score

D8.4 – Early Warning and Digital Twin Platform prototype

DATE OF DELIVERY - 31/08/2023
AUTHORS – Andrea Rucci (MBI), Ivan Boscaino (MBI), Giovanni Serafino (MBI)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003534

DOCUMENT TRACKS DETAILS

Project acronym	SCORE
Project title	Smart Control of the Climate Resilience in European Coastal Cities
Starting date	01.07.2021
Duration	48 months
Call identifier	H2020-LC-CLA-2020-2
Grant Agreement No	101003534

Deliverable Information	
Deliverable number	D8.4
Work package number	WP8
Deliverable title	Early Warning and Digital Twin Platform prototype
Lead beneficiary	MBI srl
Author(s)	Andrea Rucci (MBI), Ivan Boscaino (MBI), Giovanni Serafino (MBI)
Due date	31/08/2023
Actual submission date	31/08/2023
Type of deliverable	Demonstrator
Dissemination level	Public

VERSION MANAGEMENT

Revision table			
Version	Name	Date	Description
V 0.1	Giovanni Serafino (MBI), Ivan Boscaino (MBI), Andrea Rucci (MBI)	10/08/2023	First draft
V 0.2	José Pedro Matos (UST-ID), Filippo Giannetti (UNIPI)	24/08/2023	Updated draft internally reviewed
V 0.3	Giovanni Serafino (MBI), Ivan Boscaino (MBI), Andrea Rucci (MBI)	28/08/2023	Updated draft after contribution from partners
V1.0	Giovanni Serafino (MBI) Iulia Anton (ATU) Salem Gharbia (ATU)	28/08/2023	Final version

LIST OF ACRONYMS AND ABBREVIATIONS

Acronym / Abbreviation	Meaning / Full text
API	Application Programming Interface
CCLL	Coastal City Living Lab
CRUD	Create, read, update, and delete
DSM	Digital Surface Model
DT	Digital Twin
EBA	Ecosystem-Based Approach
EWS	Early-Warning Support
GIS	Geographic Information System
GUI	Graphic User Interface
ICT	Information and Communication Technologies
JSON	Javascript Object Notation
REST	Representational State Transfer
RPO	Regional Planning Organization
SIP	SCORE Information and communication technologies
SME	Small or Medium Enterprise
SMEs	Small-medium Enterprise
URLs	Unique Resource Locator
USE	User Scenario Evaluation
WP	Work Package

BACKGROUND: ABOUT THE SCORE PROJECT

SCORE is a four-year EU-funded project aiming to increase climate resilience in European coastal cities.

The intensification of extreme weather events, coastal erosion and sea-level rise are major challenges to be urgently addressed by European coastal cities. The science behind these disruptive phenomena is complex, and advancing climate resilience requires progress in data acquisition, forecasting, and understanding of the potential risks and impacts for real-scenario interventions. The Ecosystem-Based Approach (EBA) supported by smart technologies has potential to increase climate resilience of European coastal cities; however, it is not yet adequately understood and coordinated at European level.

SCORE outlines a co-creation strategy, developed via a network of 10 coastal city 'living labs' (CCLLs), to rapidly, equitably and sustainably enhance coastal city climate resilience through EBAs and sophisticated digital technologies.

The 10 coastal city living labs involved in the project are: Sligo and Dublin, Ireland; Barcelona/Vilanova i la Geltrú, Benidorm and Basque Country, Spain; Oeiras, Portugal; Massa, Italy; Piran, Slovenia; Gdansk, Poland; Samsun, Turkey.

SCORE will establish an integrated coastal zone management framework for strengthening EBA and smart coastal city policies, creating European leadership in coastal city climate change adaptation in line with The Paris Agreement. It will provide innovative platforms to empower stakeholders' deployment of EBAs to increase climate resilience, business opportunities and financial sustainability of coastal cities.

