D3.2 Installation Report

Deliverable type	R – Document, report
Dissemination level	PU - Public
Due date (month)	M13
Delivery submission date	29 February 2024
Work package number	WP3
Lead beneficiary	IOTIQ GmbH

This project has received funding from the Horizon Europe Framework Programme of the European Union under grant agreement No. 101094428

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them.

Document Information

Project number	101094428	Acronym	CULTURATI
Project name	Customized Games and Routes For Cultural Heritage and Arts		
Call	HORIZON-CL2-2022-HERITAGE-01		
Topic	HORIZON-CL2-2022-HERITAGE-01-02		
Type of action	HORIZON-RIA		
Project starting date	1 February 2023	Project duration	36 months
Project URL	http://www.culturati.eu		
Document URL	https://culturati.eu/deliverables/		

Deliverable number	D3.2			
Deliverable name	Installation Report			
Workpackage number	WP3			
Work package name	System Testing and Verification			
Date of delivery	Contractual	M13	Actual	M13
Version	Version 1.0			
Lead beneficiary	IOTIQ GmbH			
Responsible author(s)	Dr. Metin Tekkalmaz, IOTIQ GmbH, metin@iotiq.com			
Reviewer(s)	Neşe Şahin Özçelik, Bilkent Universitesi Vakif, nozcelik@bilkent.edu.tr Eda Gürel, Bilkent Universitesi Vakif, eda@tourism.bilkent.edu.tr			

Short Description	This report offers a comprehensive analysis of the site-specific system installation for the CULTURATI platform, detailing the settings utilized and issues encountered during the process. Each setting is accompanied by a rationale, providing insight into the decision-making process, while each issue is discussed in terms of its impact and actionable suggestions for resolution. Through this thorough examination, the document aims to provide valuable insights into the installation process, facilitating informed decision-making and smoother implementation of the CULTURATI platform across diverse cultural sites.

History of Changes

Date	Version	Author	Remarks
13 March 2024	Draft 0.1	Metin Tekkalmaz	First version
15 March 2024	Version 0.1	Metin Tekkalmaz	Revised after Review
25 March 2024	Version 0.2	Metin Tekkalmaz	Revised after Review
29 March 2024	Version 1.0	Metin Tekkalmaz	Revised after Review

Executive Summary

This report documents the successful installation of various components for the CULTURATI platform and system. It outlines the implementation of strategically placed sensors in the Ankara Citadel to monitor crowd levels, along with the setup of two servers in Germany for production and testing purposes. The authentication server, Keycloak, is undergoing pilot testing on the staging server before full deployment. Additionally, the CULTURATI Main Application and CULTURATI Wiki Application followed standard development processes, with plans for deployment upon successful testing. Notably, the CULTURATI Wiki will host separate instances for the Ankara Citadel and İstanbul Rahmi M. Koç Museum. The report also addresses site-specific settings and encountered issues, providing rationales for settings and suggestions for issue resolution. Overall, this comprehensive approach ensures the successful installation of the CULTURATI platform and sets the stage for its deployment across other pilot sites in Europe.

Table of Contents

1. Introduction

This report serves as a comprehensive documentation of the successful installation of various components for the CULTURATI platform and system. It encapsulates our journey in enhancing crowd monitoring capabilities within the Ankara Citadel, showcasing the strategic deployment of sensors to provide real-time data for analysis. Additionally, the establishment of two servers in Germany, dedicated to production and testing (staging) respectively, ensures the seamless operation of CULTURATI web applications.

Our primary focus lies in meticulously crafting a robust infrastructure to support the objectives of the CULTURATI platform. Keycloak, our authentication server, is currently undergoing pilot testing on the staging server to guarantee its seamless integration before full deployment. Moreover, the development process strictly adheres to standard procedures, with the CULTURATI Main Application and CULTURATI Wiki Application undergoing rigorous testing phases before deployment on the production server.

A notable feature of our implementation strategy is the customization of the CULTURATI Wiki to cater to the specific needs of cultural sites, exemplified by its tailored instances for the Ankara Citadel and Istanbul Rahmi M. Koç Museum. This meticulous approach ensures that our platform is finely tuned to meet the diverse requirements of cultural heritage management.

