Horizon 2020

COLL
ECTION
CARE

Innovative and affordable service for PC monitoring of individual Cultural Artefacts during display, storage, handling and transport

CollectionCare database storage II (Historic ambient data of artworks uploaded)

Deliverable number: D3.6

Version 1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 814624.

Project Acronym:	CollectionCare
Project Full Title:	Innovative and affordable service for PC monitoring of individual Cultural Artefacts during display, storage, handling and transport
Call:	H2020-NMBP-ST-IND-2018-2020
Topic:	NMBP-33-2018
Type of Action:	IA
Grant Number:	814624
Project URL:	www.collectioncare.eu

Deliverable nature:	Other
Dissemination level:	Public
WP n^0:	3
WP title:	Big data cloud computing environment for preventive conservation management
Contractual Delivery Date:	Feb 2019
Delivery Date:	29th Feb 2020
Number of pages:	15
Keywords:	Storage, Connectivity, Cloud
Authors:	Jorge Montero, ATOS Sergio Salmerón, ATOS Tomás Pariente, ATOS
Reviewers:	Ángel Perles, UPV Jaime Laborda, UPV Annamaria Siani, URO1 Marina Moscarina, URO1

Abstract

This deliverable introduces the updates and current state of the data storage solution to be used in the CollectionCare project. In this sense, the infrastructure that has been set up in the cloud is going to be described, as well as the components deployed in the cloud for the storage. Additionally, the result of the upload process of the currently available historical data is described in this document.

Abbreviations and Acronyms Glossary

API	Application Programming Interface
AWS	Amazon Web Services
EBS	Elastic Block Storage
EC2	Elastic Cloud Compute
Gbps	Gigabit per second
Gib	Gibibyte
KB	Kilobyte
RCU	Read Capacity Unit
vCPU	Virtual CPU, Virtual Central Processing Unit
VM	Virtual Machine
WCU	Write Capacity Unit

List of figures

Figure 1. Updated version of the Swagger specification for CollectionCare Storage REST API 9

Contents

Abstract 3
Abbreviations and Acronyms Glossary 4
List of figures 5
Contents 6
1. Introduction 7
2. Review of STORAGE infrastructure 8
2.1 Database Table Schemas 8
2.2 API 8
3. Cloud infrastructure 11
3.1 Database Infrastructure 11
3.2 API Infrastructure 11
4. Data upload process results 12
5. Conclusions and next steps 13
Bibliography 14
Annex I 15

1. Introduction

Data storage is a key component in CollectionCare project. Its proper functioning guarantees that many other components in the project that depend on the data stored can work without any issues. In CollectionCare, the artwork monitoring is done all day long, so the technological infrastructure should guarantee its service 24h a day. That is the reason of choosing a cloud-based solution (in our case using Amazon Web Services) in order to deploy all the required technical infrastructure of the project.

Data storage is one of the first components defined and implemented in CollectionCare to be deployed in the cloud. Most design decisions regarding the data storage have already been taken and documented, defining the data schema of the project as well as the different operations that will allow the interaction between the different components to be deployed in the project. These operations were implemented in an API that carried out the corresponding actions in a local instance of the DynamoDB database to use in the project.

Some advances have been produced since the definition of the initial database schema and the data storage operations API. Database schema has been ported to a cloud based DynamoDB database and the API allowing the access to all the operations to interact with the data storage has been deployed in an Amazon EC2 virtual machine. This deployment has been tested by uploading the set of files including all the historical data provided by the partners in the CollectionCare project which now is included in the cloud-based database of the project.

2. Review of STORAGE infrastructure

As described in D3.2 (Salmerón, Montero & Pariente, 2019), a solution based on the NoSQL database DynamoDB has been implemented due to its versality and powerful to manage the variety of data coming from the sensors attached (i.e. assigned) to the artworks, and even the different information that could provide each museum for each artwork and for itself. A specific table schema has been adopted to manage all this information that could arrive without a predefined structure.

Linked to the storage database, a REST API has been created to implement all the services that allows to manage all the data stored, and query and provide this data to the degradation models and the user interfaces to be developed during the project lifetime.

Next section describes the updates that have been adopted to fulfil the needs of the CollectionCare project based on the discussions with the sensors and degradation models task participants. It is not a final implementation as further updates will be taken into consideration until the final version of the whole components are tested together, and other needs could arise. Most of potential future changes regarding the data storage coming from these components should not be a problem due to the adaptability of the storage solution developed when adding or changing the type of data to manage, as described in D3.2 (Salmerón, Montero & Pariente, 2019).

