Struttura della Scheda Tecnica:

1. **Nome del Software**

   RASTA Poi-Management System

2. **Descrizione**

POI Management System (PMS) is designed to manage Points of Interest (POIs), allowing contributors to create, edit, and publish related information. This system is a data-management layer for the Points of Interest and itineraries, providing back-end and front-end functionalities. The PMS is divided into back-office and the frontend of the application. The front-end may be public and show the inserted data to a public user. The back office offers an administrative layer that supports the data entry of POIs.

3. **Tecnologie Utilizzate**

Below are the technologies and tools that we used in the design and implementation of the Poi-management system of RASTA.

Docker is used to containerize applications, providing a consistent environment across development, testing, and production. **Azure** serves as the cloud infrastructure, offering customizable virtual machines, scalable storage, and network management. **EMQX**, a high-performance message broker, enables real-time communication between IoT devices using MQTT and other protocols, supported by its visual Flow Engine and powerful Rule Engine for data processing. For storing time-stamped data such as sensor readings, **InfluxDB** is used as a time-series database, while **Telegraf** acts as the data collection agent, forwarding data from EMQX to InfluxDB. **Grafana** provides rich, interactive dashboards for monitoring and visualizing this data. The system also uses the **Apache Web Server** to handle HTTP requests and serve web content, typically deployed within Docker containers. Backend services are developed using **Spring Boot**, and AI features are integrated via **Spring AI**, allowing seamless use of generative AI models like those provided by **OpenAI**. Though now deprecated, **Apache Stanbol** was initially used for semantic content enrichment and entity extraction, and is now evaluated against modern LLM-based solutions. Finally, **Apache Mahout** supports advanced machine learning tasks such as clustering, classification, and recommendation on large datasets. To see the complete details of each technology see the deliverable document D1.2 .

4. **Requisiti**

These are the software requirements needed to install and run the POI Management System for the RASTA project :

- Microsoft Windows, Mac OS X, or Unix Operating Systems and derived are supported.
- The latest OpenJDK 21 (or other Java distribution like Eclipse Adoptium https://adoptium.net/) is recommended.
- Latest Apache Maven installed.
- A MySQL instance. The MAMP (for the local installation) product contains MySQL for Mac OS or Windows.

5. **Installazione**.

To launch the RASTA PMS application follow this step by step procedure:

Create a user and a database with the following data:

```
username = rastapms
```

```
password = rastapms
```

```
database = rastapms
```

The user should have the right permission to access and modify the database. You can use a terminal or Phpmyadmin if installed.

Run the `rasta-pms.sql` SQL script file contained in the database folder. If the username, password, and database differ, please update the file application-prod.yml accordingly.

Start Rasta PMS application.

To launch the Rasta PMS application, run these maven commands that will start the application on port 8080:

Open a terminal in the directory where you downloaded the source code and type the following command: `mvn clean compile`

Open a terminal in the directory where you downloaded the source code and type the following command: `mvn spring-boot:run`

Open a browser and type the following URL: `http://localhost:8080/rastapms/`

6. **Usage**

In the following, we provide the source code repositories for the POI Management System (PMS), along with instructions on how users can access the application and interact with its features. This includes guidance on logging in, navigating the interface, and using the system for managing Points of Interest.

There are three repositories in which the POI Management System (PMS) implementation is stored, along with the installation procedure. We also propose a Dockerized installation.

7. https://github.com/gssi/official-rasta-pms (source code and installation of CMS)
8. https://github.com/gssi/rasta-me (REST Services)
9. https://github.com/gssi/official-rasta-or6 (instance of the PMS for OR6)

The homepage of the RASTA POI Management System (PMS) is presented in Figure 1 showcasing the main interface and user entry point to the platform. Users can access the login functionality by clicking the arrow icon located in the top-right corner of the interface , which redirects them to the login page.