Furthermore, the procedures outlined in this report are pivotal not only for the successful installation in Ankara but also serve as a blueprint for deployment across other pilot sites in Europe. By providing detailed accounts of site-specific settings and discussions on encountered issues, we aim to facilitate a thorough understanding of the implementation process, thereby ensuring a consistent and reliable platform for efficient crowd management and user experience enhancement across multiple cultural sites.

In the subsequent sections, we will provide more information about our methodology, challenges faced, and solutions devised in our endeavor to revolutionize cultural site management with the CULTURATI platform.

2. Sensor Installation and Integration

The Ankara Governorship has acquired sensors for crowd detection within the Ankara Citadel area. These sensors have been strategically installed at predefined locations to effectively monitor crowd

levels. The data collected from these sensors is easily accessible to our technical team and is ready for seamless integration into the CULTURATI applications. For detailed information regarding the sensors installed at locations specified by the Ankara Governorship, including their installation process and technical specifications for ongoing support, please refer to Appendix A.

3. Server Installation

Two servers designated for the CULTURATI Web Applications have been successfully installed. These servers, operating under the Ubuntu 22.04 system, are located in Germany. One server is designated as the production server, while the other serves as the staging server. Initial deployments will occur on the staging server to facilitate comprehensive testing. Upon successful testing on the staging server, final deployments will be carried out on the production server.

3.1. CULTURATI Keycloak Server

3.1.1. General Information

Keycloak, the authentication server, has been successfully installed on the CULTURATI Staging server. Once pilot tests on the staging server are successfully completed, Keycloak will be deployed on the CULTURATI Production servers.

3.1.2. Installation Instructions

Server Requirements:

The server must have installed the following programs

- Docker version 24.0.7, build afdd53b
- Docker-compose version 1.29.2, build 5becea4c

Required docker-compose.yml file can be found in Appendix B. One should run “docker compose” in the folder containing this file:

- *docker compose up –d*

This command will execute all the components needed by the CULTURATI backend application including the database server itself.

4. Application

4.1. CULTURATI Main Application

4.1.1. General Information

In the typical feature development process, the CULTURATI Main Application follows a sequential installation path on the specified servers:

1. IOTIQ Development Server.
2. CULTURATI Staging Server.
3. CULTURATI Production Server.

During the initial installation of the CULTURATI Main Application, the process was executed accordingly, with the application being installed on the development server and staging server. Following successful pilot tests, the final installation on the production server will be conducted.

4.1.2. Installation Instructions

CULTURATI application includes backend and frontend applications. Backend application is delivered as a docker image which will be installed using docker. Frontend application is delivered as a folder which should be placed at a specific location at the server.

Server Requirements:

The server must have installed the following programs

- Docker version 24.0.7, build afdd53b
- Docker-compose version 1.29.2, build 5becea4c

Backend application:

Required docker-compose.yml file can be found in Appendix C. One should run “docker compose” in the folder containing this file:

- *docker compose up –d*

This command will execute all the components needed by the CULTURATI backend application including the database server itself.

Frontend Application:

Frontend application is delivered as a folder which should be placed under `/var/www/culturati/culturati-admin-frontend/dist` folder for admin application and `/var/www/culturati/culturati-user-frontend/dist` folder for visitor application.

4.1.3. Configuring Additional Sites/Institutions

4.1.3.1. Keycloak Configuration

For adding a new site/institution to the application, a few configurations should be made in keycloak:

Adding a new realm: Adding a new realm on keycloak is an easy task which is explained in detail in keycloak official documentation (https://www.keycloak.org/docs/latest/server_admin/#configuring-realms):

- Point to the top of the left pane.
- Click Create Realm
- Enter a name for the realm. (institution name.)
- Click Create

Adding a new client in the realm: Adding a new client on keycloak is an easy task which is explained in the official keycloak documentation (https://www.keycloak.org/docs/latest/server_admin/#proc-creating-oidc-client_server_administration_guide)

- Select the realm which is created
- Click Clients in the menu.
- Click Create client
- Leave Client type set to OpenID Connect.
- Enter a Client ID. This ID is an alphanumeric string that is used in OIDC requests and in the Keycloak database to identify the client.
- Supply a Name for the client.
- Click Save.
- Navigate to Settings:
  - Set Root URL, Home URL, Valid post logout redirect URIs, Admin URL to: https://<institution-sub-domain>.admin.culturati.app
  - Set Valid redirect URIs to: https://<institution-sub-domain>.admin.culturati.app /*
  - Set Web origins to '+' .