The SCORE interdisciplinary team consists of 28 world-leading organisations from academia, local authorities, RPOs, and SMEs encompassing a wide range of skills including environmental science and policy, climate modelling, citizen and social science, data management, coastal management and engineering, security and technological aspects of smart sensing research.

EXECUTIVE SUMMARY

This document is a deliverable of the SCORE project, funded under the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003534.

The D8.4 represents the first release of the back-end part of the geographic information system (GIS)-Based Digital Twin – Early-Warning Support (DT-EWS) prototype, as a result of the activities in task T8.2 "Development of the Early Warning Support and Digital Twin Platform". Multiple versions of the deliverable will be produced during the project, since the first one will be refined at different stages to correct bugs and add new features, until the end of the project, based on the needs coming from the users and the outcomes of the deployment activities. Each release will be tagged with a unique version number.

The system is composed of two main subsystems, namely the User Scenario Evaluation (USE) and the Early Warning Support (EWS). The USE subsystem is meant to be employed for running simulations enabling what-if and long-term analyses on scenarios built by users as a mixture of real, ground-truth data from the coastal city, and hypothetical data, e.g., regarding weather events, sea state, and EBA solutions implemented within the urban study area. The EWS subsystem, on the other hand, is intended to run continuously without inputs from the users, since it has to monitor the current weather situation. By leveraging on real-time data from sensors distributed over the study area and on weather forecasts, it can make projections on the urban flood risk. This way, it can raise alerts in case the risk of potential floods in areas and/or implies water levels that can represent a risk for the population, or cause economic damages. The complete description of the system architecture and functionalities are included in the report D8.5 [1].

The present document outlines the structure of the software composing the two subsystems of the Platform, showing the general software framework, the directory structure, and listing the application programming interfaces (APIs) under development for interfacing with the graphical user interface (GUI). It is worth noticing that the software of the presented system is not open. Accordingly, it is not disclosed in this document.

LINKS WITH OTHER PROJECT ACTIVITIES

This document takes inputs that drove the DT-EWS system design from the SCORE project deliverables D8.1 - GIS Based Digital Twin Platform functional requirements [2], D8.2 - GIS Based Early Warning and Digital Twin Platform system architecture and design [3], and D8.3 - GIS-Based Early Warning and Digital Twin Platform Interface Control Document [4]. It is also based on the downscaled models elaborated in SCORE project WP3, the SCORE Information and communication technologies (ICT) platform (SIP) developed in SCORE project WP5, the exposure and vulnerability maps provided by SCORE project WP6, and the list of EBAs specified in SCORE project WP7.

Many interactions between T8.2 and T8.3 "Development of tools to access and visualise SCORE data and outcomes" also took place, since the design of the back-end system and of its GUI, at least from the functional point of view, did influence each other and could not be carried out in a completely separated way, although the actual complete integration between them is the focus of the activities of T8.4.

The outputs of this document are relevant for the SCORE WP8 system further development, integration, deployment, (project tasks T8.4), and for the subsequent activities of system validation and assessment, whose plan is reported in detail in D8.10 [4] (project task T8.5).

TABLE OF CONTENT

1. Introduction 7
1.1 Scope of the deliverable _____ 7
1.2 Structure of the deliverable _____ 7
2. The USE SUBsystem 8
2.1 Subsystem Directories Structure _____ 8
2.2 Subsystem Services _____ 9
3. The EWS Subsystem 10
3.1 Main Scripts _____ 11
3.2 The EWS Library _____ 11
3.3 The Risk Library _____ 12
3.4 Data Structures _____ 12
3.5 Maps Format _____ 14
4. Conclusions 14
5. Appendix A 16
6. References 17

INDEX OF FIGURES

Figure 1. List of the APIs available to the front-end to call back-end functions. 8
Figure 2. JSON Schemas employed by the REST APIs. 9

1. INTRODUCTION

As outlined in detail in D8.5 [1], the DT-EWS system is composed of the USE and the EWS subsystems. The operations of the former are largely driven by the user, who must create scenarios on which simulations are based. The latter, on the other hand, is intended to run continuous simulations on the actual hydraulic situation and its projections in the short term, taking into account real-time information coming from sensor data streams and weather forecasts. The reader can refer to [1] for more details about the system architecture and functionalities.