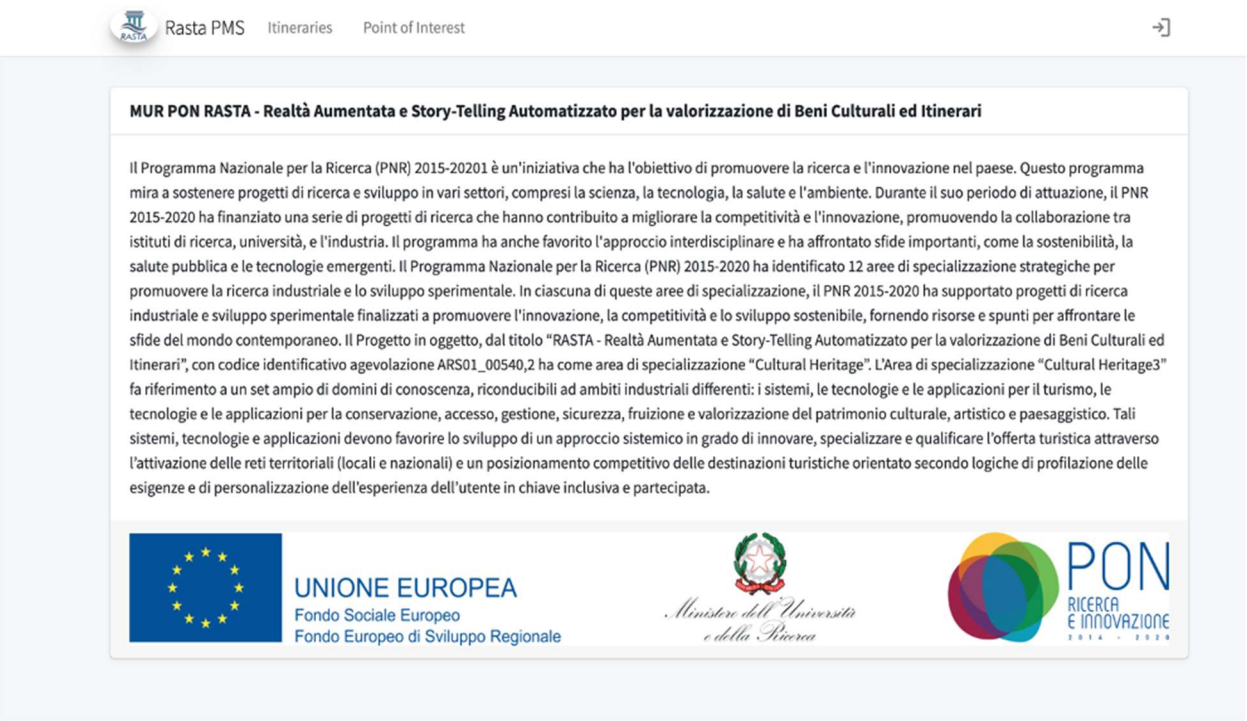


Figure 1: RASTA PMS Homepage

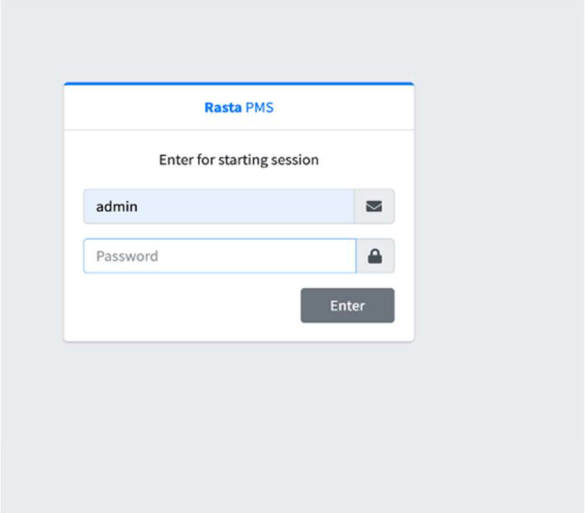


Figure 2: Login Interface of the RASTA POI Management System

We report a screenshot of the back-office to show the interplay of data entry, via web-based forms and semantic enrichment and automated categorization. The web-view where the user manages data about a POI see Figure 3 allows interaction with OpenAI to enrich the POI description with web links. Moreover, the user can automatically categorize the POI by clicking a button see Figure 4. The prompts used to interact with the LLM are detailed in D1.3, with all the experiments conducted to fine-tune the application. We have added 21 POIs descriptions .