4.1.3.2. Backend Application Configuration

A configuration file should be created for a new institution. After creating this file backend application/container should be restarted. For adding a new site/institution to the application:

- a configuration file should be added to the application folder. (under tenants directory.) The as name will be used also for the database name that will be created for this institution. The configuration is as follows:

  name=<name of the institution>

  datasource.url=jdbc:postgresql://culturati-db:5432/<name>

  datasource.username=<database-user-name>

  datasource.password=<database-password>

  datasource.driver-class-name=org.postgresql.Driver

  keycloak.url=<project-keycloak-url>

  keycloak.realm=<name> (this should match the name of the keycloak realm that is created for the institution)

  keycloak.clientId=<client-id> (the client-id for the client created in Keycloak.)

  map.center=<coordinate of the center of site map>

  map.boundingBox=<right top coordinate, left bottom coordinate>

  xwiki.username=<nimbeo api username>

  xwiki.password= <nimbeo api password>

  xwiki.baseUrl= <nimbeo api url for the institution>

4.2. CULTURATI Content Management Application (CULTURATI Wiki)

4.2.1. General Information

In the standard feature development process, the installation sequence for the Culturati Wiki Application is as follows:

1. Nimbeo Development Server.
2. CULTURATI Staging Server.
3. CULTURATI Production Server.

During the initial installation of the CULTURATI Wiki Application, the process unfolded by installing the application on the development server. Subsequently, two Wiki instances, catering to Ankara Citadel and RMK, will be installed on the staging server. Following successful pilot tests, the installation of the two instances will be extended to the production server.

4.2.2. Installation Instructions

Software prerequisites

The server must have installed the following programs:

- Docker version 24.0.7, build afdd53b
- Docker-compose version 1.29.2, build 5becea4c

Server settings

Minimum characteristics

- Operating system: Ubuntu 22.04.3 LTS x86_64
- Kernel: 5.15.0-25-generic
- Memory: 8 GB
- Processor: AMD EPYC 7282 (4) @ 2.794GHz
- Hard Disk Drive 800 GB SSD

Recommended features

- Operating system: Ubuntu 22.04.3 LTS x86_64
- Kernel: 5.15.0-25-generic
- Memory: 60 GB
- Processor: Intel Core I9 13900K 3.0 GHz
- Hard Disk: 800 GB SSD

Software prerequisites

The server must have installed the following programs:

- Docker version 24.0.7, build afdd53b
- Docker-compose version 1.29.2, build 5becea4c

Install these programs as follows:

1. Update the apt package index by using the next command:
   sudo apt update
2. Install the necessary packages for apt to use packages over HTTPS:
   sudo apt install apt-transport-https ca-certificates curl software-properties-common
3. Add the official Docker GPG key:
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
4. Add the Docker repository to the apt sources:

echo "deb [arch=$$(dpkg --print-architecture) signed by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

5. Update the apt package index again:

   sudo apt update

6. Install Docker:

   sudo apt install docker-ce docker-ce-cli containerd.io

Installing Docker Compose:

1. Install pip for Python:

   sudo apt install python3-pip

2. Install Docker Compose:

   sudo pip install docker-compose==1.29.2

   Or alternatively, using curl directly from GitHub:

   sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$$(uname -m)" -o /usr/local/bin/docker-compose

   sudo chmod +x /usr/local/bin/docker-compose

Wiki configuration

Wiki database configuration:

The developed Wiki uses a PostgreSQL database and is encapsulated in a Docker container, the configuration steps are as follows:

The development team delivers two components:

1. pgdata.zip: a compressed file with the database backup
2. xwiki-postgres.tar: Compressed file with the docker image

The above files are used as follows:

Unzip the pgdata.zip file and open a terminate and execute the next command :

docker load -i postgres-xwiki.tar

Finally, execute the command:

*docker run --net=xwiki-nw --name xwiki-dev-postgres *

-p 5434:5432 -v ${PWD}/pgdata/data:/var/lib/postgresql/data

\{{PWD}/pgdata/data:/var/lib/postgresql/data

-e POSTGRES_ROOT_PASSWORD=culturati23-pass \

-e POSTGRES_USER=culturati23-user \

-e POSTGRES_PASSWORD=culturati23-pass \

-e POSTGRES_DB=xwiki

-e POSTGRES_INITDB_ARGS="--encoding=UTF8" \

-d xwiki-dev-postgres-img:latest

With these steps the database is deployed and fully configured, you can test the access using a client such as DBeaver and the following data:

Server IP: As is natural this parameter will vary depending on which server the database is installed on, it can be obtained using the console command:

hostname -I

- Port: 5434
- User: culturati23-user
- Password: culturati23-pass
- Database: xwiki

Installing the Wiki application:

Please follow the next Docker commands for a xWiki Development Environment:

1. Commit Changes to XWiki Container command: *docker commit xwiki xwiki-dev-img*

   Description: Saves the current state of the 'xwiki' container as a new image named 'xwiki-dev-img'. Useful for capturing modifications for future use.
2. Save XWiki Image to a TAR File command: *docker save -o xwiki-dev-img.tar xwiki-dev-img*

   Description: Saves the 'xwiki-dev-img' Docker image to a TAR file. Enables transportation and loading into other Docker environments.
3. Create Directory for XWiki Data command: *mkdir data_wiki*

   Description: Creates a directory on your local filesystem for storing XWiki's persistent data.
4. Copy XWiki Data from Container command:

   docker cp xwiki:/usr/local/xwiki data_wiki

   Description: Copies XWiki data from the container to the local 'data_wiki' directory for backup or migration.
5. Commit Changes to PostgreSQL Container command:

   docker commit postgres-xwiki xwiki-dev-postgres-img

Description: Saves the current state of the 'postgres-xwiki' container as a new image with the database setup for XWiki.

6. Save PostgreSQL Image to a TAR File command:
   docker save -o xwiki-dev-postgres.tar xwiki-dev-postgres-img
   Description: Saves the 'xwiki-dev-postgres-img' Docker image to a TAR file for backup or distribution.
7. Create Directory for PostgreSQL Data command: *mkdir pgdata*
   Description: Creates a directory for storing PostgreSQL data persistently.
8. Copy PostgreSQL Data from Container command:
   docker cp postgres-xwiki:/var/lib/postgresql/data pgdata
   Description: Backs up or migrates PostgreSQL data by copying it from the container to the 'pgdata' directory.
9. Run PostgreSQL Container for XWiki Development command:
   docker run --net=xwiki-nw --name xwiki-dev-postgres -p 5434:5432 -v ${PWD}/pgdata/data:/var/lib/postgresql/data -e POSTGRES_ROOT_PASSWORD=culturati23-pass -e POSTGRES_USER=culturati23-user -e POSTGRES_PASSWORD=culturati23-pass -e POSTGRES_DB=xwiki -e POSTGRES_INITDB_ARGS="--encoding=UTF8" -d xwiki-dev-postgres-img:latest
   Description: Runs a new PostgreSQL container with configuration for XWiki development, linking to persisted data.
10. Load XWiki Image from TAR File command: *docker load -i xwiki-img.tar*
    Description: Loads an XWiki Docker image from a TAR file, useful for image transfer.
11. Execute the xWiki Development Container command:
    docker run --net=xwiki-nw --name xwiki-dev --link xwiki-dev-postgres:xwiki-dev-postgres -p 8081:8080 -v ${PWD}/data_wiki/xwiki:/usr/local/xwiki -e DB_USER=culturati23-user -e DB_PASSWORD=culturati23-pass -e DB_DATABASE=xwiki -e DB_HOST=xwiki-dev-postgres -d xwiki-dev-img:latest

Description: Runs the XWiki development container, linking to the PostgreSQL container and setting up database connectivity.