2.1 Database Table Schemas

There are two aspects to describe on the updates in the table schema:

- Sensor information to be provided and stored is not closed. Other variables are been taken into consideration to be measured in order to give more data flexibility to the degradation models. For example, some measurements related to vibrations, mainly for the transport of the artworks could be added. This kind of data will be stored easily in the sensor t table schema, thanks to the adopted solution, to be accessed in the cloud or even analyzed directly at the edge.
- Information needed by the degradation models to be executed. Extra data can be included as the Work package on the degradation models (WP 2) is still in operation. For example, it is necessary to have a description of the composition of the artwork to be able to fine tune the degradation models and apply the fittest one that will provide the best result. As presented before, it is not an issue as all this information would be attached to the artwork at the artworks table.

2.2 API

The other component in relation with the storage solution is the REST API that includes all the services developed to manage the data to be stored or queried. It works as an interface with the sensors to let them store the measurements, and with the degradation models to query and send the required data to execute those models in a proper way.

At this point, a change related to the sensors attached has been implemented to extend the functionalities and easy the process of attach and detach the sensors (see Figure 1).

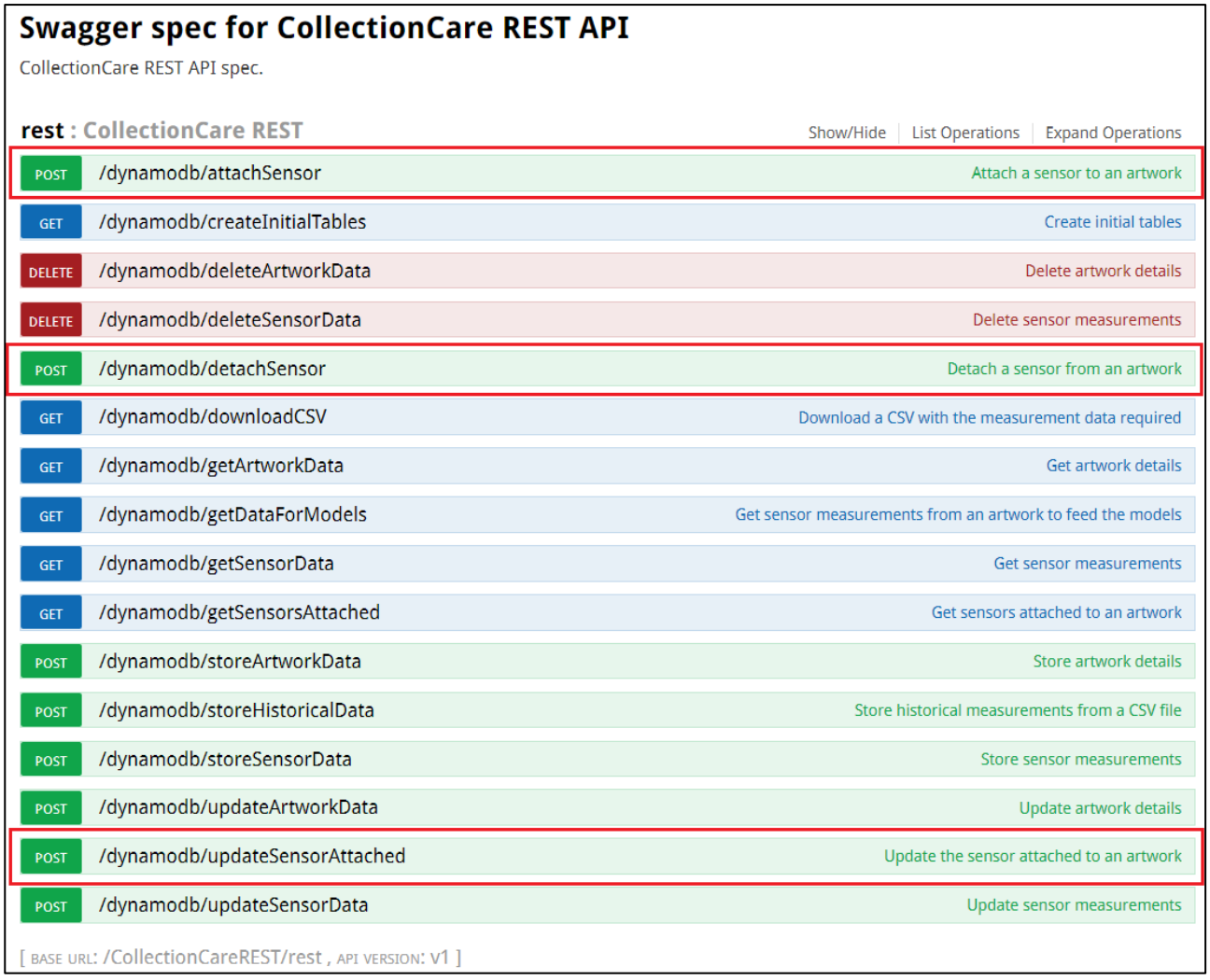

Figure 1. Updated version of the Swagger specification for CollectionCare Storage REST API

Following the approach of the previous deliverable, new and updated services are described:

- Attach_sensor: Attach a sensor node to an artwork (Artwork Table). This will associate the data collected by a sensor to a given element in the artworks table.
  - Input:
    - SensorID: The ID of the sensor to be installed in proximity to the artwork.
    - ArtworkID: The ID of the artwork sensed by that node.
    - AttachDate: The timestamp when the sensor node started.
  - Output:
    - A confirmation of the sensor node attachment.

- Detach_sensor: Detach a sensor node from an artwork (Artwork Table).
  - Input:
    - SensorID: The ID of the sensor to be detached.
    - ArtworkID: The ID of the artwork sensed by that node.
    - DetachDate: The timestamp when the sensor node stopped.
  - Output:
    - A confirmation of the sensor node detachment.

- Update sensor attached: Update if a sensor has changed its artwork target or modify the attachment/detachment dates (Artwork Table).
    - Input:
        - SensorID: The ID of the sensor to be updated.
        - ArtworkID: The ID of the artwork monitored by that node.
