours in this direction and introduce a new principle called Container Orchestration.

Let's develop the Kubernetes topic in more detail and compare it with the tools offered by Docker. Kubernetes is an open-source system designed to automate the deployment as well as to scale and manage containerized applications. Automation is the keyword and one of the reasons Kubernetes is becoming more and more popular among the engineers and other professionals who are responsible for the operation of websites, infrastructures, and systems. At first, organizations used one of several open-source tools, including Kubernetes, or created their own to manage containers on a scale. However, the popularity of Kubernetes gradually rises, and nowadays, it becomes for orchestration what Docker has already become for containerization, the synonyms. In fact, Docker and Kubernetes do not exclude, but, on the contrary, complement each other as part of a more extensive system, and the question of choosing between them is simply not worth it. However, the Docker ecosystem has its own container orchestration solution - Docker Swarm, a functional analog of Kubernetes. It is fully compatible with Docker without needing special tools to manage containers. It is generally accepted that Docker Swarm is easier to use, while for most users, learning Kubernetes is much more difficult. Even from the moment of installation, Kubernetes needs increased attention and requires serious planning to work. For an error-free installation, it is necessary to consider the operating system and to know the IP addresses and roles of each node in advance. Because of that in test environments, as usual, Docker Swarm is used. But when time is coming to move huge containerized workloads into a production environment, Kubernetes becomes an irreplaceable tool. Considering the small number of containers used in this project, it is not cost-effective to use container orchestration tools (like Kubernetes) for such small projects, and all deployment can be done using the standard Docker tools.

In conclusion, Docker Containers permit to build enterprise applications that are easier to assemble, maintain, and move around than applications created traditionally. The most highlight advantages of Docker are isolation, portability, and composition. Docker containers keep applications isolated not only from each other but also from the basic system, and in the result is obtained a cleaner software stack and more control on system resources distributing. It also permits to keep code and data separated. Also, Docker container runs on any machine which supports containers runtime environment. Applications don't have to be adjusted related to the OS of the host, so both the application environment and the OS environment don't need additional configurations and installations. The application can be moved easily between systems, or ported to the cloud environment or exported from the programmer's devices in the process of developing, as long as on the target system, Docker and some third-party tools that might be in use with it, such as Kubernetes, are installed. Not least, Docker containers make it possible to create each component of business applications (web- server, database) into a functional unit with easily changeable parts. A different container provides each piece of application and can be maintained, refreshed, swapped out, and modified independently of the others.

## **3.2 Database Management System**

Before starting this project, it was necessary to choose a DBMS, and the central dilemma consisted of choosing a relational (SQL) or nonrelational (NoSQL) structure. Both options have their advantages, as well as several key features that should be taken into consideration.

Relational databases use Structured Query Language (SQL) to define and process data. SQL is one of the most flexible and widespread query languages, so choosing it minimizes the

number of risks, and it is especially useful for complex queries. Also, SQL has some limitations - building queries requires to have a predetermined data structure, and further changes in the data structure can affect the entire system.

Non-relational databases offer a dynamic data structure that can be stored in several ways: column-oriented, document-oriented, in the form of graphs, or based on key-value pairs. This flexibility adds some new features as it is not necessary to specify document structure in advance when creating it. Also, fields can be added while working with data, and each document may have a unique structure.

Usually, SQL databases are vertically scalable, meaning that it is possible to increase the load on a single server, by adding some CPU power or RAM. On the other hand, NoSQL databases are horizontally scalable, and this means that it is possible to increase working traffic by sharding (adding more servers to DBMS). In the second case, the system can become much larger and more powerful with minimum effort. In relational DBMS, data is presented in the form of tables, while in non-relational - in the form of documents, key-value pairs, graphs, or wide-column storages.

In general, there are no databases that are suitable for absolutely everyone. That is why many projects use both relational and non-relational databases to solve various tasks. Although NoSQL databases have become popular due to their speed and scalability, SQL repositories may be preferable in some situations. One of the reasons would be the need for the database to meet ACID requirements (Atomicity, Consistency, Isolation, Durability). This approach reduces the probability of unexpected system behaviour and ensures database integrity. It is achieved by rigorously determining exactly how transactions interact with the database. It differs from the approach used in NoSQL databases, which focus on flexibility and speed, rather than 100% data integrity.

If there is a suspicion that the database may become the bottleneck of a specific project because of large amounts of data, it is worth looking in the direction of NoSQL databases that allow something that relational databases do not know how to do. Again, storing large volumes of unstructured information is a crucial feature of Non-relational databases. Also, fast development is a substantial property of NoSQL databases. If it is needed to develop a system using agile methods, using a relational database can slow down the work. In order to model, configure, and deploy a NoSQL database, less amount of time and operations are needed in comparison with relational databases. Finally, the excellent scalability of non-relational databases facilitates their integrations in cloud storage. Using, for testing and development, local equipment, and then transferring the system to the cloud, where it works, is for what NoSQL databases are created.

Some examples of SQL databases include MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. NoSQL database examples include MongoDB, BigTable, Redis, RavenDB, Cassandra, HBase, Neo4j and CouchDB.

In the end, we could conclude that in the modern world, there is no confrontation between relational and non-relational databases. Instead, it's worth talking about their joint use to solve problems in which a particular technology performs best. Also, in the last years, the integration of these technologies into each other is increased. To develop our platform, however, a choice is necessary. Taking as a basis the requirements of the project, namely the heterogeneity of data in the case of entities, it is evident that using NoSQL databases is the only viable solution. But again, the question arises, from non-relational DBs what type of data storage is preferable for our project. Taking into account the specific characteristics of the researched objects and the operations to be performed on the data, most appropriate would be document-based databases.

And, of course, MongoDB, as the flagman of document-based, non-relational databases, is the solution chosen within the project.

MongoDB [11] is one of the next generation NoSQL-type databases disputing the dominance of relational DBMS (Oracle, Microsoft). It is used now in most corporate data centres and by some big players on the IT market, such as Craigslist, eBay, SourceForge, Viacom, and many others. It is an open-source project that does not require a description of the table schema. As well MongoDB is positioned as a High-performance, scalable DBMS that occupies a niche between fast and scalable systems that operate on key/value format and relational DBMSs, functional and convenient when using queries. MongoDB supports storing documents in a JSON-like format, has a sufficiently flexible language for generating queries, can create indexes for various stored attributes, effectively provides storage of large binary objects, supports the logging of INSERT/UPDATE operations to the database, and can work following the paradigm Map / Reduce, supports replication and building fault-tolerant configurations.

MongoDB conceptually is the same as the usual, familiar DMBS. Inside MongoDB can be zero or more databases, each of which is a container for other entities. The database may have zero or more “collections.” The collection is so similar to the traditional “table” that can safely be considered the same. Collections consist of zero or more “documents.” Again, a document can be considered as a "row." A document consists of one or more “fields” that are similar to “columns.” “Indexes” in MongoDB are almost identical to those in relational databases. “Cursors” are an essential concept in MongoDB. When it is requested any data from MongoDB, it returns a cursor (which can be counted or can skip a certain number of records), without loading the data itself. The general structure of MongoDB collection is illustrated in Fig. 3.2.

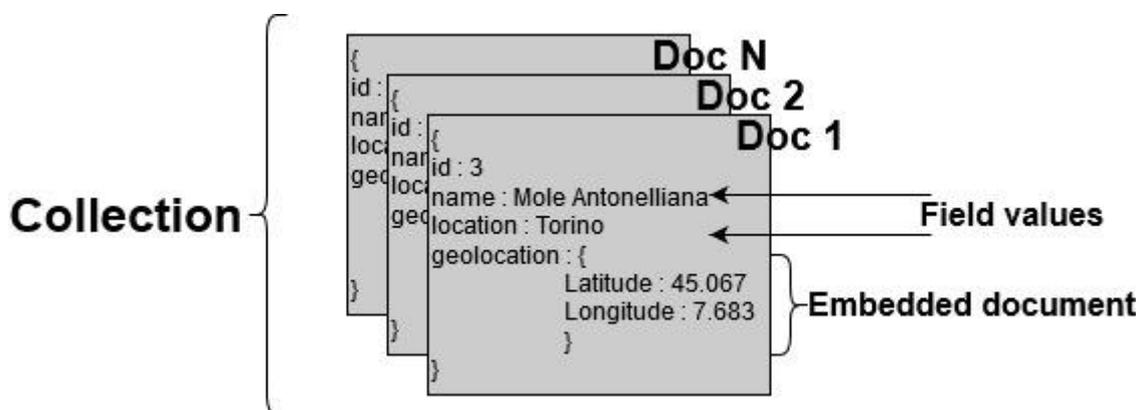


Fig. 3.2 General structure of MongoDB collection

An interesting aspect of databases in general, and MongoDB in particular, is indexing. It supports efficient query execution. When searching for documents in small collections, usually, there are not experienced any particular problems. However, when collections contain millions of documents, and the selection should be done by a certain field, then the search may take some time, which may turn out to be critical for the task. Without indexes, MongoDB needs to scan each collection document to select those documents that match the query. So, indexes are special data structures that store small pieces of data in an easily recognizable form. They contain the ordered values of a particular field or set of fields specified in the index.

As MongoDB includes query operations with geospatial coordinates as GeoJSON objects, it also offers specialized, geospatial indexes to facilitate these operations. There are 2 types of geospatial indexes: 2d - which supports queries that calculate their values on a two-dimensional plane and 2dsphere - which supports queries that calculate their data on an earth-like sphere. They support the determination of the geo-position of one object in relation to another (their intersection, the inclusion of the boundaries of one object in another, the determination of the vicinity with the indication of the minimum distance between objects).

Since MongoDB was also designed to support high-performance queries, it is significant to mention the aggregation mechanism and pipeline concept, because it was used in the current project. Aggregation is a grouping of the values of many documents. In MongoDB, it is done using the aggregate() method. Aggregate operations allow the manipulations of such grouped data. Tab. 3.1 illustrates all aggregate operations.

Operation	Description
<b>\$sum</b>	Summarizes the specified values of all documents in the collection.
<b>\$avg</b>	Calculates the average value of the specified field for all documents in the collection.
<b>\$min</b>	Gets the minimum value of the specified document field in the collection
<b>\$max</b>	Gets the maximum value of the specified document field in the collection
<b>\$push</b>	Inserts a value into an array in the resulting document.
<b>\$addToSet</b>	Inserts a value into an array in the resulting document but does not create duplicates.
<b>\$first</b>	Gets the first document from the grouped ones. Commonly used with sorting.
<b>\$last</b>	Gets the last document from the grouped ones. Commonly used with sorting.

Tab. 3.1 Overview on aggregate operations

On UNIX systems, the pipeline concept means that the operations can be performed on some input and use the output as input for the next command. MongoDB also supports this concept. In Tab. 3.2 is a list of possible stages, and each of them can have as a set of documents for input and generate a set of documents obtained as a result of processing. The general schema of aggregation pipeline are illustrated into Fig. 3.3.

Operation	Description
<b>\$project</b>	Used to select some particular fields from the collection.
<b>\$match</b>	A filtering operation that can reduce the number of documents that are submitted for input to the next step.
<b>\$group</b>	The aggregation itself.
<b>\$sort</b>	Sorts the documents.
<b>\$skip</b>	Ignores the list of documents in the existing set.
<b>\$limit</b>	Limits the number of documents to be output to the amount passed to the method, starting from the current position.
<b>\$unwind</b>	Used to deconstruct documents that use arrays.

Tab. 3.2 The possible stages of aggregation pipeline

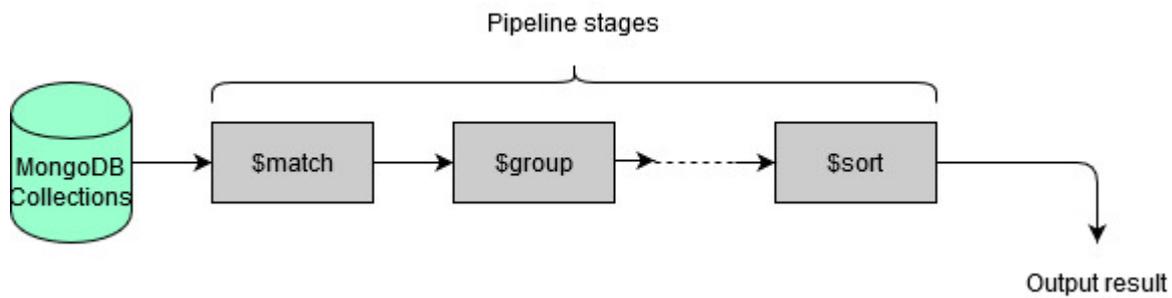


Fig. 3.3 Aggregation pipeline

A real example of using an aggregate pipeline can be found in Chapter 5 of the present thesis.

MongoDB uses so called “dynamic scheme” concept. It permits a flexible approach with the data scheme without the need to change the existing data itself. Also, MongoDB is horizontally scalable, which makes it easy to reduce the load on the server with large amounts of data, and it is flexible as it allows adding fields or columns without harming existing data, their structure, and database performance. Also, it is designed to take into account high performance for simple queries. Due to its simplicity, it doesn't require DB Administrator and can easily be used by any actor in software development and management processes. And last but not least, it is free to use.

Also, MongoDB has facilities for providing sharding - distribution of a data set among servers based on a specific key. Combining it with replicating data, a horizontally scalable storage cluster can be built. This type of "construction" ensures no single point of failure; automatic recovery after failure; transfer of load from the failed node. Expanding a cluster or converting one server to a cluster is performed without stopping the database operation by simply adding new machines.

### 3.3 Frontend development framework

In our time, with the advancement of IT technologies and the increase in the number of complex web projects, there have been significant changes in the methodology and processes of web programming. First of all, the exact roles for developers and technologies covering different aspects of the project were defined. The entire process was divided into frontend and backend development. And in modern companies, for large projects, rarely it can be meet a development team which is responsible for both aspects at the same time.