Installing multiple Wiki for multiple institutions:

For each institution, it is necessary to install a separate instance of the wiki. This process can be carried out on a single server, using one of the following strategies to differentiate each installation:

Port Differentiation: A unique port is assigned to each wiki on the server. In this way, each instance is distinguished by its specific port configuration.

Use of Specific Domains: Each institution acquires its own domain and configures its DNS records to point to the server hosting its wiki. On the server, requests can be handled by Nginx, which acts as an intermediary, redirecting each request to the Docker container corresponding to the institution's wiki.

By selecting the method that best suits the user to differentiate the Wiki installations and following the steps mentioned above, you will have no problems to run the wiki correctly.

Important clarification: Since this installation is done through a Docker image, no dependency or specific version is required to perform the above steps, since the version management is inside the containers, making it unnecessary to install dependencies or specific software versions, just follow the mentioned steps above.

Safety measures into the Wiki

Assign password to a user:

Once the Wiki is installed, select the side menu and click on 'Manage Wiki', there you will see the 'Users & Rights' section and within this you will find the 'Users' menu.

Here you will find a list of all the users registered in the Wiki, if you click on any of them, you will be able to make configurations for each user, among them, change the password or modify the group to which the selected user belongs.

Note that the action of resetting passwords can only be done by the super user Wiki administrator, this by default is the user with which the installation steps mentioned in the previous section are done.

Adding a user to a group:

In the global configuration section (accessing from the side menu) you can see the section of groups and users within the Wiki:

By clicking on 'Groups' you will be able to create the group that the user considers necessary, although by default you will find the groups shown below:

The 'xWikiAdminGroup' group is the group that contains only users with advanced knowledge of the Wiki flow, users who are part of this group will have super user permissions on the Wiki, so it is a sensitive role that few people should have access to.

The 'xWikiAllGroup' group is the default group that is assigned to a user that has been created and no specific group has been specified at the time of its creation in the Wiki.

The other remaining groups (content-creators, content-editor, data-entry-operators) are the groups that were created to differentiate the roles between users who use Culturati, each of these groups can do different actions in the Wiki and each one of them has its own interface.

Assign a group to a user:

To assign a group to a user, simply go to the groups section mentioned in the previous steps, find the group to which you want to add the user and click 'Edit', in which you will see the following interface:

By clicking on the 'Users to Add' field, the list of all the users created in the Wiki will be displayed, select one or more of them and finally click the 'Add' button.

If, on the other hand, you want to remove a user from a specific group, repeat the previous step and when you see the user to be removed from the group, simply click on the 'remove' button that has a red arrow that can be seen in the previous image.

Backup strategies

If you want to generate a backup copy of all the data held by the Wiki, we have made this process easier only by following the following Docker commands:

1. *docker commit xwiki xwiki-img*
2. *docker save -o xwiki-img.tar xwiki-img*
3. *mkdir data_wiki*
4. *docker cp xwiki:/usr/local/xwiki data_wiki*
5. *docker commit postgres-xwiki postgres-xwiki-img*
6. *docker save -o postgres-xwiki.tar postgres-xwiki-img*

7. *docker exec -t postgres-xwiki pg_dumpall -c -U culturati23-user > backup_db.sql*
8. *mkdir pgdata*
9. *docker cp postgres-xwiki:/var/lib/postgresql/data pgdata*

Conclusion

This document provides a comprehensive overview of the installation process for the system of CULTURATI, focusing on crucial components such as sensor deployment, server infrastructure, and authentication setup. It details the completion of sensor deployment in the Ankara Citadel and the seamless integration of data into the system. Additionally, it outlines the meticulous setup of server infrastructure and authentication mechanisms, with rigorous testing underway to ensure a smooth launch of the CULTURATI applications. Thus, this document highlights the significant progress made in implementing the system of CULTURATI and sets the stage for its successful deployment.

Appendix A. PROJECT SENSOR INSTALLATION REPORT

THE REPUBLIC OF TÜRKİYE
ANKARA GOVERNORSHIP
CULTURATI PROJECT
SENSOR INSTALLATION REPORT

This report describes the sensors installed in the locations determined by the Governorship of Ankara within the scope of the "CULTURATI Project", their installation and technical conditions for technical support. For the CULTURATI Project, the following practices were carried out in order to procure and install the sensors within the system to be installed in the Ankara Citadel Region determined in the project application for the CULTURATI Project and to provide technical support to be provided during the project. The installations were completed between January 10 and January 12, 2024.

