Horizon 2020

COLLECTION CARE

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

CollectionCare database storage I

Ready for data accommodation; Historic ambient data of artworks uploaded

Deliverable number: D3.2

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 Artifacts 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 (PU)
WP n^{0}:	WP3
WP title:	Big data cloud computing environment for preventive conservation management
Contractual Delivery Date:	December 2019
Delivery Date:	27th December 2019
Number of pages:	27
Keywords:	Database Storage, Storage Architecture, Storage Requirements, Amazon Web Services, DynamoDB, REST API, Amazon Lambda, Apache NiFi, Security
Authors:	Sergio Salmerón, ATOS Jorge Montero, ATOS Tomás Pariente, ATOS
Reviewers:	Ángel Perles, UPV Jaime Laborda, UPV Ana María García Castillo, UPV

Abstract

This deliverable reports on the implementation of CollectionCare database storage to be used by the whole project. The document explores different storage architectures based on the requirements specified in CollectionCare WP1 and under the umbrella of the overall cloud architecture defined in WP3.

It also describes the table schema chosen for storing the data and the communication mechanisms to interact with the database by providing a set of services.

Finally, a list of security measures is presented and initially implemented.

Abbreviations and Acronyms Glossary

API	Application Programming Interface
ATOS	Atos Spain SA
AWS	Amazon Web Services
CSV	Comma-Separated Values
D	Deliverable
DB	Database
GUI	Graphical User Interface
ID	Identity
JSON	JavaScript Object Notation
OAS	OpenAPI Specification
RDBMS	Relational Database Management System
REST	Representational State Transfer
SDK	Software Development Kit
WP	Work Package

List of figures

List of tables

Contents

1. Introduction

Data storage is fundamental when working with data. In CollectionCare, the use and management of data is essential, as it is the core of the project. Artworks are monitored by sensors and the estimation of potential damage and degradation is performed based on the data collected. Due to the complex infrastructure to be developed in the project (sensors sending data, models consuming that data, users viewing data through GUIs, etc.), it is important to carry out a wise design of the data storage layer, as it plays a central role in the technical infrastructure of the project. All the technical components developed interact with the data storage solution, which makes it a component that, if not properly set up, could lead to a bottleneck situation or even shut down the operation of the entire technical infrastructure of the project.

To avoid potential errors, it is very important to evaluate different possible approaches in order to design a storage solution that fits all the project requirements. To do so, different services have been estimated and an initial version of the storage solution to be deployed in the project has been developed. An initial proposal of the infrastructure to be set up as well as the schema to use to store the data currently identified in the project has been generated. In addition, some currently discarded services initially evaluated have been described and a justification of their exclusion is provided. Nevertheless, as the project continues, the solution proposed in this document will both continue to grow with missing functionalities identified at the current time and advance in the direction of the developments and changes introduced, adapting its functioning to the new requisites that might appear.

2. Summary and Review of STORAGE requirements

In deliverable D1.9 (Sánchez et al., 2019), a definition of technical requirements for CollectionCare was submitted. In that document, a subsection was dedicated to the technical requirements to be taken into account for the data storage solution to be adopted in the project, as well as a set of possible solutions to be used in the project. In this section, those initial solutions and technical requirements are reviewed, analysing how they have been addressed in the initial development of the data storage layer of the CollectionCare project.

2.1. Storage requirements

In this section, we shall evaluate the initial technical requirements of the storage solution. The following table was initially included in section 5 of D1.9 (Sánchez et al., 2019) and has been extended with the implementation approach followed in the current state of the data storage solution in the project.

Table 1. Technical requirements included in D1.9 [1] with the implementation approach followed to address them.

ID	TYPE	NAME	DESCRIPTION	IMPLEMENTATION APPROACH
R1.1	Functional	Sensor data storage	Provide a data storage solution adapted to store the data collected by the different sensor nodes deployed in the project. This solution should be built upon the cloud architecture implemented in the project and provide space enough (and scalability options) to store the data volume proposed in this project.	The data storage solution has been developed in order to work with AWS as back-end. This choice affords us enough space on demand and allows us to easily resolve any possible scalability problems due to the dynamic allocation of resources provided by AWS.
R1.2	Functional	Sensor data upload mechanisms	Provide the mechanisms required for uploading data in near-real time and concurrently. This requirement must be able to handle data uploading from all the sensor nodes with a frequency of up to four samples per hour and node.	An API has been developed to allow the different sensing nodes to send the collected data in a transparent way, storing all the data received in the corresponding measurements table.
R1.3	Functional	Historical data storage	Provide a data storage solution similar to the one provided for the sensor node data but this time able to handle different uploaded historical records. This solution must provide enough space and scalability options both to handle all the historic data proposed in this project and to grow with the potential inclusion of new data.	Historical data will be integrated in the sensors database, to complement the sensor collected data. In order to use the historical data together with the sensor data, it should be uploaded with the format set out in D1.2 (Rossi Doria et al., 2019).
R1.4	Functional	Historical data upload mechanisms	Provide the mechanisms that allow the upload of historical ambient condition data. This	Functionality to upload historical data has been developed. As in R1.2, an API