        - (Optional) AttachDate: The timestamp when the sensor node started.
        - (Optional) DetachDate: The timestamp when the sensor node stopped.
    - Output:
        - A confirmation of the sensor node update

In next phases of the CollectionCare project, the list of provided services could increase or be updated to manage new requirements. These updates will be related mainly to the ongoing development of the degradation models, and to the tasks that have been started recently (T3.4: Design and develop a user interface and a data visualization engine for the cloud infrastructure) or will start in the future (Task 3.6: Edge computing). The former will aim to probably implement a service to provide specific information to the user interfaces, and the latter will be devoted to send required information to the edge or to store any kind of data in the process.

3. Cloud infrastructure

Once the different components of the data storage solution have been clearly defined in D3.2 (Salmerón, Montero & Pariente, 2019) and implemented, next step is to deploy it in the cloud computing infrastructure facilitated in the project, described in D3.1 (Juan Ferrer et al., 2019). In this sense, Amazon Web Services has been chosen (as seen in D3.1 (Juan Ferrer et al., 2019)) as the cloud-based service provider that will allow not only the deployment of the solution designed, but also its maintenance.

As we have seen in the previous section, the data storage solution developed in the project consists of two main components: a NoSQL database and API implemented to perform the required functionalities regarding the interaction with the database. Due to the different functionality of the components, a different service in the cloud has been set up for each one of them.

3.1 Database Infrastructure

As it was mentioned in previous reports, the database service to be used was DynamoDB (Amazon, 2020a), so the two tables mentioned in section 2.1 were set up in DynamoDB. Both tables have been created with 10 Read Capacity Units (RCUs, the unit used to measure the number of consistent read operations per second for items up to 4KB) (Amazon, 2020d) and 10 Write Capacity Units (WCUs, the unit used to measure the number of consistent write operations per second for items up to 1KB) (Amazon, 2020d). This initial configuration is going to be monitored over time and adapted according to the project needs. Nevertheless, DynamoDB also provides the capability of setting up the resource allocation on-demand, which will be evaluated depending on the Read and Write Capacity Units consumption during the life of the project (as interactions with data storage will increase when sensing units and user interface functionalities start interacting with the data storage).