PROPERTIES AND QUANTITIES OF MATERIALS

RADAR SENSORS: A total of 9 (nine) Radar Sensors with the specifications listed below were installed in the specified locations and delivered in a trouble-free and operational condition.

Properties:

a) Does not take film, works with person counting technology, (mmWave Radar technology),
b) Counting bidirectional movements,
c) 5 adjustable independent person counting zones,
d) Working at a minimum range of 10 m, 120 degree viewing angle,
e) Supports MQTT protocol, Telegram/Email notifications,
f) 4G supported embedded sim card,
g) Operating between -20C and 70C,
h) Wi-Fi support,
i) Supports a minimum 16 GB microSD card for backup,
j) Working in all kinds of light and weather conditions,
k) IFTTT, Shelly, and Web Socket support.

INFRARED SENSORS: A total of 6 infrared sensors and 3 gateways for infrared with the specifications listed below were installed in the specified locations and delivered in trouble-free working condition.

Infrared Sensor Properties:

a) Working with the principle of infrared beam transmission,
b) Has an internal memory with a capacity of at least 2000 records,
c) Counting bidirectional (Entry-Exit) transitions,
d) Can connect to the gateway device at a maximum distance of 700 meters (Open Area),
e) Accuracy of 95% or more within a distance of 2 meters,
f) Having three infrared beams,
g) With sabotage function with audible alarm,
h) Battery operated,
i) Battery life up to 1 year,

j) With the manufacturer's software to be installed locally, the information can be retrieved to a local SQL database on demand.

Gateway Properties for Infrared Sensors:

a) Has an internal memory with a capacity of at least 2000 records,
b) Connected to sensors at a maximum distance of 700 meters (Open Space),
c) Minimum 30 sensors connected,
d) Ethernet Port with POE support or Wi-Fi support,
e) Has a USB port for configuration.

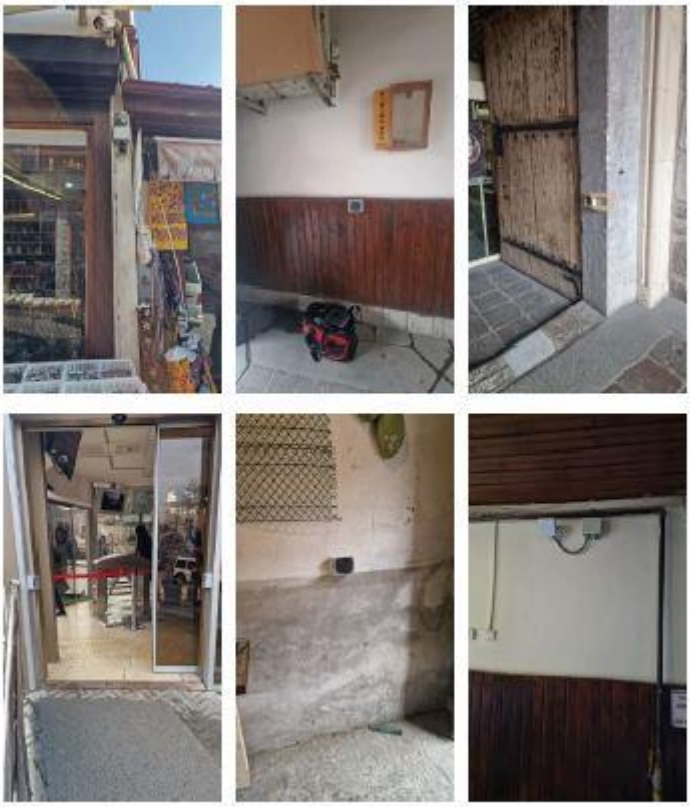
CLOUD SERVICE: For the cloud service of the sensors, users were defined on the interface and login information was given to Ankara Governorship staff and IOTIQ staff. The cloud system is planned to provide information to 3rd party software by allowing integration of sensor information via Application Programming Interface (API). The cloud servers will use the data from the sensors to provide the instantaneous number of people for each area to the users via the API. Cloud servers are planned to provide instantaneous person counting information and reports on a daily, weekly, or monthly basis through the web user interface. Instant information is displayed via Telegram. The contractor organization has committed that the UPTIME of the cloud system will not be below 98%.