Let's try to define more briefly frontend and backend notions. The frontend is all that a browser can read, display, and run. Backend is everything that works on the server. For the programmer, frontend is all HTML, CSS, and JavaScript code, and for backend, any available tools on the server can be used. It means that any universal programming language: Ruby, PHP, Python, Java, or a database management system such as MySQL, PostgreSQL, MongoDB can be used.

In this paragraph, we are focused on the frontend. From the beginning of web programming, all user interfaces were designed and developed manually using raw HTML and CSS. Later, coding in raw JavaScript was the most obvious, but at the same time, not the easiest method to

give dynamism to web pages. Nowadays exists a bunch of frontend frameworks, behind which are already preconfigured, a lot of styles and scripts for every component of the web page, and the developer should only choose the right one to obtain a pleasant and well-functioning user interface. The frameworks are designed to simplify the developer's lives and free them from writing the same type of code. But, as the code base of some frameworks grows significantly, they begin to add their share of complexity to the projects.

In order to facilitate the elaboration of the frontend part of our project, it is necessary to analyze and compare all the possible solutions among the existing frameworks and stop at one. We focused our attention on the following frontend frameworks: Bootstrap, Vue.js, React.js, Angular. From the beginning, it is worth mentioning that we chose Bootstrap. However, until its detailed description, we need to specify the characteristics of each of the frameworks briefly. And of course, it is necessary to state the reasons why they were not chosen.

Let's start with Vue.js [12]. It is a progressive JavaScript framework for creating user interfaces. Unlike monolith frameworks, Vue is made suitable for gradual implementation. Its core primarily solves problems of the view level, which simplifies integration with other libraries and existing projects. Vue.js provides extensive functionality for the presentation layer and can be used to create powerful single-page web applications thanks to its core components, such as Router and Vuex. Vue is suitable for small projects that need to add a bit of reactivity, submit a form using AJAX, display values when entering user data, authorization, or other similar tasks. It also allows us to create component templates, bind data to the DOM structure with conditions and loops, and declarative rendering, with simple syntax based on the specification of web components. On the other hand, Vue is easy to scale and well suited for large projects, so why it is called a progressive framework.

In conclusion, it should be noted that the main advantages of Vue JS are its simplicity and ease of study. Instead of learning complex terminology and tools to create a simple application, as is the case with React (short description of it are below), it is possible to start the development immediately. This fact makes Vue JS the perfect choice for startups or any development team that wants to build high-performance web applications with easy-to-read code quickly.

React.js [13] is a tool for creating user interfaces. Its main task is to ensure that what can be seen on web pages is displayed on the screen. The working principle of React consists of breaking each page into small fragments, which are called components. React has no controllers, views, models, templates - everything is a component. Components can and should be reused, inherited from each other, assembled. In fact, each component is a JavaScript function that returns a piece of code representing a fragment of a page. In order to form a page, these functions should be called in a specific order. Then we collect the results of calls together and show them to the user.

A significant feature of React is that absolutely everything should be written in JavaScript. And it is not only about HTML structures, but recent trends also include CSS control inside JavaScript. This approach has its advantages, but also makes compromises that may seem ineffective for each developer.

Angular [14] is a client application framework developed by Google. Firstly, it was created to facilitate the development of the SPA (Single Page Application) solutions. In this regard, Angular is the descendant of another framework - AngularJS. At the same time, Angular is not a new version of AngularJS, but a fundamentally new framework. One of the main functions of Angular is the ability to develop for several platforms: web, mobile devices, and native desktop (AngularJS did not have mobile support by default). One of the key features of Angular

is that it uses TypeScript as the programming language. As TypeScript relies on JavaScript, it is not needed to learn an entirely new language, and many functions like static typing, interfaces, classes, namespaces, decorators are also available. Angular contains not only tools but also design templates for creating a serviced project. If the Angular application is created correctly, it is not possible to have any confusion in classes and methods that, in principle, are hard to edit and even harder to test. The code is conveniently structured and quickly understandable. But Angular is a fairly large and complex framework with its own philosophy. It isn't very easy for beginners because of the large number of framework concepts to learn, and it requires to know a lot of additional technologies. It is worth mentioning that sometimes using Angular for an application is excessive. If the project is small or medium without a sophisticated UI and interaction, it might be better to draw attention to other frameworks. Therefore, it is very important to evaluate all the requirements, functions of the new application, and deadlines before deciding on the use of this JS framework.

Regarding the reasons why Bootstrap was chosen instead of the frameworks mentioned above, only two words can be said: simplicity and style. In general, the purpose of our project is to develop a lightweight and fast platform that fully covers the requirements regarding its functional part. Mostly, the advantages of the above-described frameworks are the dynamicity and reactivity of the web pages. These aspects greatly complicate the code, as well as affect the work speed of the site. Bootstrap, however, offers the full range of tools needed to develop a modern, functional, and well-designed interface without spending much time on configuration, development, and testing. In other words, the most straightforward framework was chosen in order to focus attention on the most critical part of the project – backend functionalities.

	<b>Bootstrap</b>	<b>Vue.js</b>	<b>React</b>	<b>Angular</b>
<b>Programming language</b>	JavaScript	JavaScript	JavaScript	TypeScript
<b>Library size</b>	187 KB	58.8KB	133KB	143KB
<b>Coding speed</b>	Fast	Fast	Normal	Slow
<b>Complexity</b>	Low	Low	Medium	High
<b>Easy to Learn</b>	Easy	Easy	Medium	Difficult because of TypeScript
<b>Community</b>	Huge	Big	Huge	Huge
<b>Good documentation</b>	Yes	Yes	Yes	Yes
<b>Ideal for</b>	Fast developing of apps with adaptive layout	Small SPA apps	Interactive and fast apps with high flexibility	Large-scale apps with complex business logic
<b>Google trend (Search results - last 12 months)</b>	56	27	86	63

Tab. 3.3 Main characteristics of the compared frontend frameworks

In Tab 3.3, one more time are briefly illustrated the main characteristics of the compared frontend frameworks. A more detailed description of the Bootstrap framework is reported below.

Bootstrap [15] is an open and free HTML, CSS, and JS framework that is used by web developers to make responsive website designs and web applications quickly. It consists of pre-recorded styles and scripts for the use of which it is necessary only to choose the right style classes and attributes to the HTML elements.

The Bootstrap framework is used around the world, not only by independent developers but sometimes by entire companies. The main area of its application is in the development of frontend sites and admin interfaces. Among similar systems (Foundation, UIKit, Semantic UI, InK, and others), the Bootstrap framework is the most popular.

In general, CSS frameworks are built to make more accessible the site creation process for other web developers. The fact is that if a web developer is going to develop a site from scratch, he needs to take care of many things. All CSS styles, all web scripts, need to be written from scratch, and as a result, it counts hundreds and thousands of lines of code to the project. Also, many mistakes in layout can be made (for example, the resulted template looks different in the main browsers, or it is not adaptive). To ensure an adaptive layout to the site, the web developer has to deal with much more complicated tasks. He needs to make sure that the site displays well on all screen resolutions. He has to use media queries for this. Large templates may require a lot of such queries; in addition, it still has to be learned how to write them. In general, when developing an adaptive template from scratch, web developers have to work hard, while his layout skills should be quite high.

In general, just for the adaptive layout, it is worth using Bootstrap. It significantly simplifies the layout. Firstly, the framework takes care of cross-browser compatibility and adaptability, and these are the main things that the developer must consider. But with Bootstrap, implementing them is very simple. It allows creating an HTML template even by a person who has previously worked very little with layouts and is not particularly familiar with CSS. Secondly, the framework is ideal for teamwork. The layout creation on Bootstrap with the proper skill and understanding takes 3-5 times faster, and the uniformity of the code allows any member of the team to make corrections. If a layout without a framework is taken into consideration, then each developer can have his style, and another person has to spend time studying his code.

Overall, Bootstrap consists of grids, classes for styling text, images, tables, and other content, components designed to create buttons, various forms on the page, horizontal and vertical navigation menus, sliders, drop-down lists, modal windows, tooltips, and other interface elements, and classes for solving auxiliary problems most often faced by web developers (aligning text, hiding or displaying an element, setting color and background for an element, setting margin and padding indentation, and much more).

As an advantage of the framework, it's worth mentioning a vast community, the availability of proper documentation and a large number of articles, recipes, and videos; all this, if desired, helps not only to understand the framework well but also to find answers to almost any questions. Within this community, members recreated the design of framework components and created new Bootstrap templates. Such templates could be used for creating sites, making only small changes on the right template.

Other advantages of the Bootstrap framework when developing the frontend part of sites and admin interfaces on its basis, as mentioned before, is the high speed of creating high-quality adaptive layout even for novice web developers (by using of ready components created by

professionals). Besides, cross-browser and cross-platform (correct display and operation of the site in all browsers and operating systems supported by this framework) and well-designed and consistent components (in Bootstrap, all components are in the same style) can be considered as advantages. Also, the ability to customize components for a project is an essential feature of the framework. It is achieved by changing CSS variables and using Bootstrap mixins (changing the number of columns, colour, the radius of rounding of corners of elements, space between columns, and much more). To work with the framework is not necessary to have in-depth knowledge of HTML, CSS, JavaScript, and jQuery (it is enough to know only the basics of the above technologies).

In fact, Bootstrap doesn't have only positive characteristics. As negative ones, two of them can be considered. The first one is that the framework libraries usually contain more code than if web developers should write from scratch during development. It contains everything for all occasions, even what, in a standard project, to cover necessary functionalities, only a small part of scripts is useful. The possibility to choose components of the framework to be loaded into the CSS file solves this problem quickly. For example, the developer can generally download only the grid, and do the rest himself.

The second drawback is the template design. Visiting different sites can be observed the similar Bootstrap buttons there (or other framework elements). But this problem can be easily solved by customizing the components of the framework during the developing process.

## **3.4 Backend development framework**

In the contemporary era, with the increase in the number of large-scale IT projects, the notion of the architectural pattern was introduced. It is a description of the typical organization of the system. In other words, it can be considered as a global range of methods and practices, tested and verified on various systems and in different environments. Over time, numerous patterns were developed to complete the tasks put in front of programmers and system designers. But in the web development, the most popular was and remained MVC [16], which stands for Model-View-Controller. It is a way of organizing code that involves the separation of the application data model, user interface, and the logic of user interaction with the system in three components. So, modifying one of these components has minimal or no effect on the others. A brief description of MVC components is presented in Tab.3.4. In Fig. 3.4 is illustrated a schematic structure of MVC and all interactions between pattern components. The user sends a request to the controller (in the case of web applications, this is an address). The controller processes the request, requests data from the corresponding model, receives data, maybe performs some additional processing, for example, aggregates them with other data, and then transfers the data to the view. The view generates data following the specified display template and returns the result to the user.

But why in general we should use in practice MVC principles? The answer to this question is not obvious because each developer or project manager chooses the structure and method of data transmission and processing by himself. For other programmers to understand this structure (and not only it) is better to use generally accepted "standards." So, the work becomes more productive, and the accumulated knowledge of these "standards" allows quickly include them in the development process if necessary.

The most evident benefit we get from using the MVC concept is a clear separation of presentation logic (user interface) and application logic. Support for various types of users who

use different types of devices is a common problem today. The provided interface should be different if the request comes from a personal computer or a mobile phone. The model returns the same data, but the only difference is that the controller selects different types for data output. Also, the MVC concept significantly reduces the complexity of large applications. The code is much more structured, and thus the support, testing, and reuse of solutions are facilitated.

	Description
<b>Model</b>	It provides an object model that contains data and methods of working with this data, responds to requests from the controller, returning data and/or changing its state. Moreover, the model does not contain information about the methods of data visualization or formats of their presentation and also does not interact with the user directly.
<b>View</b>	It is responsible for displaying information (visualization). The same data may be presented in different ways and different formats. For example, a collection of objects using different views can be represented at the user interface level both in tabular form and in a list; at the API level, it can be exported in either JSON or XML.
<b>Controller</b>	It provides communication between the user and the system, uses the model and view to implement the necessary response to user actions. As a rule, at the controller level, the received data is filtered and authorized - the user's rights to perform actions or receive information are checked.

Tab. 3.4 MVC components description

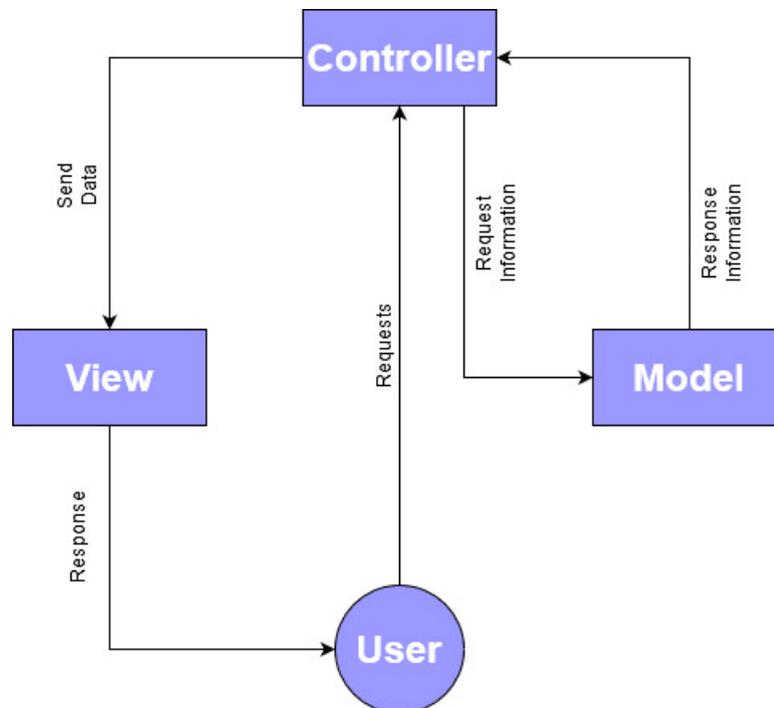


Fig. 3.4 MVC structure and interactions between components

Because the MVC concept has proven to be the ideal solution for developing web applications with a very high level of complexity, it has not been long before the backend

frameworks have been designed, having as the main principle the MVC pattern. In the modern struggle between MVC frameworks to implement the most efficient and high-quality web products, two leaders can be found: Django and Ruby on Rails.