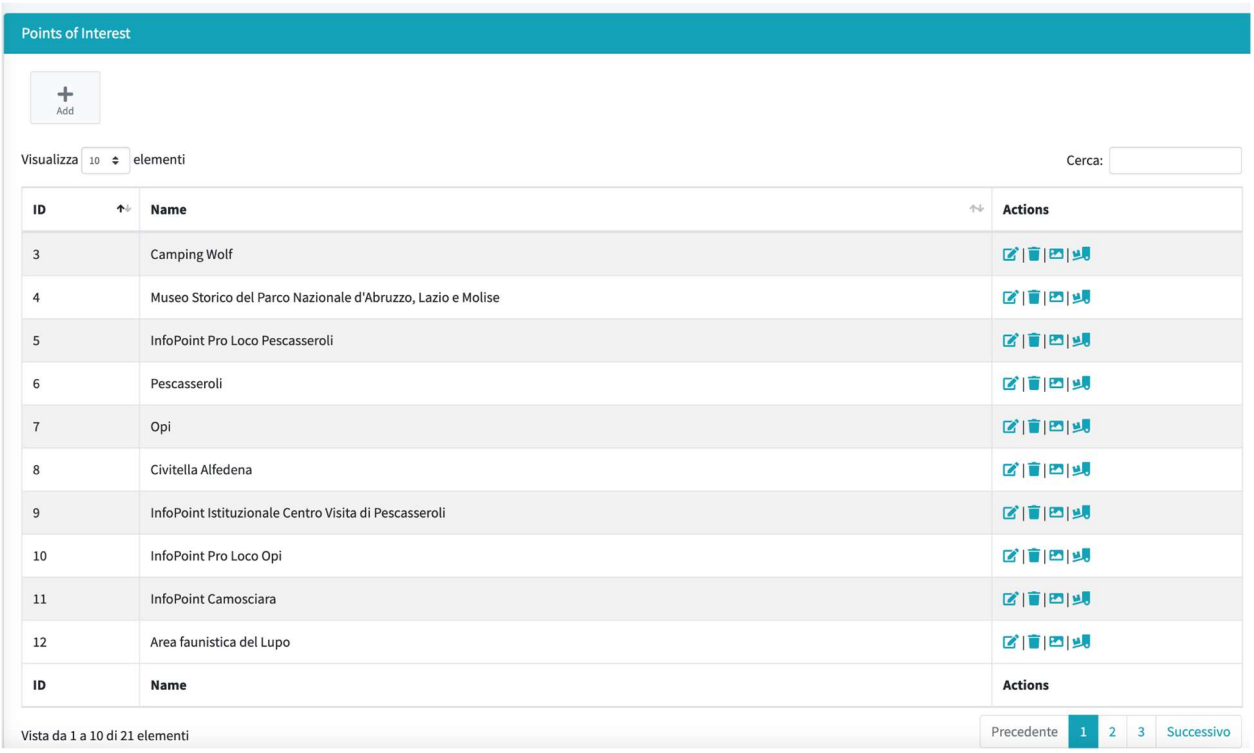


Figure 3: POIs from National Park

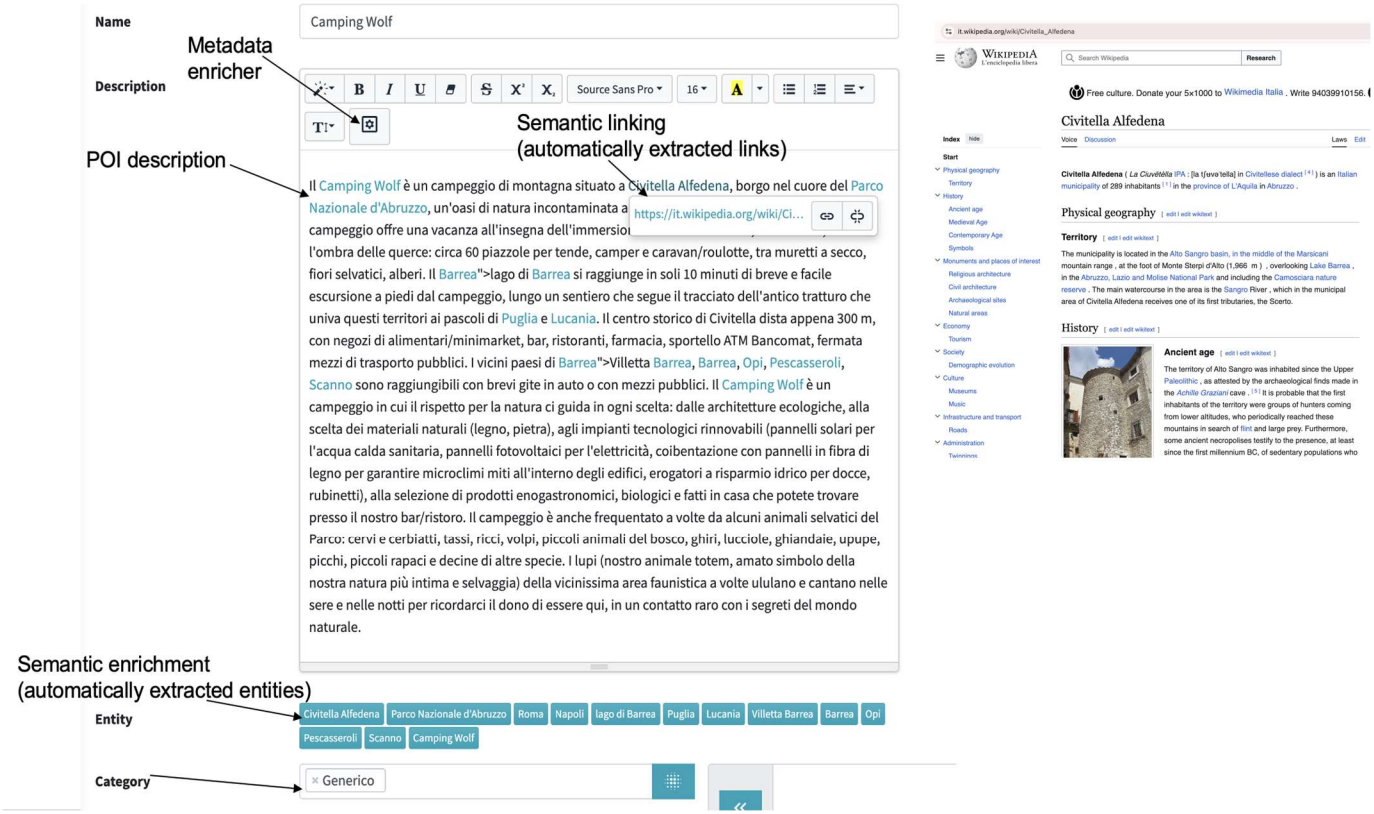


Figure 4 : Automatic Semantic enrichment and linking of POI

The front-end of the application offers a showcase of the POIs:

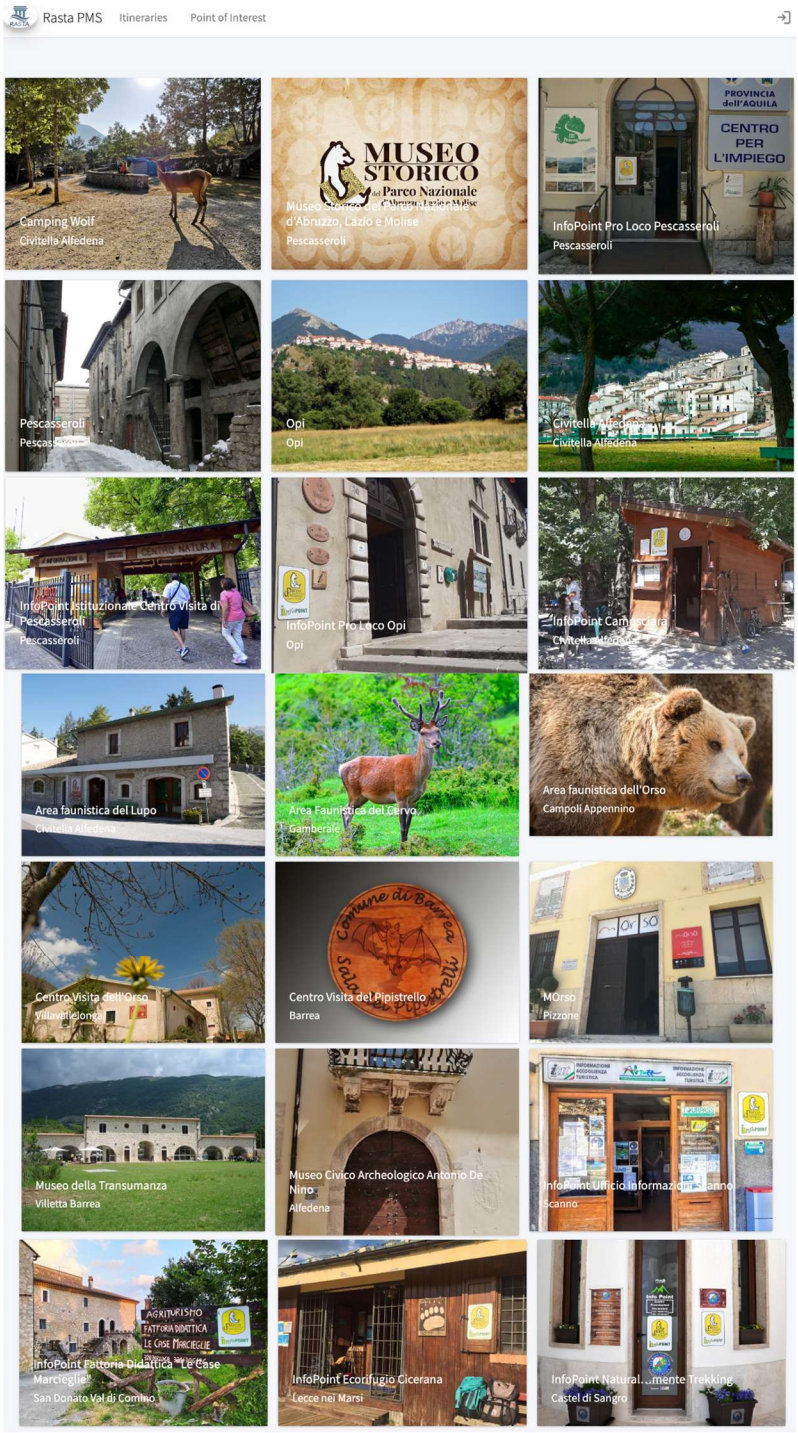


Figure 5 : Front end of applications with POIs

10. **Configurazione**

The entire architecture will run on Docker as independent modules. The deployment procedure is explained in Deliverable D2.6.

11. **Contatti**

The following members from the Gran Sasso Science Institute (GSSI) are involved in the RASTA POI Management System project:

Sadia Azam, Maria Giovanna Brandano, Gianlorenzo D'Angelo, Martina De Sanctis, Amleto Di Salle, Paola Inverardi, Ludovico Iovino, Rickson Pereira, Claudio Pompilio, Franco Raimondi

For any issues, technical support, or inquiries, please contact the team. Email addresses are available on the official GSSI website.

Struttura della Scheda Tecnica:

12. **Nome del Software**

Itineraries and Point of Interests addition in RASTA PMS (instantiation of the developed modules in the context of Abruzzi National Park)

Descrizione

The software is developed to facilitate the addition and management of POI descriptions within the RASTA POI Management System (PMS). Real POI descriptions from the Abruzzo National Park have been incorporated as part of the initial dataset.

13. **Tecnologie Utilizzate**