Within the SCORE project, the DT-EWS subsystem is to be deployed in the WP8 frontrunner CCLLs Massa (IT), Vilanova i la Geltrú (ES), and Oarsoaldea (ES). However, it can be interesting testing it also in other scenarios, depending on the available computational resources. At the present stage, the DT-EWS allows simulations by different CCLLs at the same time, but only one user per each CCLL at the time. However, additional computational resources will be available and, since the USE is realised as a Django project, its number of parallel runs can be easily scaled, just adding new computational resources. The EWS, on the other hand, does not represent scalability issues, as better explained at the beginning of Sect. 3.

1.1 Scope of the deliverable

The purpose of D8.5 is to provide a demonstrator of the first DT-EWS back-end system complete prototype, as the result of activities of Task T8.2. At the moment, this back-end part is partially accessible, since MBI srl, which is the leader of work package (WP) 8, and responsible of the implementation of this system, cannot completely disclose the implemented code. The integration of the back-end with the front-end part, i.e. with the GUI, whose development is part of the task T8.3 activities and is described in [5] and [6], is part of the ongoing task T8.4, started in M24. Therefore, a system version equipped with the GUI will be realised in the coming weeks. With that development, non-technical users will be able to operate with the system.

Here, we report the high-level structure of the code that underpins the system and the employed APIs that have been implemented up to now, that will be key tools for the back-end and front-end integration. In the following, the reader will find the URL where the APIs and their documentation are available.

1.2 Structure of the deliverable

The document is divided in two sections, each dedicated to one specific subsystem of the DT-EWS. The directories structure of the USE and of the EWS subsystems are briefly outlined, indicating the role of each element in the whole system, which are the main scripts and the employed libraries. Moreover, the APIs for the USE subsystem that have been listed in [1], along with their functions, are included; on the other hand, the APIs exploited by the EWS subsystem to interface with sensors for getting real-time data streams, and with other system modules are mentioned but not showed, since they are internal to the system.

For the complete description of the system architecture, please refer to [1].

2. THE USE SUBSYSTEM

2.1 Subsystem Directories Structure

The project is based on the Django high-level Python web framework, and it is structured like a typical Django project [2]:

```
└─── SCORE_DT_API
    └─── api
    └─── app
    └─── manage.py
    └─── mysite
    └─── requirements.txt
```