3.2 API Infrastructure

Regarding the required infrastructure to run the API developed to provide access to the different functionalities of the data storage layer of the project, a virtual machine (VM) was set up using Amazon Elastic Cloud Compute (Amazon EC2) (Amazon, 2020c). The machine configured to deploy the API developed is a m5a.large instance. m5a instances are General Purpose Instances, and the large one is the most economical one within m5a instances family. That kind of instances have 2 virtual CPUs (vCPUs), 8GiB of memory, 30GiB of Elastic Block Store (EBS) storage (Amazon, 2020b), and up to 10Gbps of network bandwidth. With these specifications, the VM should be capable to handle not only the API workload but also other workloads to come (e.g. web server, etc.). Nevertheless, Amazon EC2 allows the possibility of upscaling the instances so in case more resources are needed, a more powerful instance type can be set up.

Java and Tomcat have been installed in the current VM in order to deploy the java-based API developed. Additionally, a static IP address has been configured in order to be accessible by the different components that require the interaction with the data storage layer.

4. Data upload process results

Once the different components have been set up in the cloud and the API has been deployed, the historical data compiled in the project was uploaded. To do that, a shell script using the API services was run as well as the historical data provided in the deliverable D1.2 (Rossi Doria et al., 2019). This scrip calls the service developed for uploading historical data with each one of the files included in the deliverable aforementioned. The decision to run a script was to avoid the upload of every single file from D1.2 (Rossi Doria et al., 2019) (which contains 206 files) by hand.

After the upload of all the data, 2.646.633 values are currently available in the CollectionCare project database (currently consuming 218,59 MB) and can be queried through the different API implemented methods accessible by the project members.

5. Conclusions and next steps

An initial version of the data storage component of the project is currently online and functioning. It has been initially populated with 2.646.633 values and is ready to query data as well as to upload new (or historical) data.

Due to the forthcoming tasks, the current solution will be implemented in many ways to address the new functionalities of the project. As degradation models are included in the cloud solution, some new ways to access the data might have to be implemented. Additionally, the development and release of the Graphical User Interface (GUI) to interact with the project will require also the adaptation of some of the services provided to interact with the data storage.

The edge computing scenario (where the artworks have to be monitored in time windows where there is no internet connection) might also have an impact on the services to be provided in the data storage. Although currently expected scenarios (like the connection of the sensor units) might also evolve, the design of the current database schema should be capable to handle those changes, but in case the changes introduced in the way of sending or querying the data cannot be faced by the current setup, the flexibility of the approach adopted makes it quite simple to adapt the current solution to those potential changes.

Bibliography

Amazon. (2020a). Amazon DynamoDB. Retrieved from https://aws.amazon.com/dynamodb/ in February 20th 2020.

Amazon. (2020b). Amazon Elastic Block Store (EBS). Retrieved from https://aws.amazon.com/ebs/ in February 20th 2020.

Amazon. (2020c). Amazon EC2. Retrieved from https://aws.amazon.com/ec2/ in February 20th 2020.

Amazon. (2020d). Managing Throughput Settings on Provisioned Capacity Tables. Retrieved from https://docs.amazonaws.cn/en_us/amazondynamodb/latest/developerguide/ProvisionedThroughput.html in February 20th 2020.

Juan Ferrer, A., Salmerón, S., Montero, J., Pariente, T. (2019), Design and implementation of CollectionCare cloud computing architecture. CollectionCare Project, Deliverable D3.1.

Rossi Doria, M., Gittins, M., Mercuri, G., Perles, A & Peiró Vitoria, A. (2019). Compiled and unified historic environmental data of selected artworks of partner museums in .CSV file. CollectionCare Project, Deliverable D1.2.

Salmerón, S., Montero, J., Pariente, T. (2019). CollectionCare database storage I. Ready for data accommodation: Historic ambient data of artworks uploaded. CollectionCare Project, Deliverable D3.2.

Annex I

This is the script used to upload the historical data to the Amazon DynamoDB table. The variable SERVER has to be set pointing at the server URL:

```
#!/bin/bash
SERVER=<SERVER URL>
echo "Creating table structure..."
curl -X GET --header "Accept: application/json"
"$SERVER:8080/CollectionCareREST/rest/dynamodb/createInitialTables"
sleep 5
SCRIPTDIR="/home/ec2-user/historicaldata"
FILES="$SCRIPTDIR"/*
for f in $FILES
do
 echo "Processing $f file..."
 # take action on each file. $f store current file name
  curl -X POST --header "Content-Type: multipart/form-data" --header "Accept: application/json"
"$SERVER:8080/CollectionCareREST/rest/dynamodb/storeHistoricalData" -F "file=@$f"
 echo "\nProcessed $f file"
done
```