Let's start with the description of Ruby on Rails [17], the not chosen one. Ruby on Rails is a framework written in Ruby programming language for implementing the Model-View-Controller scheme, which allows developers to structure the code and create applications of any complexity. Ruby is an interpreted, fully object-oriented programming language with explicit dynamic typing. It combines a Perl-like syntax with an object-oriented approach. Also, some features are borrowed from the programming languages Python, Lisp, Dylan, and CLU. The only problem with using Ruby was that it was not designed for the web until Rails (the short name of the Ruby on Rails in the developer's community) appeared.

Today, programmers opt for Ruby on Rails because, firstly, the Ruby language itself is so clear, simple, and logical that it is often advised to beginners. Secondly, is the speed of application creation. Thirdly, this software is in the public domain, and, as a result, there are many communities where it is possible to find a standard solution or to get informational support. The other important feature is the built-in testing solutions, which allows quickly to detect errors. Internal testing can be complex as well as modular or functional. At the same time, the framework is very flexible and allows to adapt applications to various updates, in particular, related to the client's desire to upgrade or change it. But this flexibility can often be a disadvantage as the multitude of ways in which a goal can be reached can lead to wasting time to understand how the code works. Also, the same cause can lead to difficulties when transferring the project to other teams. It is worth mentioning the limitations of the Ruby language regarding new trends in the IT-sphere about data analysis and big data functionalities.

But still the question of why Django and not Rails remains. Ruby on Rails and Django have a lot in common (Django detailed description is below). MVC pattern, open-source, objected-oriented languages are only a few shared features. Python, Django REST framework, build-in security and user authentication tools are only a small part of particularities that characterize Django as the perfect framework to quickly build performant web applications that are easy to maintain. However, it should be mentioned that the competition between these two frameworks only benefits everyone. Significant communities and permanent discussions about the advantages of one over the other only lead to the development of new functionalities, and as a result, they complement each other.

Now let's discuss more in detail about Django [18]. It is a high-level web infrastructure suitable for developing complex sites and web applications, written in the Python programming language. Built by experienced developers, Django takes care of many web development issues, so the developer can focus on writing an application without having to reinvent the wheel. It is a free and open-source platform, has a vast and active community, excellent documentation, and many options for free and paid support. It appeared in 2005, and gradually became one of the best frameworks that have helped and is helping thousands of developers to do this or that job within minutes.

Initially, the development of Django was held to provide more convenient work with news portals; a fact which had a significant impact on the architecture: the framework provides some tools that help efficiently and quickly development of informative websites. For example, the developer does not need to create controllers and pages for the administrative part of the site, Django has a built-in content management application that can be included in any site made on Django, and which can manage multiple sites on one server at once. The administrative application allows the developer to create, modify and delete any content objects on the site, to log all committed actions, and provides an interface for managing users and groups (with the

object-based assignment of rights). Django web framework is used in such large and well-known sites like Instagram, Disqus, Mozilla, The Washington Times, Pinterest, and others.

Django was designed to run under Apache (with the `mod_python` module) and using PostgreSQL as a database. Currently, in addition to PostgreSQL, Django can work with other DBMSs: MySQL (MariaDB), SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere, Oracle or MongoDB. To work with a database, Django uses its own ORM, in which Python classes describe the data model, and it generates the database schema.

As the Django framework is based on Python, it is obvious to say some words about it too. The number of Python programmers has been increasing very much in recent few years. This is evidenced by the StackOverflow study for 2019[19]. It is one of the most used programming languages, and in the opinion of some developers, it is an almost perfect programming language. There are several reasons why it merits this epithet: easy to learn, cleanliness and readability, versatility, coding speed, one pieced design. It is used to create different products, from commercial to scientific. The system is aimed to simplify the logic of the code and increase the productivity of the developer. The Python programming language has a concise syntax, and its libraries provide a large stack of solutions. As a universal language, it is used in web development, creating desktop and mobile applications, programming games, as well as in analytics and machine learning.

Python supports several programming paradigms: structural, object-oriented, functional, imperative, and aspect-oriented. The language contains dynamic typing, automatic memory management, full introspection, an exception handling mechanism, support for multi-threaded computing, and convenient high-level data structures. Python code is organized into functions and classes that can be combined into modules, and they, in turn, can be combined into packages.

Despite all the advantages, the language has its drawbacks. Programs on it are considered one of the slowest. Python is not suitable for tasks requiring a large amount of memory - it is better to solve them with inserts in C or C ++. Another disadvantage is the strong dependence of the language on system libraries, which makes porting to other systems difficult.

Django's architecture basically corresponds to Model-View-Controller (MVC). The controller of the classic MVC model roughly corresponds to the level that in Django is called the View, and the presentation logic of the View (from MVC) is implemented in Django by the Templates level. Because of this, the layered architecture of Django is often called Model-Template-View (MTV).

In Fig.3.5, a schematic view of Django MTV is illustrated. URL Dispatcher (configured in file `urls.py`) determines which resource should process the received request based on the requested address. View (usually in `views.py`) receives the request, processes it, and sends a response to the user. If it is needed to access the model and the database to process the request, then View interacts with them. If caching is enabled, View can check if the requested page is cached and bypassing the next steps, can return the cached version of the page. To conclude, View contains the logic of how to access the models and apply the appropriate template. Model (usually in `models.py`) describes the data used in the application. It knows everything about data: how to access it, how to check it, how to work with it, and how the data is related. Individual classes typically correspond to tables in the database. The template makes decisions regarding the presentation of data: how and what should be displayed on the page or in another type of document.

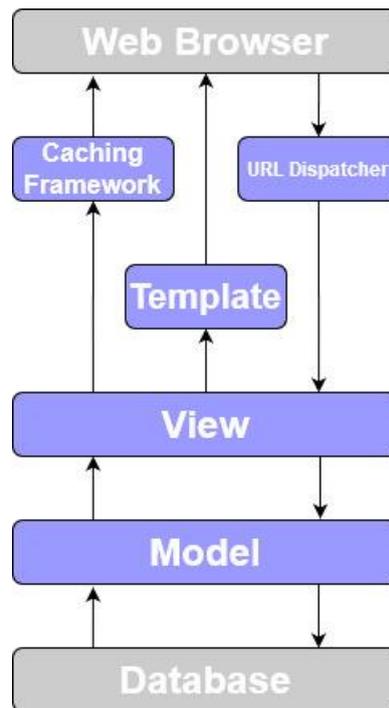


Fig. 3.5 Django Architecture

One of the main principles of the framework is DRY [20] (don't repeat yourself). Web systems on Django consist of one or more applications, which should be alienated and pluggable. This is one of the most significant differences in the architecture of this framework from others (for example, Ruby on Rails). Also, unlike many other frameworks, URL handlers in Django are explicitly configured (using regular expressions), and are not automatically set from the controller structure.

In general, Django has many features with different degrees of usefulness. Its specific ORM - transactional database access API is one of the most powerful features of Django, but at the same time is the most criticized component of the framework because of its untraditional, "pythonic" way to create queries. Also, it is worth mentioning the Django REST framework [21], which is a convenient library for working with REST (Representational State Transfer) web services [22] based on the ideology of the Django framework. Another often-used element is built-in administrator interface, with existing translations into many languages. The possibility of authorization, authentication, and connection to external authentication modules: LDAP, OpenID, and others is a security requirement in modern IT projects and is supported by the framework. Django includes a regex-based URL manager, caching system, and extensible template system with tags and inheritance. Also, it has a filter system ("middleware") for building additional request handlers, such as filters for caching, compression, normalization of URLs, and support for anonymous sessions. In addition, it supports site internationalization and incorporates a plug-in application architecture that can be installed on any Django sites. Also, it has a library for working with forms (inheritance, building forms according to the existing database model) and built-in automatic documentation for template tags and data models, accessible through the administrative application.

Some components of the framework are weakly related to each other so that they can be quite simply replaced with similar ones. But with some (for example, with ORM) of them, this is not easy. In addition to the features built into the core of the framework, some packages exist that expand its capabilities.

Let's describe some advantages of Django (Tab. 3.5), which is not only a quick solution in web development, including everything web developers need for a high-quality code but also an excellent platform for working with the clients of a particular business.

	<b>Description</b>
<b>Speed</b>	Django was designed to develop applications as quickly as possible. During the developing process of the project, Django saves time and resources at each stage of it. Therefore, it is the ideal solution for projects where the deadline issue is a priority.
<b>Full options</b>	Django contains additional features that help a lot with user authentication, site maps, content administration, RSS, and more.
<b>Security</b>	Working in Django, protection against security-related bugs that threaten the project is guaranteed. Some of the common problems like SQL injection, cross-site forgery, clickjacking, and cross-site scripting are avoided. It has its own user authentication system for the efficient use of logins and passwords.
<b>Scalability</b>	The Django framework is best suited for handling the highest traffic, and many busy sites use it to meet traffic requirements.
<b>Versatility</b>	Content management, scientific computing platforms, even large organizations - all this can be effectively managed with Django.

Tab. 3.5 The strengths of Django

Cons always go hand in hand with the pros. Django uses a routing pattern with URL specifications. It is too monolithic and everything it's based on ORM Django. Components deploy together. To be able to work it is needed to know the entire system. That's why Django cannot be called flawless.

Based on Django, a lot of ready-made solutions have been developed, distributed under a free license, including systems for managing online stores, universal content management systems, as well as more specialized projects. They facilitate the process of developing new projects only by customizing some components of a ready-made application. In this way, the time and effort required to develop the project are reduced.



# Chapter 4

## Information System Architecture

Current software applications and information systems have reached such a level of advancement that the term "architecture" as applied to them has not been surprising for a long time. Nowadays, to build an information system competently, efficiently, and reliably operating is not easier than to design and build a modern multi-purpose building. When it comes to "information system architecture," there is usually no shortage of definitions, but they all have a considerable degree of similarity. For example, most definitions indicate that architecture is associated with structure and behavior, as well as only with significant decisions. Also, they emphasize that the information system corresponds to some architectural style; it is influenced by the people interested in it and its environment; it implements decisions based on logical justification. The architecture of a software system covers not only its structural and behavioral aspects but also the rules for its use and integration with other systems, functionality, performance, flexibility, reliability, reusability, completeness, economic and technological limitations, as well as a user interface issue. With the development of software systems, their integration with each other in order to build a single enterprise information space is becoming increasingly important and become an essential element of architecture. To build the correct and reliable architecture and correctly design the integration of software systems is necessary to follow modern standards in these areas strictly. Without this, it is possible to create an architecture that is unable to evolve and meet the growing needs of IT users.

It's worth mentioning the existence of the classification of software systems by their architecture. It includes centralized architecture, "file server" architecture, two-tier "client server" architecture, multi-tier "client server" architecture, distributed system architecture, web application architecture, or service-oriented architecture. It is important to note that, like any classification, it is not rigid. In the architecture of any particular information system, the influence of several standard architectural solutions exists.

As our project is classified as a web application, only this type of architecture raises more interest. Web applications are a particular type of software built on a client-server architecture. Their peculiarity lies in the fact that the Web application itself is located and runs on the server - the client only receives the results of the work. The application is based on receiving requests from the user (client), processing them, and issuing the result. The transfer of requests and the results of their processing takes place via the network (in many cases through the Internet). A browser usually does the display of query results, the data reception from the client, and their transfer to the server. On the server-side, the web application is executed by special software (the Web server), which receives client requests, processes them, generates a response in the form of a page described in HTML, and transfers it to the client. During the processing of a user request, the Web application compiles the response based on the execution of server-side code, a Web form, an HTML page, and other content, including graphic files. As a result, it dynamically generates a response using executable code - the so-called executable part.

In other words, the Information System architecture can be defined as a bunch of models, which, in their interaction, represents the purpose of the system. Each model describes various components and views of the system. Components and views are basic units of the Information System, with the difference that components can act as a system on their own. Usually, five general types of components can be distinguished: Hardware Resources, Software, Databases, Networks, and Human Resources. For particular projects, the creation of new types by mixing or truncating the respective components is also not excluded. But, in the case of our Information System, from a developing point of view, only two of them are of interest: Database System and Software. Other components are more related to the deployment stage of Information System creation.

As our platform is deployed on Docker, it is worth mentioning the container's system inside it. Fig. 4.1 illustrates the diagram that describes the container's structure of the "Atlante dei palazzi comunali" project.

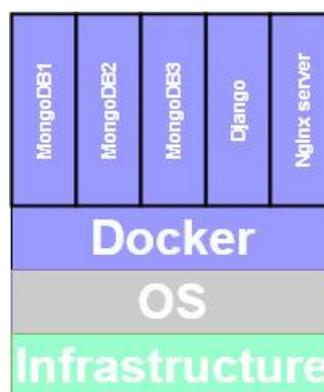


Fig. 4.1 Docker container distribution of our platform

If to take a look at the container level of our Docker configuration, it can be divided into three groups. The first would be MongoDB containers. We create three instances of them in order to be able to use the replication technique. This technique involves special configurations at both container and instance level of MongoDB by joining them in a common group called replica set. It is done to keep the data safe, maintain their integrity, and ensure permanent access to information in case one of the DB servers fails. The second one is the Django container. Here the entire activity related to the project is carried out because, in this container, there are the

tools for developing the application. Django includes the full range of plugins, frameworks, and libraries to develop both frontend and backend parts of our platform. The third one is related to Nginx, which is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. This web server is used to handle the requests for images and static resources directly from the file system.