WARRANTY AND TECHNICAL SUPPORT: The products offered have been delivered with 2 years warranty. Malfunctions excluding user errors will be eliminated during the warranty period. Technical support as needed during the project is 2 years. Maintenance and repair support will be provided once a year.

SENSOR MAP AND LAYOUT

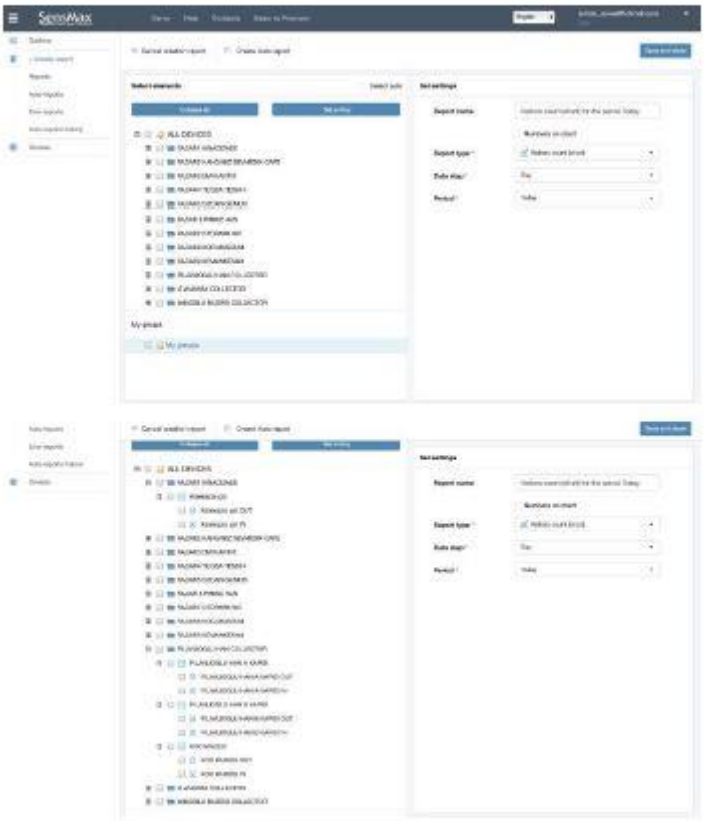
Sensor Numbers		Area Covered
RD9, INF2	A	PURPLE AREA
RD2, RD6, RD7, RD8, RD5	B	BLUE AREA
RD2, RD3	C	YELLOW AREA
INF3, INF4	D	GREEN AREA- PILAVLIOĞLU HAN
RD8, RD4	E	PINK AREA
INF5	F	PIRINÇ HAN
INF6	G	AHI ŞERAFETTIN MOSQUE
INF1	H	KOÇ MUSEUM
RD1, RD5	I	RED AREA
RD5	J	WHITE AREA

INSTALLATION SAMPLE PHOTOS




EMİN ANTİK

WEB INTERFACE AND TELEGRAM PAGE SAMPLE IMAGES




Ankara kale radar sensörler
NOTE:
DCC - Zone occupancy duration
AVG - Avg occupancy time per visitor
Koc muzesi
DAILY REPORT 01/03/2024 22:00:03
Gone 1
IN 1212
OUT 848
NOTE:
DCC - Zone occupancy duration
AVG - Avg occupancy time per visitor
Emin amik
DAILY REPORT 01/03/2024 22:00:03
Gone 1
IN 861
OUT 1389
NOTE:
DCC - Zone occupancy duration
AVG - Avg occupancy time per visitor
Kinacizade konagi
DAILY REPORT 01/03/2024 22:00:03
Kalekapisi giri
IN 570
OUT 662
NOTE:
DCC - Zone occupancy duration
AVG - Avg occupancy time per visitor