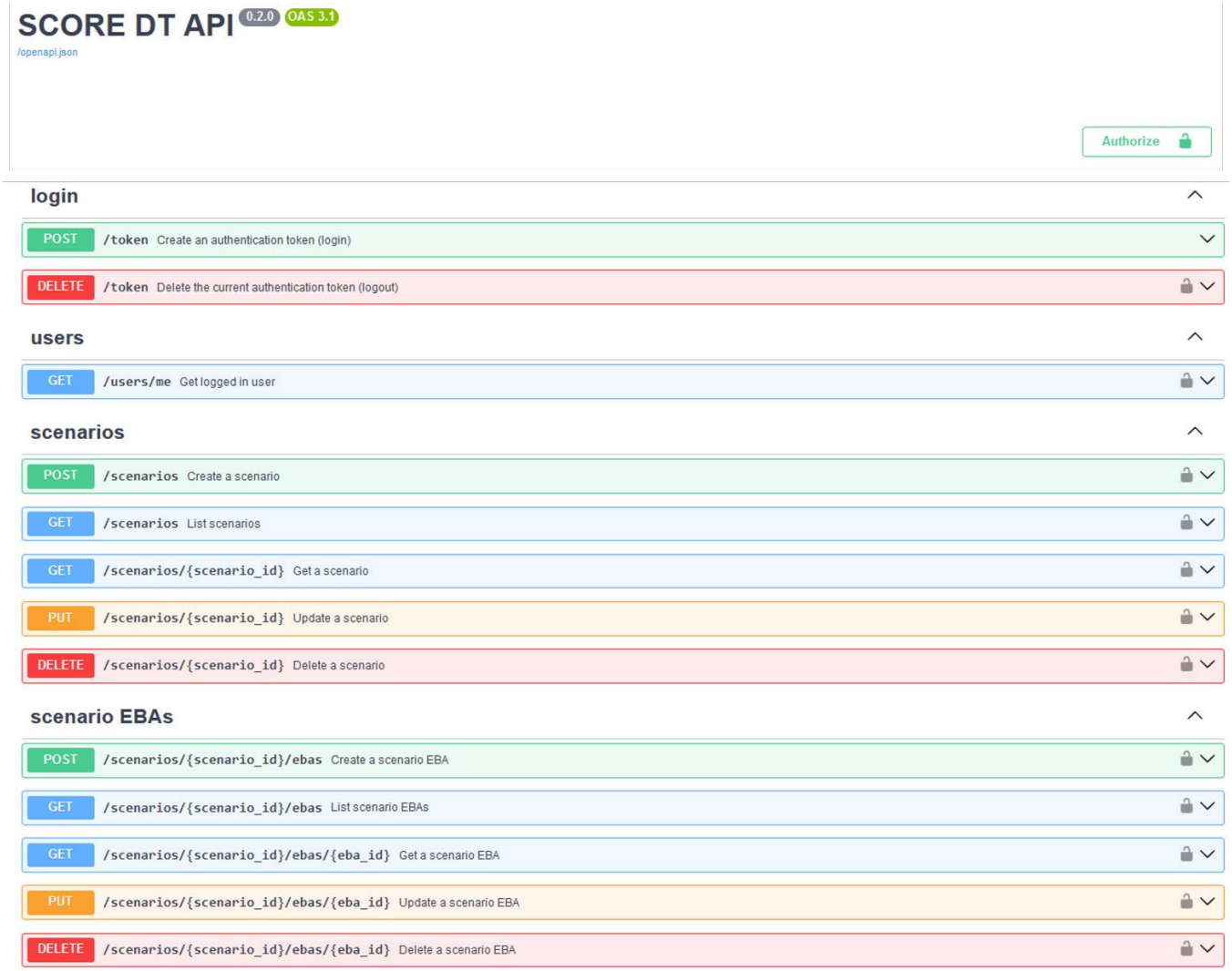

Figure 1. List of the APIs available to the front-end to call back-end functions.

Django is the high-level Python web framework employed for the development of the USE subsystem because it allows creating REST-APIs in a very flexible and scalable way, and smoothly interacting with the database. The EWS

subsystem, on the other hand, has been developed as an independent module that can run autonomously, and that has been directly realised in Python.

The main directory contains the classic Django configuration files and sub-directories: `manage.py`, `mysite/` and `requirements.txt` used to list the additional Python modules needed by the project.

In addition to the usual Django files, the `app/` directory contains the `models.py` module that defines all the database models (tables) used by the APIs for authentication, operations on scenarios, and simulations management, as described in [1].

The `api/` directory:

```
└── api
    └── main.py
    └── schemas.py
```

contains the definition and implementation of all API endpoints and related input and output Javascript object notation (JSON) schemas used for automatic validation and documentation. The employed schemas are listed at the URL reported in the following section, and are listed in Figure 2.

Figure 2. JSON Schemas employed by the REST APIs.

2.2 Subsystem Services

The USE module consists of the following services:

- The representational state transfer (REST) API,
- The Database (DB),
- The task manager.