## 4.1 Database System

Data models are the best way to describe a Database System. They establish how data is stored, extracted, organized in the database, and the relationships and flows between each of the data elements. Data modelling helps accurately implementing the requirements and visual representing of the data. Also, it assures at early stages enforcement of business rules and definition of system policies. Data models guarantee consistency in naming rules, semantics, and security, which in result, increase data quality. Three types of data models are used in practice nowadays. Conceptual Data Model is a high-level view of the system, usually independent from a DMBS. Its primary goals are to define and organize business concepts and rules. Logical Data Model describes information about entities stored in DB and the relationship between them, regardless of the DMBS. The main goals are to define the data structures of the DB and the rules among them. Physical Data Model describes the implementation steps of the DB in specific DMBS. The main goal is the actual implementation of the DB. A more straightforward explanation of these three models is that the business analyst uses a conceptual and logical model to design the data required and generated by the system from a business point of view. At the same time, the database developer exploits the physical model to represent the physical structure of the database, ready for building.

Now it is time to overlap project requirements with data types models described above. The data model can be represented by a graph containing different types of nodes. Some common type features characterize each node. Other features deriving from a specialization defined by subcategories also identifies nodes. In fact, in Fig. 4.2 is represented the conceptual data model, which summarizes the proposed model with the dependencies between the various entities.

To cover the full range of requirements, we decided to define five types of entities: Location (Localita), Building (Palazzo), City Building (Edifici comunali), Source (Fonte), Descriptive Element (Elemento descrittivo). In Fig. 4.2, they are represented by circles of different colors depending on the type, in order to easily distinguish them. Also, regarding the links between entities, it is worth mentioning that the descriptive elements as the main object of study are connected with the sources where they are mentioned and with the building where they are located. Through building, they are also connected with the localities. Last but not least, it should be noted that for a single descriptive element, it is characteristic of being mentioned in several sources.

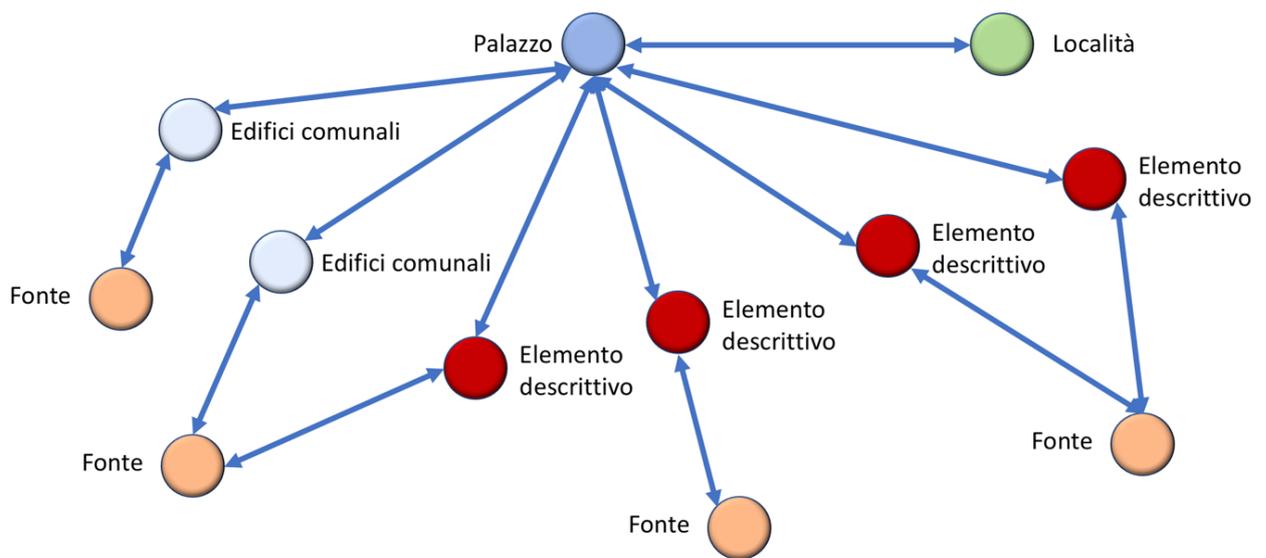


Fig. 4.2 Conceptual Data Model of “Atlante dei palazzi comunali”

Since, for our project, MongoDB was chosen as DMBS, for data modelling, it must be considered several additional particularities. It is all because data has a flexible scheme for storing documents in a single collection. It means that documents should not have the same structure or set of fields. Common fields in collection documents can store various types of data. As mentioned above, for the good usability of the database, at the modelling phase, it is necessary to follow a set of recommendations. Combine objects in one document if they are planned to be used together. Duplicate the data since the computation time is “more expensive” than disk space. Use complex aggregations in a scheme and optimize it for the most common cases.

Using a NoSQL DBMS, Physical Data Model is more representative than others for the description of the Database System. Below, it follows the information about all MongoDB collections one by one. Every document in the collection has its own ID. There are two types of collections inside the database: uniform and heterogeneous.

Uniform collections are characterized by a scheme that doesn't change from one document to another. However, heterogeneous collections don't have a well-defined scheme for every document. In the case of our project, heterogeneous collections have a range of predefined structures, and each document corresponds to one of them. These structures as well are stored in the database in separate collections and they are configured by the users in a special configuration module of the project.

Let's start with **Location** collection, which is a uniform one. In this collection are stored documents about the localities in which the researched buildings are located. Every document has a name, previous names, geolocation shape, coordinates, and first attestation. In this case, the most interesting information from the point of view of representation in DB is geolocation data and previous names. Geolocation information contains two components: the shape of representation and the list of geographical coordinates of each point of the shape. As an example, if the shape is a 4-sided polygon, the list will contain the coordinates of all four vertices. In traditional SQL DBMS's representations of the lists involve creating additional tables and connecting them with the main one, via foreign keys. While MongoDB allows the storage of all information within the same collection, which simplifies querying and minimizes

the execution time. Tab. 4.1 presents a simplified schema of this collection with the necessary conventions that can be used to represent a schema for the NoSQL database.

<b>Location</b>
id : string
Name : string
Previous_names : list of strings
Geolocation_shape : string
Coordinates : list of pairs of floats
First_attestation : int

Tab. 4.1 Location collection schema

In **Building** collection, documents are characterized by a name, old names, geolocation of the building, short review, and location. It keeps the information about researched buildings, architectural, and cultural elements of which are analyzed and mentioned in sources. As can be seen from Tab. 4.2, which describes the building collection scheme, there are similarities with the previous collection. Old\_names and geolocation information have the same meanings and the same implementation as the previous one. The most significant difference is the presence of the Review field, which is a list. Every element of it is composed of three components: language, text, and author. Again, comparing with traditional DBMS, the respective solution permits framing all this information in a single collection, which makes the querying process easier and faster. It is meriting noting that the field Location\_id acts like a FOREIGN KEY in SQL databases, and it is used for linking Building collection with Location. This approach slightly contradicts the NoSQL database principles, which suggest avoiding this type of constructions and duplicate information. It has been chosen because the project involves a relatively small number of researched materials, and the right implementation would require more time and effort than it is worth.

<b>Building</b>
id : string
Name : string
Old_names : list of strings
Geolocation_shape : string
Coordinates : list of pairs of floats
Recognized : boolean
Review : list of objects
Location_id : string

Tab. 4.2 Building collection schema

**City\_building** collection is used for saving information about municipal buildings relevant to the geopolitical context but not subject to detailed research. Its documents have the following features (Tab. 4.3): name, geolocation, building type, and function. It is a simpler version of the previous collection, storing only the strictly necessary information about the neighborhood of the researched places.

<b>City_building</b>
id : string
Name : string
Type : string
Geolocation_shape : string
Coordinates : list of pairs of floats
Function : string

Tab. 4.3 City\_building collection schema

The last uniform collection is **Translation**. It ensures the multilingual aspect of the project by storing translations of platform expressions and messages in all languages defined in the system. In Tab. 4.4 can be seen that it contains the initial message and a dictionary with the translations (a list of key-value pairs, where the key is the language and value is the translation).

<b>Translation</b>
id : string
Message : string
Languages : dictionary

Tab. 4.4 Translation collection schema

Besides this, the system includes two heterogeneous collections and two “scheme” collections, which describes the possible structure of the documents from the heterogeneous collection. Let’s start with scheme collections, and the first one is **Source\_type**. Here it is stored information about all types of sources. As can be seen in Tab. 4.5, every document has a name and a list of inputs. The main characteristics of input are name, type, the order in source form, and if it is required or no. Based on the information stored in this collection, the structure of each document in the **Source** collection is formed, as well as the HTML form of insert/edit sources is designed.

The second schema collection is **Elements\_type**. It has the same structure (Tab. 4.5) as the previous one, with the difference that it records information about the elements collection structures. And respectively, through the information stored here, the HTML form for the insertion/editing of descriptive elements is formed.

<b>Source_type/Elements_type</b>
id : string
Name : string
Elements : list of objects

Tab. 4.5 Source\_type/Elements\_type collection schema

For a better illustration of these specific collections of our platform, it is necessary to bring a concrete example and to explain it in detail. Of course, the most interesting part is related to the list of elements. As it was described above, the list of elements is a list of inputs that are displayed on Source and Descriptive element insertion forms. As they have the same structure, we focus only on one of them, namely **Source\_type**. Tab 4.6 shows a part of the structure of “Published Sources”, which includes the first 3 elements of it and the view of this structure on

the insertion form. It is enough to be able to describe the features and functionality of the chosen approach. The usage of each of the four features of the elements in the list can be easily seen in this table. The name is displayed before each input and identifies it. The type has five possible values, and it defines the type of each input: textbox, text area, number, checkbox, date textbox with validation. The required feature defines the compulsoriness of the form input. The order is used to place the inputs in a user-defined order.

And the last, but not the least, the time has come to describe heterogeneous collections. First of all, there is **Sources** collection, which contains the records about the researched sources with a detailed and customized description of it, using the structures defined in **Sources\_type** collection. On the other hand, **Elements** documents record information about the researched element and link it with the building where it is present and with the source where it is described — the detailed description of the element bases on the structure defined in **Elements\_type** collection. We choose this approach for the designing sources and descriptive elements management, in order to create a customizable product and to allow the users to decide by themselves what type and level of information should be contained in the DB. Of course, this was not possible to do without MongoDB features, and the use of SQL DBMSes, for sure, makes this part of the project more difficult and increases development time and effort.

Example of Source_type	
MongoDB Document	Insertion form of the new “Published Source”
<pre> { ... "name": "Fonti edite", "elements": [ { "name": "Autori", "type": "text", "required": "on", "order": "1" }, { "name": "Titolo", "type": "text", "required": "on", "order": "2" }, { "name": "Tomo", "type": "number", "order": "3" } ] } </pre>	<div style="margin-bottom: 5px;"> <p>Autori*</p> <input type="text" value="Giovani Crosetto"/> </div> <div style="margin-bottom: 5px;"> <p>Titolo*</p> <input type="text" value="Cirie tra sviluppo e guerra 1900 - 1938"/> </div> <div> <p>Tomo</p> <input type="text"/> </div>

Tab. 4.6 Source\_type example

## 4.2 Software infrastructure

In general, a modern application must combine a practical user interface and a healthy functional. In turn, an effective user interface combines two components: a pleasant appearance and ease of use. To create such an interface is crucial to understand the basics: how the user interacts with the UI, what he expects, and what can become a source of potential problems. As our project is not for mass use, and requests in considerable measure refer to the functional part of the platform, UI design was focused on utilitarian project purposes and not on presentation.

Taking into account the above, as a web infrastructure for the project, was chosen the Django framework. It has the right tools in order to reach all desired objectives and together with other frameworks to design a well-looking and straightforward interface with minimal effort. Also, Django allows the simple implementation of all functional requirements and ensures the rapid and qualitative development of applications due to the well-known MVC principles. As the Django has a well-defined architecture, the description of our project follows its structure. As the project has two parts, the private one, which is the most complex, is described more in detail.

Let's start with the bottom of Django, with templates. Each project page has its own Django template. All the templates are designed using the Bootstrap framework to create a unique and convenient style for the platform. As Django permits to use the same blocks of HTML for different parts of the website, the standard part of the templates pulls into a separate file, called `private_base.html`, and each template link with it by adding the tag “extends” in the start of the file. It contains the navigation bar menu (Fig. 4.3).



Fig. 4.3 Navigation bar menu – contained in `private_base.html`

Apart from templates, any page has a JQuery [23] script file. It is used to fill the template with data and add dynamic functionalities to the page (e.g., Hide some buttons at a particular stage, prepare the list of suggestions, define on click events, and much more). The data is retrieved/sent from/to Django APIViews, performing asynchronous HTTP (Ajax) requests. APIViews it's a REST framework implementation in Django. Regular views in Django are used to render each page template. In other words, in our project, Views are used to display templates, APIViews are used to make queries to MongoDB by asynchronous requests.

Another way to define a web application structure can be considered sitemaps [24]. A sitemap is the graphical or list representation of all the web application pages and the links between them. We can distinguish three types of sitemaps: XML, page list, and diagram. Usually, XML sitemaps are used for mapping sites in search engines, to facilitate the search process. List sitemaps are published directly on-site in the form of links to each component of the web application. They are used by users to understand the structure of the site and to navigate directly from it. Diagrams are more used at the design stage of the web application in order to show the global view of it to all actors who are involved in the development process. In Fig. 4.4 are illustrated exactly such a diagram.