ID	TYPE	NAME	DESCRIPTION	IMPLEMENTATION APPROACH
			functionality should allow data uploading in .csv files.	method has been developed in order to upload historical data.
R1.5	Functional	Data access mechanisms for the cloud components	Allow the different elements of the cloud solution to access the data storage when needed and allowed.	As mentioned above, an API providing access to all the functionalities related to the storage layer of the CollectionCare solution has been developed.
R1.6	Functional	State of the artworks data storage	Provide the mechanisms and tools to allow the storage and updating of different information related to the historic or current state of the artworks being sensed.	A table containing artwork information has been designed, allowing the addition of all the data that the CollectionCare user considers necessary.
R1.7	Functional	Data storage permission management	Include the use of permission management tools to allow or deny access to the data depending on the design of the data storage solution.	Different security measurements have been evaluated at this point. Due to the integration of different components in the storage layer, there are security measurements to be included in the different components involved. This is further described in section 6.
R1.8	Functional	Data storage backup system	Configure a backup system that ensures the availability of the data in case an error occurs in the database. This mechanism should avoid or minimise any potential data loss that might derive from a human or technical failure.	A backup system is to be set up to guarantee the availability of data in case of unsolicited data loss. This can be done by automating the backup of the database in a different server or hiring the backup system offered by AWS. Nevertheless, the possibility of downloading the data through an API service is also being implemented.
R1.9	Functional	Data pre-processing mechanisms	Set up a series of pre-processing operations to be applied to the incoming data from the different partners in order to adapt the data format to the storage in the cloud solution developed in the project.	A system able to handle different pre-processing techniques (e.g. unit conversion, timestamp adaptation, formatting, etc.) is evaluated. This point is currently discussed in the section 5.1 of this document.
R3.1	Technical	APIs expected traffic	Be able to serve public and private REST APIs which expect about 30-60 requests per minute.	An API connected to the AWS storage solution has been developed to allow the execution of all the interactions expected to take place with the data storage of the project.

3. Overall CollectionCare STORAGE Architecture

A wide range of infrastructural items could be used for implementation of the storage solution in CollectionCare. Amazon Web Services (AWS) has been chosen as the main service provider, thanks to the many possibilities it offers regarding cloud infrastructure, matching all the requirements introduced in D1.9 (Sánchez et al., 2019). Nevertheless, due to the amount of services offered, different approaches could have been followed depending on the services chosen to that end.

3.1. Evaluated approaches

As AWS is the most popular cloud service provider, the focus was set on their services when evaluating different services to deploy the CollectionCare data storage solution. Initially (as mentioned in section 2.1 of D1.9 (Sánchez et al., 2019)), AWS S3 and AWS RDS were listed as possible solutions to be included in the project, but AWS DynamoDB has been chosen as the main service to implement the storage solution in the project. With the technical requirements defined, we studied the adoption of each of the initial storage methods proposed in the cloud service options:

- Amazon Simple Storage Service (S3) (Amazon, 2019f) is the most popular object storage service provided by AWS. It does provide a good service for storing files, but due to the nature of the data to be used in the project and the way it will be accessed, this solution might not be the most suitable to match the needs of the project. This service is a good solution for storing unstructured data, but due to the way the data may be accessed in the project, perhaps a database solution may perform better (as the data is expected to be accessed in a similar way every time the models are executed).
- Amazon Relational Database Service RDS (Amazon, 2019e) was also a candidate to be used in the project, as it is the most widely used relational database management system (RDBMS) service in the AWS environment. It allows users to store data and provides performance with a very low latency, which might be useful for the project. Nevertheless, the structure of the data to be stored needs to be designed beforehand, and that might affect the future of the project, where different versions of the sensing stations are expected to be evaluated (adding new parameters to the databases). Additionally, each museum might want to add different elements to the database, even depending on the artwork to be described. This need to predefine the structure of the databases and the difficulties of updating this database schema when needed were the basis for ruling out this option in favour of a more flexible approach to the database.