The REST API service is documented and testable for developers at https://score_dt_api.mbigroup.it/docs. Figure 1 shows the web page with the list that can be found at the URL with the related create, read, update, and delete (CRUD) operations.

The technical URL for programmers is https://score_dt_api.mbigroup.it from which, after authentication, APIs can be used, based on the documentation contained in the previous URL. In Appendix A, a brief example of Python code for using the APIs is reported.

3. THE EWS SUBSYSTEM

The EWS subsystem has been developed in python 3.10. The project is structured in two separate Python packages: the EWS itself and the Risk package. The former contains the code and the scripts which are specific for the EWS functions, while the latter, called by the EWS processes, is a separate library devoted to the management of vulnerability and exposures (see WP6 work in D6.3 [8]), and to the computation of the associated risks.

The general structure is the following:

```
└── EWSS
    └── bin
    └── data
    |   └── common
    |   └── test
    |   └── massa
    |   └── oarsoaldea
    |   └── ...
    └── lib
    |   └── EWS
    |   └── Risk
    |   └── Utils
    └── scripts
```

The bin/ directory contains the runner scripts for the EWS and the general package setup, calling the libraries in lib/, which are specific for the EWS, Risk and other shared utility functions. The data/ directory is instead devoted to the specific CCLLs on which the system is operating. Along with common data, to each CCLL is dedicated a specific sub-directory in which exposure maps [8], reference scenarios, test files and warning results are stored. A scripts/ directory is also present and contains general scripts in bash or python that are useful for data management.

A peculiarity of the EWS consists in running simulations avoiding the use of LISFLOOD-FP [9], the digital hydraulic model, thanks to AI/ML algorithms, whose functions are extensively reported in D8.5 [1]. This implies very fast operations and a hugely reduced need for computational resources, compared to the USE subsystem. For this reason, in this stage of the development, where few realizations of the EWS are required, there is no need for a particular software architecture to guarantee parallel runs of the EWS on the same machine. However,

in the perspective of a better scalability of the system, future EWS versions may include, as in the USE subsystem, the Celery Python library for task management, with RabbitMQ as message-broker software, that will allow transparently parallel runs on different machines.

3.1 Main Scripts

The `bin/` directory contains two runners called `compute_risk.py` and `ews.py`. The first is responsible for computing the damage from a given flooding layer using a specific exposure element and map. The latter, instead, is the actual runner of the EWS, whose instance is expected to run separately and specifically for each CCLL in background and with regular time basis.

```
└── bin
|   └── compute_risk.py
|   └── ews.py
|   └── setup.py
|   └── README.md
└── ...
```

The EWS runner relies on a single configuration file in which all the necessary parameters are defined (see D8.5 [1] for details).

3.2 The EWS Library

The EWS library contains all the code and scripts related to the actual Early Warning Support functionality of the system. The `processing/` sub-directory is devoted to the definition of the abstract algorithms used in the data processing, while scenarios, defined as abstract objects implementing specific operations, are defined in `scenarios/`. The main class is defined in the `ews.py` file and is responsible for the whole run process and depends on the configuration whose template structure is defined in `config.py` (more details in [1]).

```
└── EWS
    └── __init__.py
    └── classifier.py
    └── config.py
    └── datastream.py
    └── ews.py
    └── preset.py
    └── validator.py
    └── scenario
    |   └── __init__.py
        |   └── scenario.py
```

```
│   └── operations.py
└── processing
    ├── __init__.py
    ├── aggregation.py
    ├── cluster.py
    ├── imputation.py
    ├── outlier.py
    └── oversample.py
```

Sensor and forecast datastreams are connected to the EWS by specific classes defined in `datastream.py` and rely on the same APIs used by the DT-EWS, connected to the activities of WP5 regarding the interfaces with sensors.