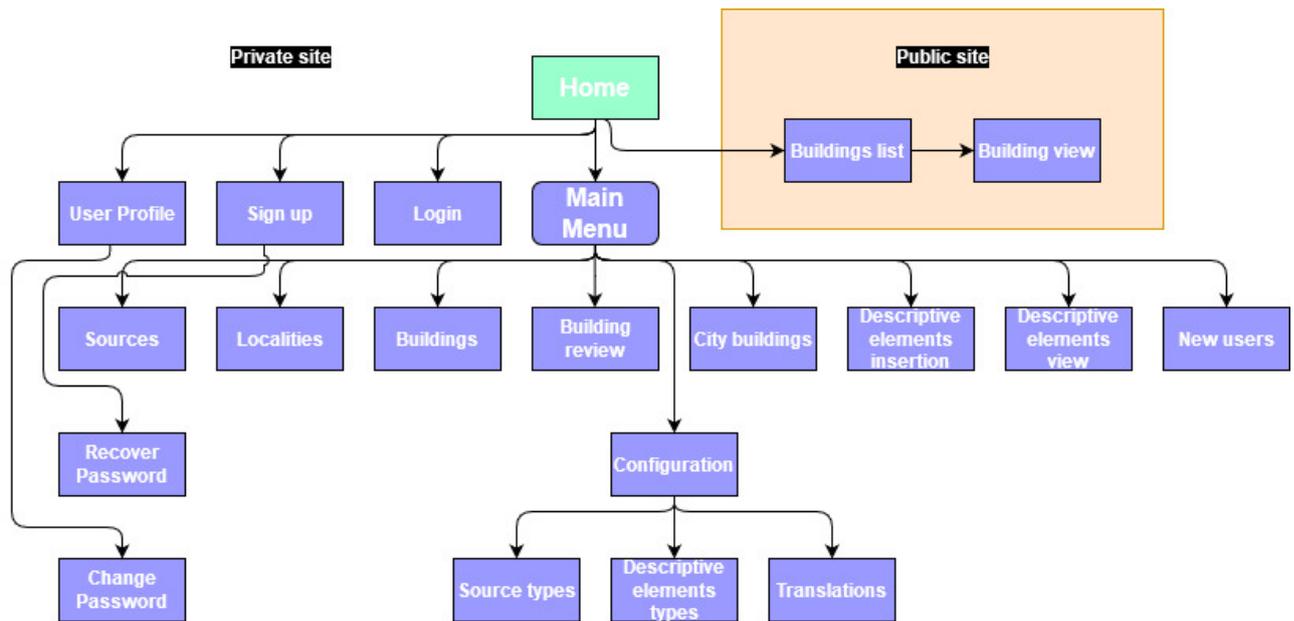


Fig. 4.4 Sitemap of our web application

If we look at the diagram above, some peculiarities can be observed. First of all, it is about delimiting the platform in the private and public parts. Secondly, the presence of the main menu with many options shows that we have in the web application a convenient navigation tool. The presence of the configuration page is also worth mentioning, as it contains modules for configuring several entities.

In conclusion, the elaborated user interface covers all submitted requirements. It permits easy and intuitive navigation through platform elements and provides, even to a novice user, a full range of functionalities without doing any pieces of training. Once more worth to mention the flexibility of the system in general and user interface customizability in particular. Also, the use at this stage of standard tools of the frameworks leaves room for future improvements and modifications.

### 4.3 Platform functionalities

Before jumping to the next chapter, it is time to say some words about “Atlante dei palazzi comunali” functionalities. According to the requirements, the platform should cover a large number of entities. As it was designed and developed for a group of researchers, a part of these entities was left to the discretion of the users. Of course, it is about the possibility of inserting new types of sources and elements with dynamically generated forms. If these forms were hardcoded from the beginning, then there was no possibility of developing the system in the future. And with each new type of discovered source or researched descriptive element, it would be necessary for the intervention of a software developer to insert the changes. But how the project is designed, there are no limits in this direction.

Another feature of the system is the possibility to determine the position and boundaries of the researched localities and buildings directly on the map. This facilitates the insertion of data

and permits users to have a complete view of the researched region.

Worth mentioning the multilingual aspect of the system. This feature was implemented as the group of researches is multinational. The translation mechanism is a configurable subsystem of the platform. It allows users to insert translations of each platform message in every configured language and displaying the page corresponding to the chosen language. When a translation is not inserted, it displays the default version of the message.

The possibility of inserting a detailed and HTML stylized description of the researched building is another interesting functionality. It allows the users to customize the style of the description text in many ways and also keep this style when displaying it.

Inside the project, a customized Django authentication system is implemented. It permits to manage login and logout, change password, recover of forgotten password, and view profile operations by any user in the system. Besides that, only authenticated users can access the private part of the program. Also, together with authentication, authorization is realized in the platform as dividing users into two groups: Admins and Researchers. The configuration page and creation of the new users are accessible only to Admins. Researchers can insert/edit/view all other entities of the project.

In conclusion, the developed platform has a lot of configurable and customizable features. This allows users by themselves to manage and develop the system without the help of a software developer alongside.

# Chapter 5

## Use cases

The purpose of this chapter is to demonstrate with real use case scenarios the functionality of the system and the fulfillment of the requirements. The use case UML diagram can be found in Fig. 5.1. In particular, it illustrates the processes of configuring a new type of source and descriptive element. We can find in it the process of inserting/editing the following entities: source, descriptive element, locality, building, city\_building. Also, the management part of the site that does not refer to the research itself is described: user management and translation process. Last but not least, it shows the structure of the homepage of the private part, which contains a little bit of statistical information. Below is a detailed description of all the mentioned use cases.

But first, because inside the platform, there are some common operations for different pages, and not to specify the requests that go to the Django framework for the same operation at each use case, we describe their general behavior here. It's about saving and retrieving the data to/from the database. A detailed description of database collections is done in Chapter 4. It is worth mentioning that JQuery scripts process all events on-page elements (clicks, change values, and other). After clicking the save button, to store the data in the database, from JQuery script, an asynchronous HTML POST request is sent to the server. The request is reaching the View component of the Django (Model-Template-View architecture is mentioned in Chapter 3), where are described how this POST request should be treated. Usually, the request contains the data, which is needed to be saved, and View matches it with the model of the collection and inserts or replaces it in the MongoDB. It depends on whether a new entity is introduced or an existing one is modified.

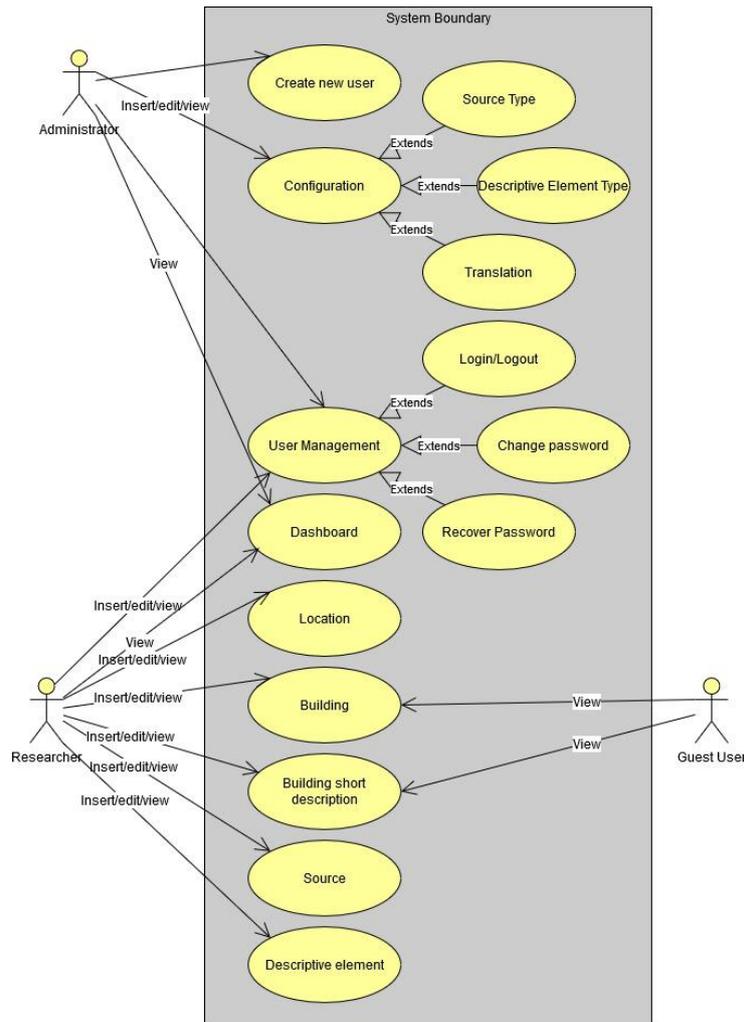


Fig. 5.1 Use Case UML Diagram

Regarding retrieving the information from the database, it is almost the same behavior. But this time, an asynchronous HTML GET request is sent to the Django. In the same way, it reaches View component where are as well described GET requests too. Usually, the request contains a list of parameters to specify the selection of the necessary information. Within the View, depending on those parameters, a query is set, which returns the desired information from the collection. Which later, the JQuery script arranges it on the page.

Below for each webpage is described all possible use cases with the detailed description of the workflow within it. The schematic version of workflows is also presented for the most important and complex use cases, in order to highlight the use case actions.

## 5.1 Configuration

First of all, it is necessary to start with the configuration unit because the subsequent use of other parts of the application depends on it. As the first operation, we need to login to the application in order to enter the private part of the platform. As user management is described

later in this chapter, let's consider a successful login and display of the private site. A schematic view of the configuration workflow is illustrated in Fig 5.2.

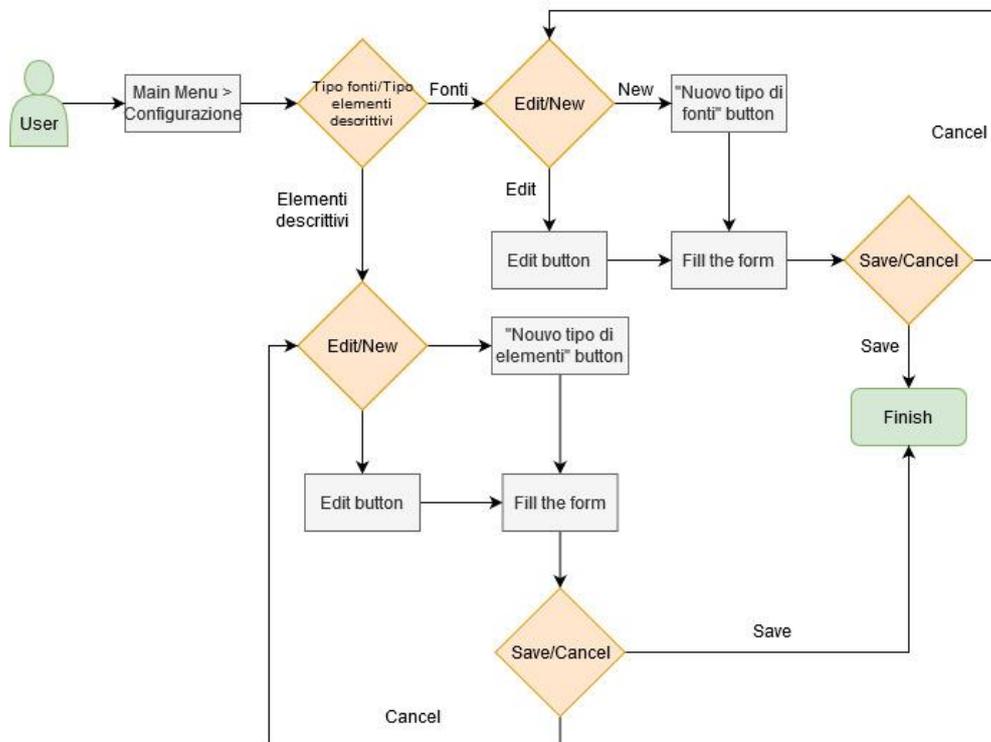


Fig. 5.2 Configuration workflow

In order to enter the configuration page, from the main menu, we select *Configurazione*. Then to set up a new type of source, we select the tab *Tipo fonti* and then press the button *Nuovo tipo di fonti*. Fig. 5.3 illustrates all these actions. Like it can be observed on the right side of the page, it is a blank card. There is a list of already inserted items.



Fig. 5.3 Entering in configuration page for inserting new type of source

Subsequently, the form for the insertion of new source types opens (Fig. 5.4). First of all, it is necessary to put a name for the type of source. Then by pressing the button *Aggiungi elemento form*, the blocks for setting the fields of the source are added one by one. If it is necessary to cancel a block, it can be done by pressing the red cross button. All these behaviors are made possible by JQuery scripts. Each block is composed of four settings: name, input type, a checkbox, which indicates if the field is mandatory or no, and the order in the form of source insertion. The input type should be chosen between five possible values: text, text area, number,

data, and boolean. After compiling the form, we have the option to save the data (*Salva* button) or to press the *Cancella* button and to lose all the inserted information. After saving the data, all the information is stored in the **Source\_type** collection. And of course, the block of the page, which contains the information about already inserted entities are populated with new information (Fig. 5.5). In case we want to make some modifications to an already inserted entity, this can be done by pressing the *Modifica* button. In this case, the JQuery script sends in GET request the id of the source\_type as the parameter. As a result, it receives a response, all the stored data about it and populates the form (from Fig. 5.4). And after making the necessary changes, they can be saved or cancelled. This is general behaviour for all *Modifica* buttons of the platform. As a particularity, each one sends the id of the entity that wants to modify.

The screenshot shows a form titled "Tipo Fonte" with a header bar containing "Salva" (green) and "Cancella" (red) buttons. The form contains several input fields: "Nome" (with "Fonti edite" entered), "Autori" (with "Autori" entered), "Tipo" (with "text" selected in a dropdown), "Obbligatorio" (a green toggle switch), and "Ordine" (with "1" entered). A blue "Aggiungi elemento form" button is located to the right of the "Nome" field. Below the main form, there is a section with a "Nome" field containing "Titolo" and a red "X" delete button.