These technologies and tools are used in design and implementation of this software. Docker, Azure, and Apache Web Server provide the core infrastructure for deploying and hosting the system. EMQX, InfluxDB, Telegraf, and Grafana manage real-time IoT data collection, storage, and visualization. Backend services use Spring Boot and Spring AI for application logic and AI integration, while entity enrichment and machine learning are supported by OpenAI, Apache Stanbol, and Apache Mahout (see Deliverable D1.2 for full details)

14. **Requisiti**

Docker has to be installed on the machine.

15. **Installazione**

For the local installation to add itineraries and Point of Interests related to the Abruzzo National Park, run the pna-data.sql sql script file in the rastapms database. This can be accomplished through a terminal or a database management tool such as phpMyAdmin. As a prerequisite,, it is necessary to follow the official installation instructions for the RASTA PMS, available at: https://github.com/gssi/official-rasta-pms/blob/main/Install.md

For production deployment, the system architecture is operated using Docker. Open a terminal in the directory where the source code is downloaded and type the following command:

```
docker exec -i mysql-db mysql -u rastapms  -prastapms  rastapms
--default-character-set=utf8mb4 < ./pna-data.sql
```

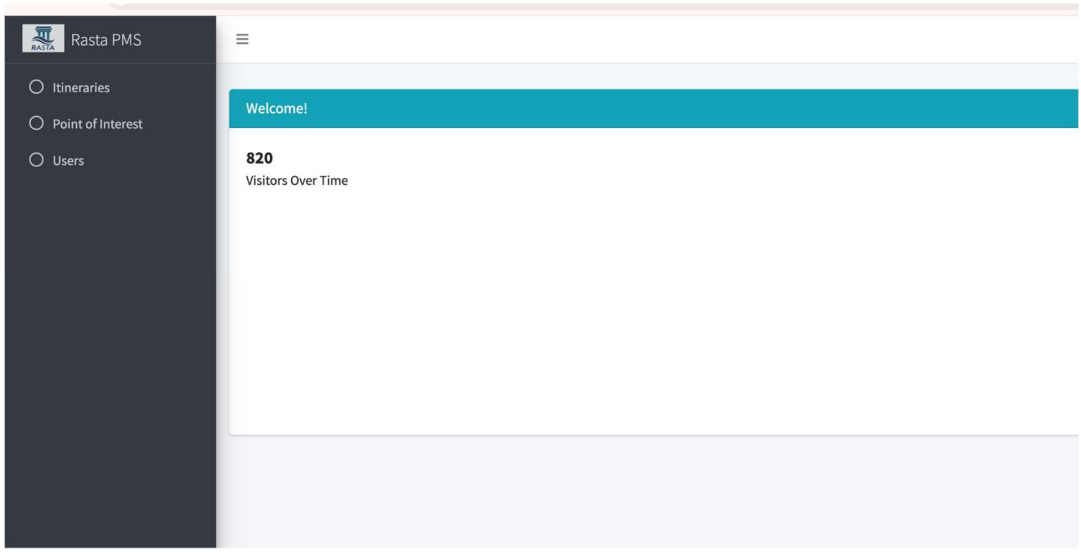
On Windows systems, the following command should be used:

```
type .\pna-data.sql | docker exec -i mysql-db mysql -u rastapms
-prastapms rastapms --default-character-set=utf8mb4
```