3.3 The Risk Library

The `Risk/` library is devoted to the computation of the damages on a CCLL due to the flooding on different exposure elements. Vulnerability functions defined in [8] are implemented as abstract classes which can be passed to a risk object able to compute the damage on a given exposure map due to a specific hazard.

```
└── Risk
    ├── __init__.py
    ├── definitions.py
    ├── predefined.py
    ├── risk.py
    ├── vulnerability.py
    └── vulnerability
        ├── vulnerability_functions_buildings.csv
        ├── vulnerability_functions_roads.csv
        └── vulnerability_functions_railways.csv
```

Specific vulnerability functions, which are common for all the CCLLs, are tabulated into `.csv` files in the `vulnerability/` sub-directory, while the abstract objects are defined in the `risk.py` and `vulnerability.py` files. Exposure maps are instead placed into the data directory which will be described in the next subsections.

3.4 Data Structures

Data used by the whole package, along with the results of each run are stored in the `data/` directory which possesses the structure shown below:

```
└── data
```

```
└── common
└── massa
|   └── config.yml
|   └── dsm.asc
|   └── exposure
|   |   └── buildings_potential.geojson
|   |   └── buildings_total.geojson
|   |   └── population.geojson
|   |   └── roads.geojson
|   └── scenarios
|   |   └── scenario_1
|   |   |   └── layers
|   |   |   |   └── res_0.asc
|   |   |   |   └── res_1800.asc
|   |   |   |   └── ...
|   |   |   └── rain.csv
|   |   |   └── sea.csv
|   |   |   └── river_1.csv
|   |   └── ...
|   └── tests
|   └── warnings
└── oarsoaldea
|   └── ...
└── ...
```

For each CCLL, a specific directory is defined. Each directory contains the configuration file `config.yml`, defining all the parameters needed by the EWS and the digital surface model (DSM), stored as raster files in compliance with those of the city used by LISFLOOD-FP [9]. In the `exposure/` sub-directory there are the vector layers defining the specific exposure of the cities for the different elements, i.e., population, buildings, roads, or railways, that will be used by the Risk package called by the EWS. The `scenarios/` sub-directory, instead, contains the set of historical or synthetic scenarios used in the EWS classification process. For each scenario, a set of `.csv` files with the temporal series associated to each relevant object is present, each one associated to a different input of the digital hydraulic model [9]. Results of the simulations obtained using this model are placed in the `layers/` sub-directory which, therefore, contains the set of all the flooding layers related to the scenarios during the time-evolution, with time in seconds embedded in the file name.

Results of the EWS are instead stored in the `warnings/` directory, with files reporting the run time to keep track of the different background runs. An example of single CCLL run result is the following:

```
└── data
```

```
└── massa
|   └── warnings
|   |   └── 2023-08-03 15:50:00+00:00_config.yml
|   |   └── 2023-08-03 15:50:00+00:00_log
|   |   └── 2023-08-03 15:50:00+00:00_buildings_total.geojson
|   |   └── 2023-08-03 15:50:00+00:00_population.geojson
|   |   └── 2023-08-03 15:50:00+00:00_roads.geojson
|   |   └── 2023-08-03 15:50:00+00:00_roads.geojson
|   |   └── 2023-08-03 15:50:00+00:00_res.geotiff
|   └── ...
└── ...
```

Results of the damage computation are exported as GeoJSON files, while the flooding map in a GeoTiff [10] raster. A copy of the configuration file and the logs of the run are always kept in order to check for possible problems or details about past runs.

3.5 Maps Format

Raster and vector geospatial data are internally converted and stored in EPSG:4326 [11] format projections. Scenarios flooding layers are stored as raster files in AAIGrid [12], a format compliant with LISFLOOD-FP [9], whereas the results are converted in GeoTIFF [10] which guarantee more versatility. Exposure maps are given in [8] as vector files; in the EWS subsystem, these are converted and stored internally as GeoJSON files [13]. Possible variations could be done in the next releases, if needed in the process of integration with the front-end, or if explicitly requested by the final users.