Fig. 5.4 Inserting new type of source form

The screenshot shows a table titled "Lista tipologia fonti". The table has two columns: "Nome" and "Modifica". The rows are:

Nome	Modifica
Fonti edite	
Pubblicazioni scientifiche	
Fonti archivistiche	

Fig. 5.5 List of already inserted Source types

The process of inserting/editing of descriptive elements types is identical to that of the source types except that all the actions are done under the tab *Tipo elementi descrittivi* on the configuration page. All the inserted documents are saved in the `Elements_type` collection.

## 5.2 Insert/edit of a new source

After we have configured the source types, we can proceed to the insertion of a new source that will be linked later with the researched descriptive element and the building in which this element is located. Firstly, in Fig 5.6 the schematic workflow for managing sources is illustrated. To deal with the sources in the main menu is necessary to access *Fonti*. Before proceeding to the insertion, it is necessary to select the type of source with which we want to work. It is worth mentioning that the previous paragraph of the chapter covered these source types. It is interesting that in order to complete the list of source types, the JQuery script performs the retrieval of the information from the DB (general operation described at the beginning of the chapter). All previously described actions are illustrated in Fig. 5.7.

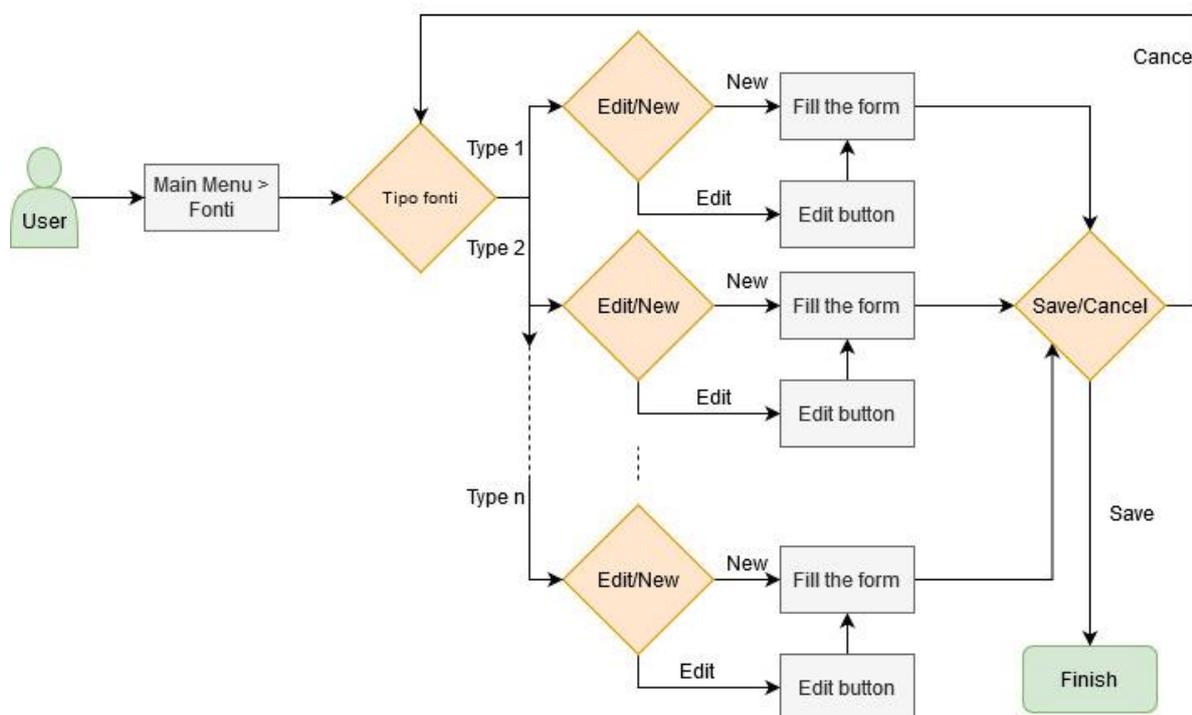


Fig. 5.6 Sources management workflow

After selecting the source type, the displayed page acquires a familiar outline: on the left side, the insertion form, and on the right side, the list of already inserted entities. It is important to draw attention that the components of the form fully correspond to the configurations of the source type covered in the previous paragraph of the chapter (Fig. 5.8). The JQuery script performs this page layout. Firstly, it retrieves from DB configuration data about the fields of the early selected type of source. Then with the returned result, it draws the form on the page as it is described below. Each component of the form has a label that corresponds to the name

from the configuration and an input, type of which is taken from configuration too. Also, the fact that the component is mandatory or not is extracted from configuration info. And lastly, the order of the components in the form corresponds strictly to the order get from the configuration.

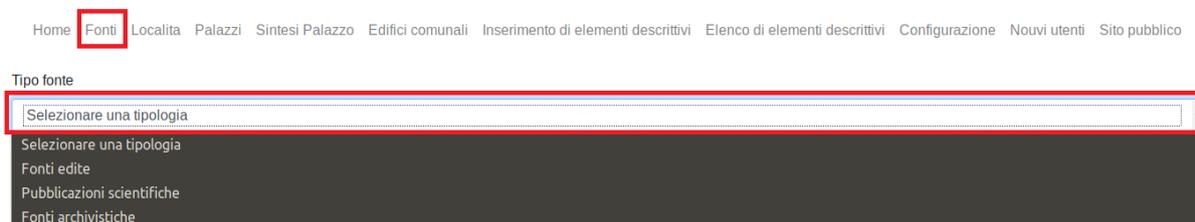


Fig. 5.7 Entering in source page

Autori\*

Titolo\*

Tomo

Numero documento, tavola o figura

Pagine in cui si trova l'informazione

Nota

Data del contenuto (gg/mm/aaaa)

Anno di pubblicazione

Fig. 5.8 Inserting new source form

After filling the form with the desired information, it can be saved by clicking *the Salva* button located at the bottom of the form (Fig. 5.9). It is stored in the **Sources** collection of our DB. Of course, this action also refreshes the list of already introduced sources (Fig. 5.10). After saving, the platform proposes the user to jump directly to the descriptive element configuration page, with the source field already completed with the one which has only been saved.

Annotazioni

Salva
Aggiungi nuovo elemento

Fig. 5.9 Form management buttons

Lista fonti

Show  entries Search:

Autori	Titolo	Modifica
Giovani Crosetto	Cirie tra sviluppo e guerra 1900 - 1938	
Mario Minardi, Ermanno Franchetto	Il Canavese ieri e oggi	

Showing 1 to 2 of 2 entries

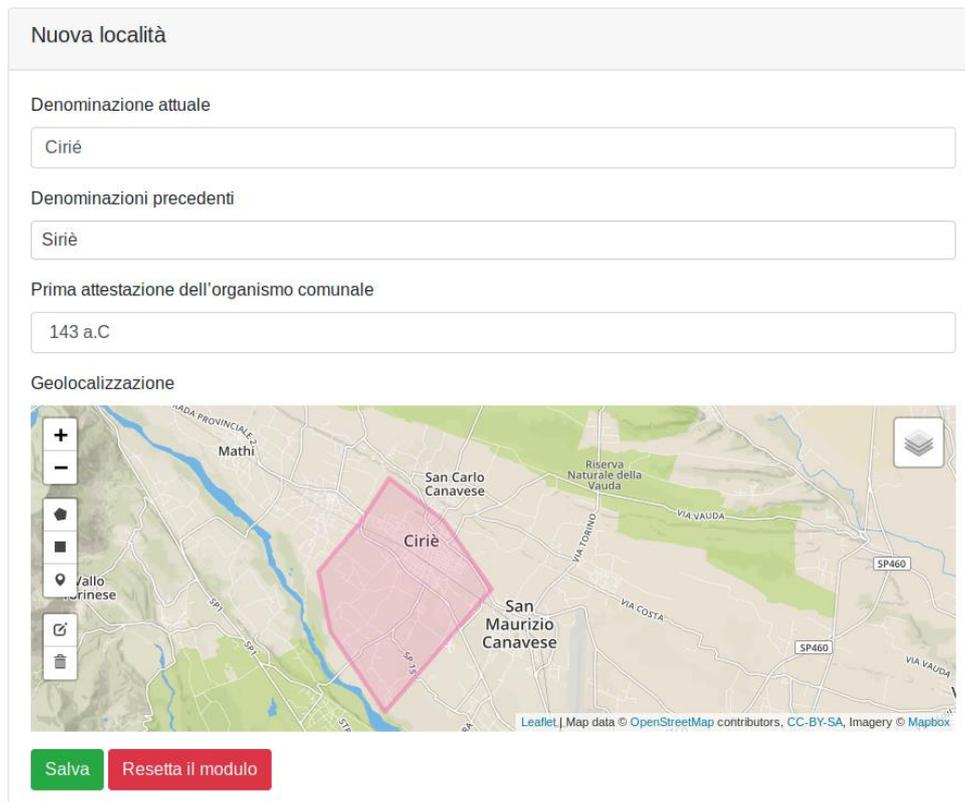
Fig. 5.10 List of introduced sources

The functionalities of the *Modifica* button described in the previous paragraph are also identical for this page too. However, from the edit mode of this form, it is a possibility to jump directly to the descriptive element configuration page, with the source field already completed with the value of the current one by clicking *Aggiungi nuovo elemento* button (Fig. 5.9).

### 5.3 Insert/edit of a locality

Before inserting the descriptive elements, it is necessary to configure other types of entities that are used there. So, let's start with the localities. As usual, to enter the management page of some entity, we need to access the main menu. To maintain localities, we should click *the Localita* option of the menu. In this case, immediately appears the well-known template with insertion form on the left and the table of inserted localities on the right side of the page. The form is designed to gather information about the name, old names, and the year of the first attestation of the locality. A special mention deserves the field of old names; it has attached the *amsify-suggesttags* plugin. It is a simple jQuery tag/token input plugin that converts the regular input into a multi-select, auto-suggesting tagging system. Besides standard text inputs described above, the current insertion form brings to our attention a new type of input: a new interactive map that is used to locate the exact borders of the locality (Fig. 5.11). It is implemented using the Leaflet JavaScript library. In general, it has many features, but in the

case of the described page, we use zoom in/out and the possibility to add the location on the map. It supports different shapes, but we left only rectangle, polygon, and point. It also had the possibility to modify the already drawn shape or to delete it.



Nuova località

Denominazione attuale  
Cirié

Denominazioni precedenti  
Sirié

Prima attestazione dell'organismo comunale  
143 a.C

Geolocalizzazione

Salva Resetta il modulo

Fig. 5.11 Form for inserting new locality

After saving the data (click on *the Salva* button), it is evident that everything is saved in the DB in **Location** collection. It is interesting to mention that in case the drawn locality has the polygon or rectangle shapes, it saves the coordinates of all vertices as a list. The table (on the right side of the page) containing the list of localities already entered is refreshed (Fig. 5.12). The form also permits to reset all its inputs by clicking *Resseta il modulo* button. The list of already inserted localities has the same functionalities as other pages described before. *Modifica* button allows doing the same edit actions on inserted localities. It is worth mentioning the navigation options of the list, like pagination or search input. This approach improves visual effect on the page and, of course, makes it easier to navigate through the list.

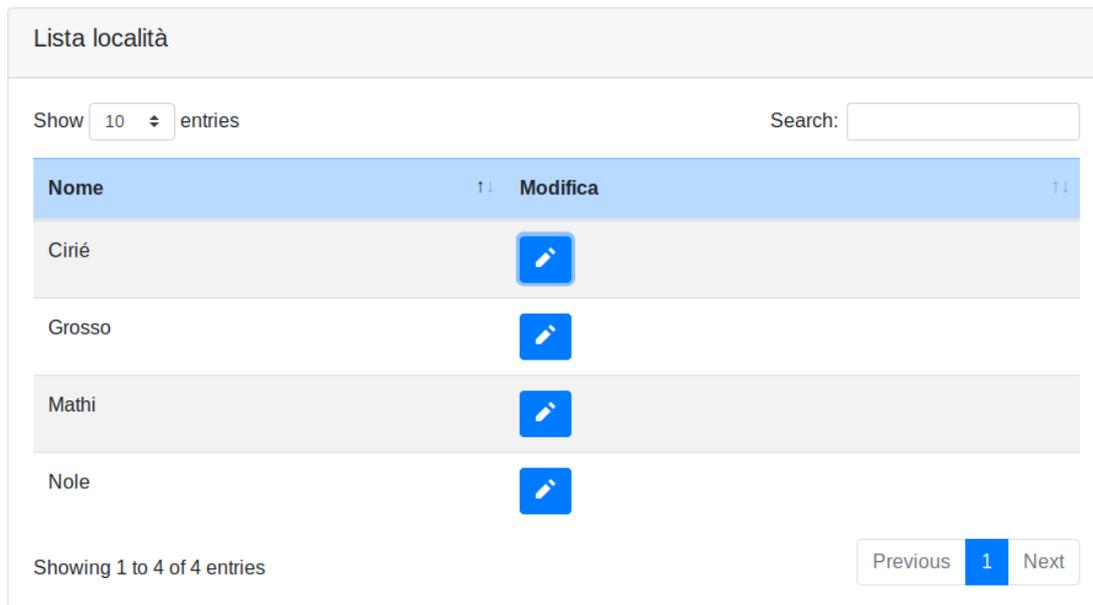


Fig. 5.12 The list of already inserted localities

## 5.4 Insert/edit of the researched building or city building

Because the concepts of researched building and city building have many common features within the project, we present them in the same chapter. The main differences between them are the played role of each in the project. Researched buildings are clearly to be the object of study of the researchers who uses this platform, and city buildings are meant to determine the neighborhood in which the researched objects are located. To display the insertion form for the buildings and city buildings, on the main menu should be chosen *Palazzi* and *Edifici comunali* options, respectively. Before the display of the forms, it is necessary to choose the locality, which contains the building. This feature is implemented using the Typeahead.js plugin, which adds to the input autocomplete search functionality. As the source dataset again an asynchronous GET request is used, from the **Location** collection. Fig. 5.13 shows all described above actions.