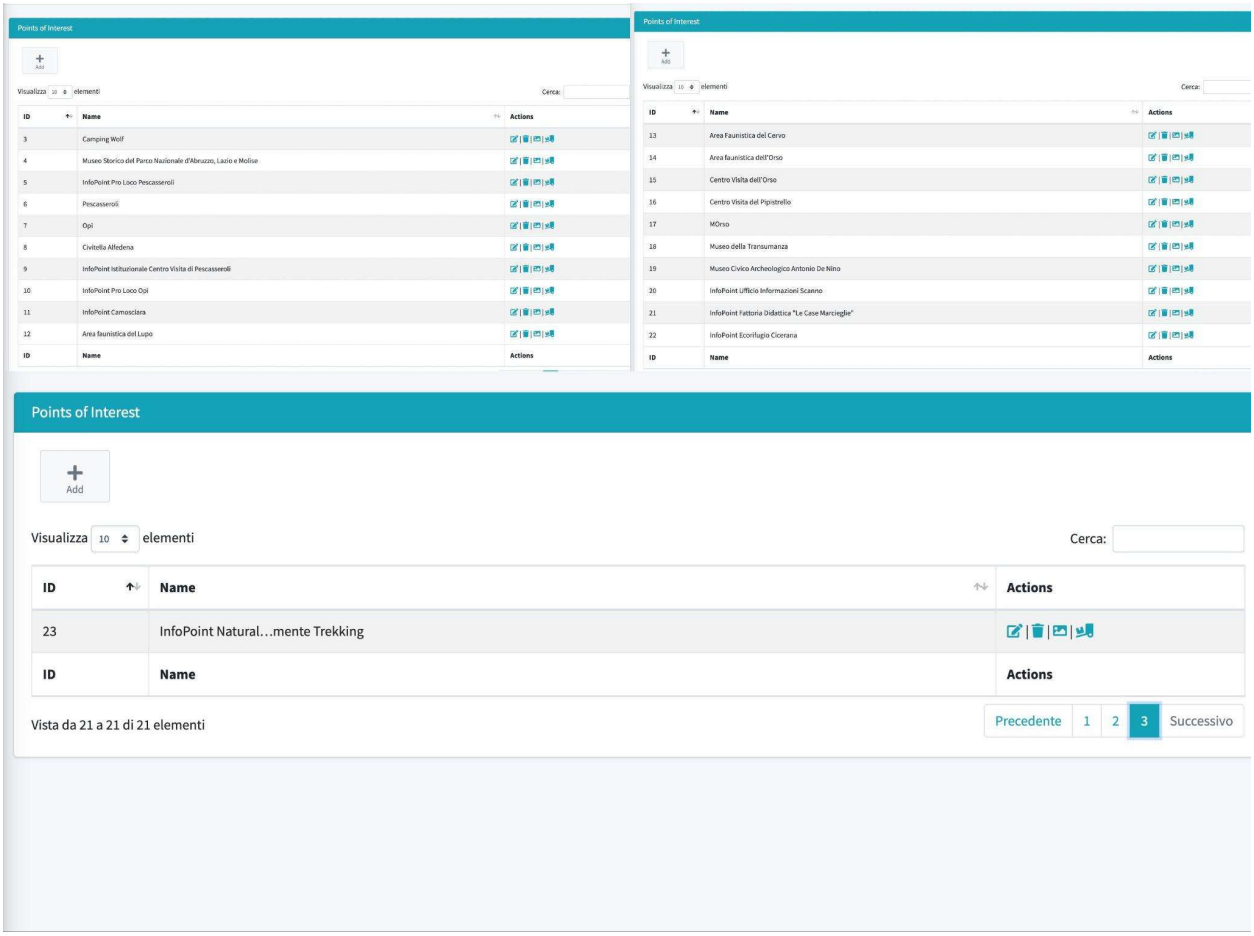
16. **Usage**

In the following, we provide the link source code repository for adding the POI Management System (PMS) Itineraries and Point of Interests addition in RASTA PMS .https://github.com/gssi/official-rasta-or6 (instance of the PMS for OR6).

When the user logs into the RASTA PMS interface, the complete procedure for starting the application and logging in is described in the RASTA Management System section. Below are the screenshots where the user can view the added itineraries and POIs. In the first screenshot, when the user clicks on **Point of Interest** on the left-hand side, the list of POIs is displayed, as shown in the second screenshot. These POIs are taken from the website of the National Abruzzo Park https://www.parcoabruzzo.it/page.php?id=251. Users can perform different actions on these POIs.