4. CONCLUSIONS

This document reports on the structure of the SCORE DT-EWS back-end system prototype, that will be used for simulating scenarios assembled by the final users, thanks to the USE subsystem, and to monitor and predict possible risks due to weather events and river discharge, thanks to the EWS subsystem. The scheme of the directories, with a brief explication of the software structure, is illustrated. However, since this deliverable is public, the implemented code is not disclosed.

At the moment, the finalization of the development of the REST APIs for a functional integration of the DT-EWS back-end (developed in task T8.2) and front-end (developed in task T8.3) is undergoing in task T8.4, started in M24. Interoperability and functional tests will be performed in order to ensure that defined functionalities specified in the previous WP8 deliverables are provided as expected [3] - [5]. The results of the integration activities and related validation procedures will be reported in the deliverable D8.8 “GIS-Based Early Warning and Digital Twin Platform integration report”, due at M30. The integration will also aim to ensure information flows between the complete integrated system and the modules developed in the other WPs are respected. This activity aims also to verify that the data processed by the DT-EWS is returned to stakeholders as needed. This goal will be pursued thanks to

meetings, training and feedback sessions with the end users and stakeholders after back-end and GUI integration. Thanks to the GUI, it will be possible also for non-technical users (i.e., people not familiar with programming) to explore and operate with the complete system functionality.

Once the system back-end and front-end parts integration is complete, the task T8.4 will move to the deployment of the integrated solution in the frontrunner coastal cities, with a particular focus on system customization to the specific CCLL needs.

5. APPENDIX A

Here we report, as an example, a piece of Python code with some instructions that a developer can use to test the APIs listed in this document and linked in Section 2.2.

```
import requests


url = "https://score_dt_api.mbigroup.it"


# ------------------------------------------------------------------------
# Get a token for the user
# ------------------------------------------------------------------------
data = {"username": "alice", "password": "test"}
r = requests.post(f"{url}/token", data=data)
r.raise_for_status()
response = r.json()
token = response["access_token"]


# ------------------------------------------------------------------------
# Prepares the headers for the next requests
# ------------------------------------------------------------------------
headers = {"Authorization": f"Bearer {token}"}
# ------------------------------------------------------------------------
# Create a scenario
# ------------------------------------------------------------------------
data = {"name": "scenario 1", "description": "scenario 1 description"}
r = requests.post(f"{url}/scenarios", json=data, headers=headers)
r.raise_for_status()
scenario = r.json()


# ------------------------------------------------------------------------
# Read a scenario
# ------------------------------------------------------------------------
r = requests.get(f"{url}/scenarios/{scenario['id']}", headers=headers)
r.raise_for_status()
response = r.json()
print(response)
```

6. REFERENCES

[1] SCORE D8.5 – Early Warning and Digital Twin Platform release notes and user manual

[2] https://docs.djangoproject.com/en/4.2/ (last accessed 21/08/2023)

[3] SCORE D8.1 - GIS Based Digital Twin Platform functional requirements

[4] SCORE D8.2 - GIS Based Early Warning and Digital Twin Platform system architecture and design

[5] SCORE D8.3 - GIS-Based Early Warning and Digital Twin Platform Interface Control Document

[6] SCORE D8.6 – Tools to access and visualise SCORE data and outcomes

[7] SCORE D8.7 – Tools to access and visualise SCORE data and outcomes release notes

[8] SCORE D6.3 – Exposure database and vulnerability curves for the frontrunners CCLLs

[9] https://www.seamlesswave.com/LISFLOOD8.0.html (last accessed 20/08/2023)

[10] https://www.ogc.org/standard/geotiff/ (last accessed 21/08/2023)

[11] https://epsg.io/4326 (last accessed 21/08/2023)

[12] https://gdal.org/drivers/raster/aaigrid.html#raster-aaigrid (last accessed 21/08/2023)

[13] https://geojson.org/ (last accessed 21/08/2023)