Both forms have collective and individual inputs, depending on the information to be stored about one entity or another. The building page can store the name of the building, previous names, its recognizability, and of course, geolocation (Fig. 5.14). The city building page contains more general information as the name of the building, type, functionality, and geolocation. An essential feature of the interactive map used to determine the position of buildings is the already displayed surface of the locality. It allows facilitating map navigation and search of the building. JQuery script retrieves the list of coordinates of the current location and draws on the map of the shape that includes the borders of the locality. After it automatically zooms in and centers the map in order to cover in the window all the borders of the locality.



Fig. 5.13 Entering in building or city building pages

Also, here are the *Salva* and *Resetta il modulo* buttons described in the previous chapter, which have the same functionality. The inserted into the form information is saved in **Building** and **City\_building** collections. The tables which contain already inserted data have the same form, features, and functionalities like that from the location page, and it is not worth to repeat the description.

The image shows a web form titled 'Nuovo Palazzo'. It contains several input fields: 'Denominazione moderna dell'edificio' with the text 'Palazzo dei Marchesi D'Oria', and 'Denominazioni antiche' which is empty. There is a 'Riconoscibilità' toggle switch that is turned on. Below this is a 'Geolocalizzazione' section featuring a map of the Cirié area with a blue location pin. At the bottom of the form, there are two buttons: a green 'Salva' button and a red 'Resetta il modulo' button.

Fig. 5.14 Form for inserting new building

But for buildings, another feature is present in the project, the possibility to make a short description of it. As this text appears on the public page of the site, it is necessary to stylize it to ensure a coherent and beautiful display on the page. The description unit was moved to a separate page in order not to hinder the original insertion form of the buildings. To access it *Sintesi Palazzo* option from the main menu should be clicked. Before proceeding to edit the description, it is obvious that firstly the researched building should be chosen. All presented above actions are displayed in Fig. 5.15.



Fig.5.15 Entering in the building's short description page

As the project was designed as a multi-lingual platform, it gives the possibility to insert the description in any platform languages. And, of course, the display on the public part is in that language which is set by the user. The form also permits to indicate the authors, to format the text, by applying different style features (colours, fonts, and others), and to insert a lot of HTML entities (links, images, tables, paragraphs, headings, and others). In order to do all the above-described stuff, we use the TinyMCE editor (free HTML editor written in JavaScript), which permits to edit the text with all those features. Fig. 5.16 shows all the above functionalities. Also, *the Salva* button is used for saving the description in the DB. It is also stored in **Building** collection as a new field in already existing documents.

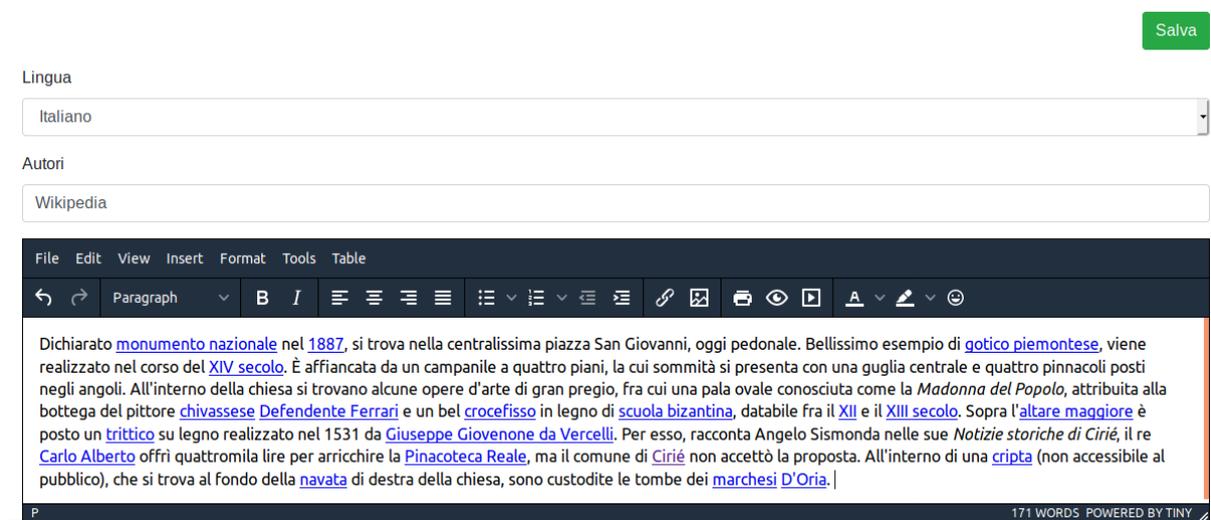


Fig.5.16 Form for inserting the short description of the building

## 5.5 Insert/edit of a new descriptive element

Now it is time to see how the management of the platform's most complicated items is designed and implemented. Of course, it is about the descriptive elements. We can call them the most complex entities because, through them, the linkage is made between all the elements described in previous paragraphs. Considering their complexity and for ensuring the best organization and display of them, we decided to move from the well-known template with the insertion form and the list of already inserted objects on the same page. The traditional template was split into two pages. To enter on the insertion page, from the main menu *Inserimento di elementi descrittivi* option should be chosen, for the display – *Elenco di elementi descrittivi* should be clicked (Fig. 5.17). The schematic workflow of this process is presented in Fig 5.18. It is also worth mentioning that the insertion of a new descriptive element can be

done directly after the insertion of a new source. Thus, the application jumps to the insertion form described in Fig. 5.19 with the “Fonti” input already completed with the value inserted at the previous stage. This approach was implemented to simplify the process of introducing information by creating a pipeline that models the actual process of describing data continuously within the research in the field of cultural heritage.

Fig. 5.17 Entering into descriptive elements insertion/display pages

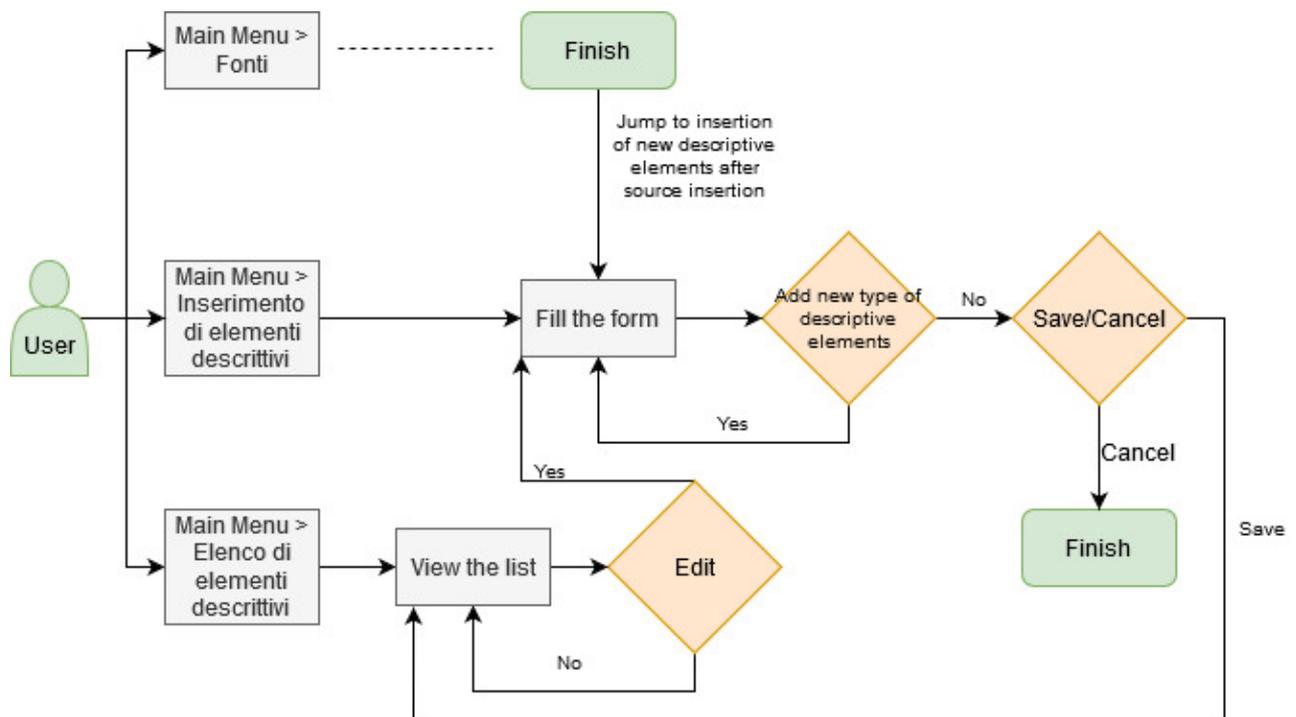


Fig. 5.18 Descriptive elements management workflow

Let’s start like always with insertion form. It was designed to permit the insertion of several descriptive elements at a time. It can be divided into two blocks. In the first one, which remains unchanged from insertion to insertion, can be compiled general information of elements: the list of the sources where these elements are mentioned, location, and the building where the descriptive elements are located. By the way, the list of sources is also attached to amsify.suggestags and typeahead.js plugins. It combines the feature of both libraries because the input should have as well as multi-select and autocomplete search functionalities. The second block permits to define each element one by one, by selecting the type of element from the last input on the form (Fig. 5.19).

After picking the desired element type, a new block of inputs is added to the form (Fig. 5.21), and the Combobox for selecting another type is moved to the bottom of the form. It contains the form configured in paragraph 5.1 of the current chapter. The server part of the adding process of the form in order to insert a new descriptive element is similar to that described at source insertion.

Nuovo elemento

[Salva](#)

Fonti\*

Giovani Crosetto Cirié tra sviluppo e guerra 1900 - 1938 ✕  
 Mario Minardi; Ermanno Franchetto Il Canavese ieri e oggi ✕ Type here

Città

Cirié

Palazzo\*

Palazzo dei Marchesi D'Oria

Aggiungi nuovo elemento descrittivo

Selezionare un elemento

Selezionare un elemento

Pitture

Epigrafi

Sculture

Fig. 5.19 Descriptive elements insertion form

In case we want to save the already completed elements, it is necessary to press the **Salva** button. It is worth mentioning that even if we inserted more descriptive elements at once, in the DB, they are saved as separate entities. All the descriptive elements are stored in the **Element** collection. After the save process, the platform automatically redirects us to the view page of already inserted elements, which is a big table containing all the information in a brief form (Fig. 5.20). Like all other view pages described above, it can modify any existing element by clicking *the Modifica* button from the table. In this case, it redirects us to the insertion page with the already completed form. Saving the modifications updates the entries in DB.

Lista elementi descrittivi

Show 10 entries Search:

Fonti	Località	Palazzo	Tipo Elemento	Elemento	Modifica
Giovani Crosetto Cirié tra sviluppo e guerra 1900 - 1938 ;Mario Minardi, Ermanno Franchetto Il Canavese ieri e oggi ;	Cirié	Palazzo dei Marchesi D'Oria	Pitture	Romano Gazzera Ritratto di Emanuele D'Oria	<a href="#">Modifica</a>
Giovani Crosetto Cirié tra sviluppo e guerra 1900 - 1938 ;Mario Minardi, Ermanno Franchetto Il Canavese ieri e oggi ;	Cirié	Palazzo dei Marchesi D'Oria	Pitture	Giovanni Panealbo Maria Lesbia Cristina D'Oria	<a href="#">Modifica</a>

Showing 1 to 2 of 2 entries

Previous 1 Next

Fig. 5.20 Descriptive elements view page

The form is titled "Pitture" and includes a red close button (X) in the top right corner. It contains several input fields:

- Autore\***: Input field containing "Romano Gazzera".
- Titolo\***: Input field containing "Ritratto di Emanuele D'Oria".
- Tecnica**: Input field containing "olio su tela".
- Soggetto**: Input field containing "Ritratto".
- Elementi testuali**: Text area containing "Firma di autore e anno".
- Elementi araldici**: Text area containing "Elementi araldici".

Fig. 5.21 Block for defining descriptive element characteristics

## 5.6 Translation process

Considering the fact that the platform is designed to be multilingual, the Django web framework has been chosen because it allows the creation of sites in several languages. It has a complete set of tools to solve this problem: translation of a text, formatting of date, time and numbers, and support for time zones. But to ensure internationalization of the platform, Django needs translations for every language estimated to be used in the project. To cover that inside of the platform, a mechanism for translations and exporting them to Django was designed and implemented. In other words, users by themselves can translate the content of the platform during use. Besides this, to apply the translation, it is necessary to mark within the platform all the messages which need to be translated. In our project, it is done at the template level. The translation module could be reached from the main menu, under the *Configurazione* option and select the tab *Traduzioni* (Fig. 5.22).



Fig. 5.22 How to enter the translation module

In fact, it consists of a search textbox and three buttons. *Aggiungi* button opens a form to insert a message and its translations in configured languages (Fig. 5.23). Clicking *the Save* button, we store all the data to the DB in **Translation** collection. But in order to apply the translation on the platform, we should click *the Compila* button, which exports all translated expression to .po files, individually for every language, used by Django to display the correct form of the message. These files are usually created by translators and then are only attached to projects, but we allowed their creation directly from the application. In Fig. 5.25 are shown for comparison, the same two pieces of a form, one when the French language is set, the other when the Italian one is set. *Cerca* button is used for searching among already translated messages using the text from textbox as the search string. The result of the search is displayed below as a list (Fig. 5.24). It allows editing translations by clicking *Modifica* buttons.

The screenshot shows a web form titled "Traduzione". At the top, there is a search bar with the text "Search" and three buttons: "Cerca" (light blue), "Aggiungi" (blue), and "Compila" (green). Below the search bar, the form is divided into three sections: "Messaggio" with a text input field containing "Messaggio"; "Italiano" with a text input field containing "Messaggio"; and "Français" with a larger text input field containing "Message". At the bottom left of the form is a green "Salva" button.