(a)



(b)

Figure 1: List of all the added POIs in the RASTA PMS

17. **Configurazione**

The entire architecture will run on Docker as independent modules. The deployment procedure is explained in Deliverable D2.6.

18. **Contatti**

The following members from the Gran Sasso Science Institute (GSSI) are involved in the RASTA POI Management System project:

Sadia Azam, Maria Giovanna Brandano, Gianlorenzo D'Angelo, Martina De Sanctis, Amleto Di Salle, Paola Inverardi, Ludovico Iovino, Rickson Pereira, Claudio Pompilio, Franco Raimondi

For any issues, technical support, or inquiries, please contact the team. Email addresses are available on the official GSSI website.

Struttura della Scheda Tecnica:

19. **Nome del Software**

Implementation of the IoT infrastructure and Dashboards

Descrizione

In the following section, we provide the deployment details of the IoT infrastructure that is part of the architecture proposed in Task OR1.6 and OR2.6. We also describe the design and functionality of the associated dashboards used in the implementation.

20. **Tecnologie Utilizzate**

These technologies and tools are used in the design and implementation of this software. Docker, Azure, and Apache Web Server provide the core infrastructure for deploying and hosting the system. EMQX, InfluxDB, Telegraf, and Grafana manage real-time IoT data collection, storage, and visualization. Backend services use Spring Boot and Spring AI for application logic and AI integration, while entity enrichment is supported by OpenAI, Apache Stanbol, and Apache Mahout (see Deliverable D1.2 for full details).

21. **Requisiti**

Docker has to be installed on the machine.

22. **Installazione**

The following section outlines the steps required to deploy and run the different technologies and tools included in the system architecture using Docker containers.

EMQX

The host will pull the EMQX image from the repository using this command:

```
docker pull emqx
```

Then we create an ad-hoc network for sharing the container's settings in the same network:

```
docker network create --driver bridge influxdb-telegraf-net
```

We create two volumes for storing and persisting emqx data and log:

```
sudo docker volume create --name emqx-data
```

```
sudo docker volume create --name emqx-log
```

At this stage, we can start EMQX by giving a node name, the default port, the newly created volumes, the network to be used, and an alias for the image running on the docker container:

```
sudo docker run -d --name emqx-rasta -e "EMQX_NODE_NAME=emqx@rasta.emqx.com" -p 18083:18083 -p 1883:1883 -v emqx-data:/opt/emqx/data -v emqx-log:/opt/emqx/log --network influxdb-telegraf-net --network-alias rasta.emqx.com emqx:latest
```

For other nodes (for non-opensource version), we can use:

```
sudo docker run -d --name emqx-rasta -e "EMQX_NODE_NAME=emqx@rasta.emqx.com" -v emqx-data:/opt/emqx/data -v emqx-log:/opt/emqx/log --network influxdb-telegraf-net --network-alias rasta.emqx.com emqx:latest
```

then connect with the other node (rasta) in the same cluster:

```
docker exec -it emqx-rasta bash
```

```
emqx ctl cluster join emqx@changes.emqx.com
```

InfluxDB

In order to start the InfluxDB instance on the same machine, run this command:

```
docker run --name influxdb2 --publish 8086:8086 --mount type=volume,source=influxdb2-data,target=/var/lib/influxdb2 --mount type=volume,source=influxdb2-config,target=/etc/influxdb2 --env DOCKER_INFLUXDB_INIT_MODE=setup --env DOCKER_INFLUXDB_INIT_USERNAME=<influx-rasta.user> --env DOCKER_INFLUXDB_INIT_PASSWORD=<password> --env DOCKER_INFLUXDB_INIT_ORG=rasta --env DOCKER_INFLUXDB_INIT_BUCKET=rasta --network influxdb-telegraf-net --name influxdb influxdb:2
```

This command also ensures that the docker image is running in the same network so InfluxDb can communicate with EMQX and Grafana.

Telegraf

To run telegraf, a config file must be added as follows:

```
nano <path>/telegraf.conf
```

```
docker run -v <path>/telegraf.conf:/etc/telegraf/telegraf.conf:ro --network influxdb-telegraf-net --name telegraf --detach telegraf
```

In the config file, three variables must be specified related to the EMQX connection:

```
servers = ["tcp://rasta.emqx.com:1883"]
```

```
  username = <mqtt-user>
```

```
  password =<mqtt_password>
```

Moreover, if we need to extract additional tags received in MQTT dynamically, for instance, 'clientid' or ‘unit’ as tags from the JSON payload, we need to add the following:

```
  tag_keys = ["clientid"]

 ## Include specific fields

  fieldpass = ["unit"]
```

Grafana

The dashboard visualization component is implemented with Grafana. These architecture modules enable the visualization of dashboards for managing tourism and weather conditions in the POIs equipped with the sensors.