Appendix B. CULTURATI Keycloak docker-compose.yml file

```
version: "3.9"
services:
  keycloak:
    container_name: keycloak
    image: "keycloak/keycloak:22.0.5"
    restart: always
    hostname: culturati-keycloak
    ports:
     - 9080:8080
    volumes:
     - ./keycloak/config:/opt/keycloak/data/import
    environment:
     KEYCLOAK_ADMIN: admin
     KEYCLOAK_ADMIN_PASSWORD: <desicred admin password>
     KC_HOSTNAME: <desired keycloak hostname e.g.:keycloak.culturati.app>
     KC_HOSTNAME_STRICT: 'false'
     KC_HOSTNAME_STRICT_HTTPS: 'false'
     KC_PROXY: edge
     KC_DB_URL_HOST: postgres
     KC_DB_SCHEMA: public
     KC_DB_USERNAME: keycloak
     KC_DB_PASSWORD: <desired keycloak database password>
     KC_DB: postgres
    deploy:
     resources:
       limits:
         cpus: '1.0'
         memory: '2G'
    entrypoint: [ "/opt/keycloak/bin/kc.sh", "start-dev", "--import-realm" ]
  postgres:
    image: postgres:15.4-alpine3.18
    restart: always
    environment:
```

```
      POSTGRES_USER: keycloak
      POSTGRES_PASSWORD: <desired keycloak database password>
      POSTGRES_DB: keycloak
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./data:/docker-entrypoint-initdb.d
volumes:
  postgres_data:
      driver: local
```

Appendix C. CULTURATI Backend docker-compose.yml file

```
version: '3.1'
services:
  culturati-backend:
    image: iotiqdevops/culturati-backend:latest
    restart: unless-stopped
    container_name: culturati-backend
    depends_on:
      - culturati-db
    ports:
      - "8095:8080"
    environment:
      - spring.profiles.active=staging
      - defaultTenant=rmkm
      - app.jwt.secret=<any secret here>
      - seed.users.admin.password=<desired admin password>
    volumes:
      - ./tenants:/app/resources/tenants-staging
      - ./files:/files

  culturati-db:
    image: postgres:15.2-alpine
    restart: unless-stopped
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=rmkm
    volumes:
      - db-data:/var/lib/postgresql/data

  otel-collector:
    image: otel/opentelemetry-collector-contrib:0.89.0
    restart: always
    command:
```

```
    - --config=/etc/otelcol-contrib/otel-collector.yml
  volumes:
    - ./docker/collector/otel-collector.yml:/etc/otelcol-contrib/otel-collector.yml
  ports:
    - "8889:8889" # Prometheus exporter metrics
    - "13133:13133" # health_check extension
    - "4317:4317" # OTLP gRPC receiver
    - "4318:4318" # OTLP http receiver
  deploy:
    resources:
      limits:
        cpus: '1.0'
        memory: '1G'

prometheus:
  container_name: prometheus
  image: prom/prometheus:v2.48.0
  restart: always
  command:
    - --config.file=/etc/prometheus/prometheus.yml
  volumes:
    - ./docker/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
  ports:
    - "9090:9090"
  deploy:
    resources:
      limits:
        cpus: '1.0'
        memory: '1G'

loki:
  image: grafana/loki:2.9.2
  restart: always
  command: -config.file=/etc/loki/local-config.yaml
```

```
    ports:
      - "3100:3100"
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: '1G'

  tempo:
    image: grafana/tempo:2.3.0
    command: ["-config.file=/etc/tempo.yml"]
    volumes:
      - ./docker/tempo/tempo.yml:/etc/tempo.yml
      - tempo-data:/tmp/tempo
    ports:
      - "3200:3200"
      - "4317"
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: '1G'

  grafana:
    container_name: grafana
    restart: unless-stopped
    image: grafana/grafana:10.2.2
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=<grafana_admin_pass>
      - GF_SECURITY_ADMIN_USER=admin
    volumes:
      - ./docker/grafana/grafana-
datasources.yml:/etc/grafana/provisioning/datasources/datasources.yml
      - grafana-storage:/var/lib/grafana
```

```
    ports:
      - "3000:3000"
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: '1G'

volumes:
  db-data:
  tempo-data:
    driver: local
  grafana-storage:
    driver: local
```