Fig. 5.23 Insertion form for new translations

The screenshot shows the search results page. At the top, there is a search bar with the text "Search" and the value "ti". To the right of the search bar are three buttons: "Cerca" (light blue), "Aggiungi" (blue), and "Compila" (green). Below the search bar, there is a table with two columns: "Messaggio" and "Modifica". The table contains four rows of search results, each with a blue edit icon in the "Modifica" column.

Messaggio	Modifica
Titolo	
Titolo della raccolta o della rivista	
Fonti di seconda mano	
Fonti edite	

Fig. 5.24 The result of the search

Fig. 5.25 Differences between the same piece of form displayed in French and Italian languages

## 5.7 Users management

Taking into account that the functionalities of the platform require differentiation between categories of users, we decided to implement a user management module. In general, in the system are 3 types of access: Guest, Researcher, and Administrator. For having granted Guest access user doesn't need to have any userID, but he has access only on the public part of the platform. For having Researcher access type user should have a userID, and he is allowed on private site to work only with researching stuff: like inserting/editing new sources, buildings, descriptive elements. Administrator access provides all the possibilities covered by the Researcher, plus it allows us to configure entities of the platform (source types, element types, translations) and the creation of the new users.

The user management system covers the entire range of operations with its user: login and logout from the system, change password, recovery password. It also allows the creation of new users with limited rights (Researcher access) by administrators. Let's start with the login process. In order to login user should type username and password in the form presented in Fig. 5.27 and click *the Submit* button.

In case of a successful login, the platform redirects the user to the homepage of the private site. In case the user forgot his password and want to reset it, *the Reset Password* button should be clicked. As a result, the user is redirected to the password reset page where he can insert his email address and request a new password (Fig. 5.26). He receives an email with a hyperlink to the password reset confirmation page, where the user should insert new passwords and confirm.

Fig. 5.26 Reset password form

Username\*

Password\*

Submit

Reset Password

Fig. 5.27 Login form

When the user is logged in the system, he can see his profile (Fig. 5.28). It can be done by click on the username in the navigation bar. Also, from the navigation bar, the logout process can be started by clicking on **Logout**. In the profile page is presented the button **Change password** and clicking on it, platform redirects the user to the page where he can modify his password (Fig. 5.29).

Atlante dei palazzi comunali Italiano GO 11 Febbraio 2020 Logout [zAdmin] Private

Account details

Username zAdmin

First Name

Last Name

E-mail geniusk546@gmail.com

Change Password

Fig. 5.28 User profile page

Cambio password

Password attuale\*

Nuova password\*

- La tua password non può essere troppo simile alle altre tue informazioni personali.
- La tua password deve contenere almeno 8 caratteri.
- La tua password non può essere una password comunemente usata.
- La tua password non può essere interamente numerica.

Conferma nuova password\*

Change

Fig. 5.29 Change password page.

## 5.8 Statistics dashboard

The homepage of the private site is a dashboard that contains statistical information in graphic form or as a simple list, which highlights the amount of data inserted in DB. The template of the dashboard consists of two rows of bootstrap cards, with 3 cards in each. In detail, it can be seen in Fig. 5.30. In the future, existing cards can be modified, or other ones can be added to show new relevant data. At this moment, the following information is displayed: Number of buildings in location, Number of elements in location, Number of elements of each type, Number of sources of each type, List of last researched localities, List of last login users.

In order to present statistical information in graphical form, Chart.js library was added to the platform. It permits drawing many kinds of charts with a lot of options and features. It allows us to choose the type of chart, to display or no title, axis, values, etc., to set the used colours. Also, it requires as an input a dataset, which should be a list of id-value pairs. In order to complete this dataset, all the information is retrieved on the fly, using the same asynchronous GET request as in other cases. But here, the queries are a little more specific because we have to display aggregated and grouped information. Even if non-relational databases are not designed to perform such operations, MongoDB is one of the most performing and allows them to use the aggregate method. In SQL `count(*)` and with the `group by` is an equivalent of MongoDB aggregation. It matches with pipeline concept, which allows to execute an operation on some input and use the output as the input for the next command and so on. In order to obtain the desired results, it was necessary to configure the pipeline for the aggregate method with the following methods: `lookup` - to link 2 collections; `project` - to select certain fields; `group` - to group data; `sort` - to sort the result etc.

Code
<pre>pipeline=[{"\$lookup":   {"from": "building",    "let": { "palazzoid": "\$palazzoid" },    "pipeline": [{ "\$match": { "\$expr": { "\$eq": [ "\$_id", "\$\$palazzoid" ] } } } ],   }, {"\$project": { "locationid": 1, "_id": 0 } } ],   "as": "locids" } },   { "\$replaceRoot": { "newRoot": { "\$mergeObjects": [ { "\$arrayElemAt": [ "\$locids", 0 ] }, "\$\$ROOT" ] } } },   { "\$project": { "locids": 0 } },   { "\$group": { "_id": "\$locationid", "count": { "\$sum": 1 } } },   { "\$sort": SON([ ("count", -1), ("_id", -1) ]) } ] elements=db.elements.aggregate(pipeline)</pre>

Tab. 5.1 Example of aggregate method

In Tab 5.1 is illustrated a real example of using MongoDB aggregate method in our platform. It is worth mentioning the pipeline style of writing the function that reduces readability but offers a wide range of possibilities in terms of combining different methods.

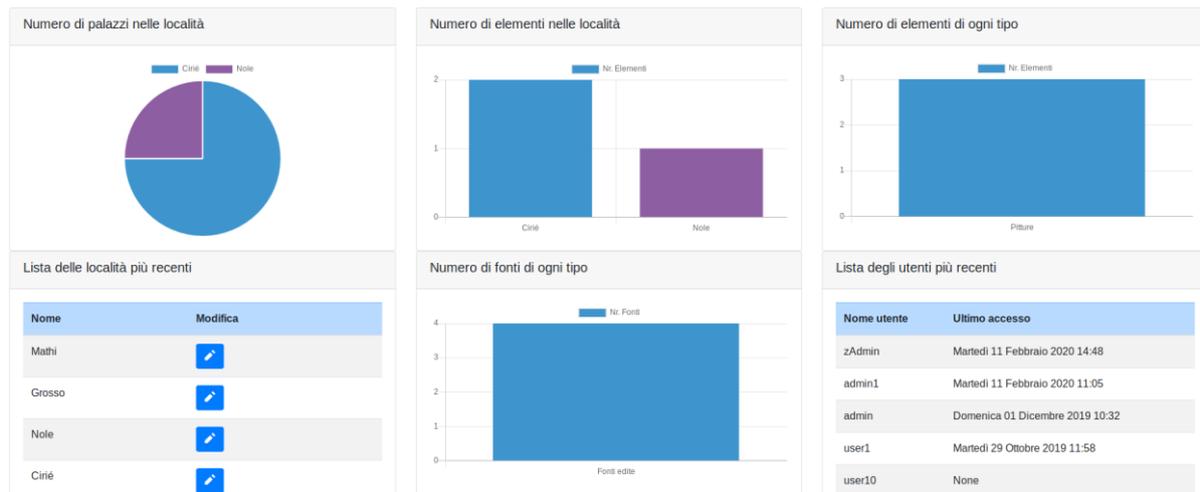


Fig. 5.30 Homepage dashboard.



# Chapter 6

## Conclusions

In the contemporary era, as the pace of life increases, humanity begins to forget its links with the past. In this context, in more and more cases, the cultural heritage is abandoned and deteriorates in time either due to natural or human-made disasters. Therefore, the elaboration of tools that will increase the level of knowledge in the cultural field will definitely stimulate the interest of local communities in documenting and preserving their cultural riches.

As has been pointed out in previous chapters, the primary purpose of our project was to develop a platform for a group of researchers working in the architectural and cultural field. "Atlante dei palazzi comunali" is a web application that fully corresponds to the submitted requirements, namely, to ensure the configuration, storage, visualization, and linking of several types of entities that are involved in the research. It is worth mentioning also the design of the graphical interface, which fully corresponds to the functional part of the requirements, but also has a pleasant and modern look.

This thesis deals with some particularities of the platform. First of all, it would be the positioning of the study objects in space, by determining their geographical coordinates and in this way to establish their neighborhood. Second would be the storage of detailed information about the documentary sources where the researched object is described. We must also mention the possibility of developing or modifying some modules of the application directly by the user. Of course, it is about the module of Configuration / Modification of the types of source and descriptive elements. And last but not least, the possibility of transforming the platform into a multi-lingual one, by the user, could be considered as an achievement of the platform. All the above mentioned make this project very challenging.

The process of developing the application can be divided into three stages. In the first preliminary stage, the design of the future platform was carried out. Also, research was done on the available technologies that would allow the implementation of the application in

minimum times and with maximum coverage of the requirements. Certainly, after this research, all the necessary tools and frameworks were selected: Docker as deployment environment, Django and Bootstrap as application development frameworks and MongoDB as Database Management System plus other small libraries and plugins.

In the second phase, all the requirements within the platform were implemented most simply and efficiently. In particular, the collections from MongoDB were modeled, the templates and views from Django were developed and customized with the active use of Bootstrap components. Also, the Leaflet library was used intensively to implement an interactive map for positioning the investigated objects in space. Not least, it is worth to mention the containerization of all systems within the Docker.

The last phase is the process of testing the application, where due to a series of test and use cases, several bugs and disparities with the initial requirements were determined, and later corrected.

Even though the developed application covers a wide range of functionalities, it should be treated as a starting point where there is room for future enhancements. First of all, the existing modules could be upgraded to include more useful information about the researched entities (especially buildings and localities). This would be useful to diversify the informational nature of the platform and to create an advanced search for these entities. Secondly, besides the information compiled within the application, it can be considered as an improvement, the possibility of attaching to each type of entity the documents or photographs in electronic format, which will significantly increase the credibility of the researched facts and will serve as a proof.

Another great enhancement would be 3D modeling. It would allow the visual contact of the researcher with the building he is studying inside the neighborhood, or with the position of the interesting descriptive element inside the building. It would also facilitate the creation of a 3D map that can contain entire neighborhoods of buildings studied and, in combination with previous improvements, would increase the interactivity of the graphical interface.

Last but not least, the development of the public part of the site, by providing access to other information contained within the platform, would increase the potential of the platform to be useful not only to the group of researchers but also to the general public. Also, implementing a feedback system with mass users would increase the possibility of discovering some architectural and cultural treasures in areas where researchers do not have access or have little information about it.

Considering all the advantages of the application and its possible improvements, we can conclude that the platform "Atlante dei palazzi comunali" has a future not only in the academic environment but has the potential to become the leading tool for collecting and preserving data regarding cultural heritage.

# Bibliography

- [1] *UNESCO - Convention Concerning the Protection of the World Cultural and Natural Heritage.*  
URL: <https://whc.unesco.org/en/conventiontext/>
- [2] Giuseppina Vacca (04.02.2018). *A Spatial Information System (SIS) for the Architectural and Cultural Heritage of Sardinia (Italy).*  
URL: <https://www.mdpi.com/2220-9964/7/2/49/htm>
- [3] *Carta del Rischio.*  
URL: <http://www.cartadelrischio.it/index.asp>
- [4] *THEMAS - Thesaurus Management System.*  
URL: <https://www.ics.forth.gr/isl/themas-thesaurus-management-system>
- [5] John Mylopoulos, Alex Borgida, Matthias Jarke, Manolis Koubarakis (October 1990). *Telos: Representing Knowledge About Information Systems.*  
URL: <http://www.cs.toronto.edu/~jm/Pub/Telos.pdf>
- [6] *ResCult - Increasing Resilience of Cultural heritage.*  
URL: <https://www.rescult-project.eu/>
- [7] *Hermoupolis Digital Heritage Management (HERMES).*  
URL: <https://hermoupolis.omeka.net/>
- [8] *Arches - An open source data management platform for the heritage field.*  
URL: <https://www.archesproject.org/>
- [9] *Docker: Empowering App Development for Developers.*  
URL: <https://www.docker.com/>
- [10] *Kubernetes: Production-Grade Container Orchestration.*  
URL: <https://kubernetes.io/>
- [11] *MongoDB: The most popular database for modern apps.*  
URL: <https://www.mongodb.com/>
- [12] *Vue.js.*  
URL: <https://vuejs.org/>
- [13] *React - A JavaScript library for building user interfaces.*  
URL: <https://reactjs.org/>
- [14] *Angular.*  
URL: <https://angular.io/>
- [15] *Bootstrap · The most popular HTML, CSS, and JS library.*  
URL: <https://getbootstrap.com/>
- [16] *The Principles of the MVC Design Pattern.*  
URL: <https://coderwall.com/p/1-a79g/the-principles-of-the-mvc-design-pattern>

- [17] *Ruby on Rails*.  
URL: <https://rubyonrails.org/>
- [18] *Django: The Web framework for perfectionists with deadlines*.  
URL: <https://www.djangoproject.com/>
- [19] *Stackoverflow Developer Survey Results 2019. Programming, Scripting, and Markup Languages*.  
URL: [https://insights.stackoverflow.com/survey/2019?utm\\_source=social&utm\\_medium=blog&utm\\_campaign=dev-survey-2019&utm\\_content=launch-blog#technology--programming-scripting-and-markup-languages](https://insights.stackoverflow.com/survey/2019?utm_source=social&utm_medium=blog&utm_campaign=dev-survey-2019&utm_content=launch-blog#technology--programming-scripting-and-markup-languages)
- [20] *The DRY Principle: Benefits and Costs with Examples*.  
URL: <https://thevaluable.dev/dry-principle-cost-benefit-example/>
- [21] *Django REST framework*.  
URL: <https://www.django-rest-framework.org/>
- [22] *RESTful web services*.  
URL: <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>
- [23] *jQuery – JavaScript library*  
URL: <https://jquery.com/>
- [24] *Cos'è una Sitemap?*  
URL: <https://www.rankingcoach.com/it-it/news/cos%27%C3%A8-una-sitemap->