To run Grafana we can use a new volume in Docker by running the following command:

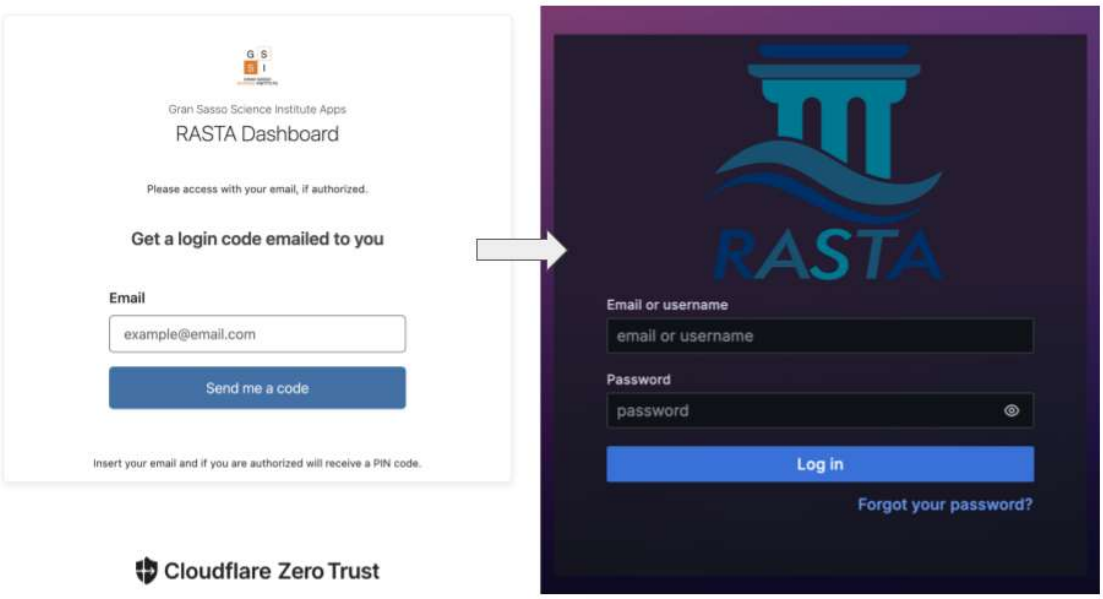
```
docker volume create grafana-storage
```

and then starting the container:

```
`docker run -d --name=grafana` --network influxdb-telegraf-net `--volume grafana-storage:/var/lib/grafana -p 80:3000 -e "GF_SERVER_DOMAIN=<server_url>" grafana/grafana`
```

23. **Usage**

To manage the authorized users on CloudFlare to grant access to the RASTA dashboards. Basically, each request routed by the DNS is first checked for authorization and authentication on CloudFlare, if authorized, it is routed to the Azure VM public address, where the basic authorization provided by our systems is also checked. We used the one-time option to grant access to the users, who were grouped into teams. The screenshot below shows how CloudFlare simplified the user management, reinforcing the security of the applications:



24. **Configurazione**

We using the docker to run telegraf, a config file must be added as follows:

```
nano <path>/telegraf.conf
```

```
docker run -v <path>/telegraf.conf:/etc/telegraf/telegraf.conf:ro
--network influxdb-telegraf-net --name  telegraf --detach telegraf
```

In the config file, three variables must be specified related to the EMQX connection:

```
servers = ["tcp://rasta.emqx.com:1883"]
```

```
  username = <mqtt-user>
```

```
  password =<mqtt_password>
```

Moreover, if we need to extract additional tags received in MQTT dynamically, for instance, 'clientid' or ‘unit’ as tags from the JSON payload, we need to add the following:

```
  tag_keys = ["clientid"]
```

```
 ## Include specific fields
```

```
  fieldpass = ["unit"]
```

25. **Contatti**

The following members from the Gran Sasso Science Institute (GSSI) are involved in the RASTA POI Management System project:

Sadia Azam, Maria Giovanna Brandano, Gianlorenzo D'Angelo, Martina De Sanctis, Amleto Di Salle, Paola Inverardi, Ludovico Iovino, Rickson Pereira, Claudio Pompilio, Franco Raimondi

For any issues, technical support, or inquiries, please contact the team. Email addresses are available on the official GSSI website.