3.2. Proposed storage solution

After a careful assessment of the proposed services, Amazon DynamoDB (Amazon, 2019a) was chosen as the most suitable solution for use in CollectionCare. Amazon DynamoDB is the main big data solution provided by Amazon Web Services. It provides the functionality of a NoSQL database without having to purchase or maintain any server. It allows data storage offering a series of advantages for the project that defined the decision for this service.

DynamoDB follows a Key-Value approach, which allows us to store many different types of data once a set of keys have been defined. In our case, some keys such as artwork can have any kind of data attached, which allows us to have a database of artworks each with a different data schema, which might vary according to the peculiarities of the artwork (e.g. the different materials of the artwork, different restorations of the artwork, particular information for that artwork, etc.).

DynamoDB offers very low latency, which allows the different components of the system to consume the project data with almost no temporal impact. This will be useful when the models need to get all the measurements attached to an artwork or when the GUI might ask for some measurement or artwork information to be presented to the user.

As it is an AWS service, almost no maintenance of the database is required, with a set of configurations at the beginning of the project defining how it is expected the DB will be maintained, and AWS will automate many maintenance tasks.

At this stage of the project, other tools have been introduced for the development of certain mechanisms related to the data storage. The connectors required for interacting with the database are being implemented as an API using Swagger to provide the required services for interacting with the data storage solution of the project. In this sense, we can define the operations that will be allowed in the data storage layer, reducing risks and making security and control of the data accesses in the project easier. Figure 1 depicts the approach followed in the project (with a detailed description of the API in the centre of the image in section 5 and a detailed description of the tables in the right side of the image in section 4).

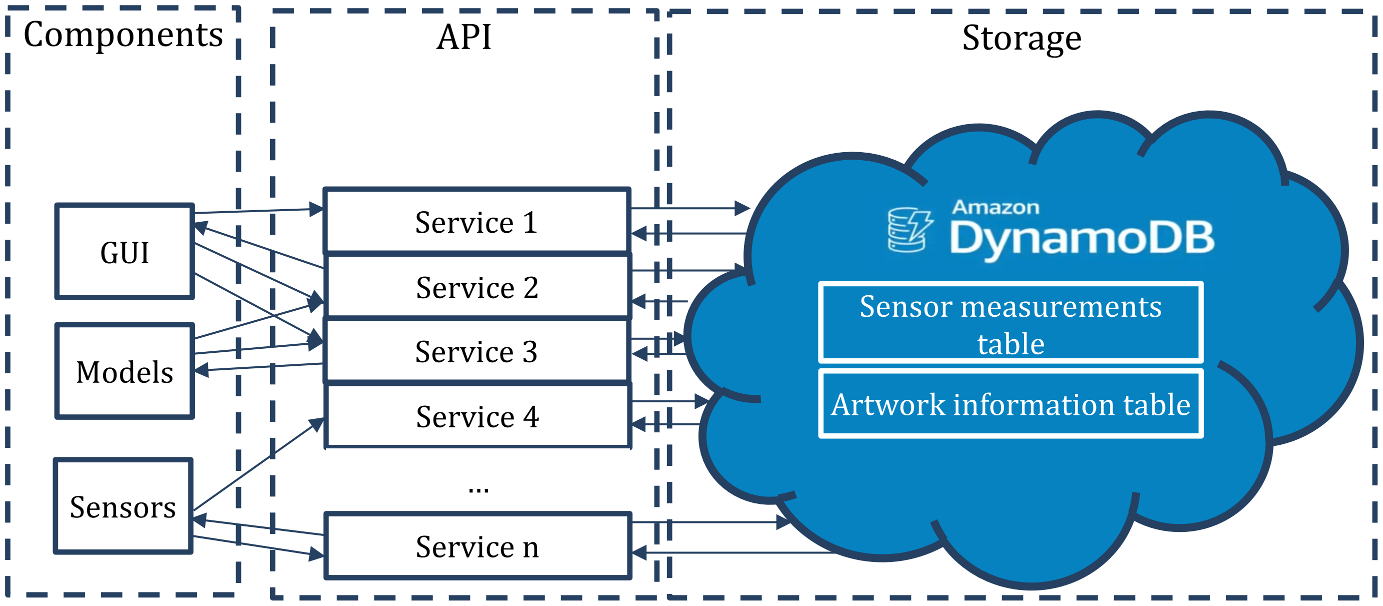


Figure 1. Graphical description of the API connections between the components and the database

4. Table schemas

Based on the storage requirements described in Section 2.1, it is necessary to store different kind of data from different sources.

4.1. Physical sensors source

The main source consists of all the sensors installed in the museums that will send measurements in real time. The specific requirements are set out in D1.4 (Perles, Peiró Vitoria & Chazal, 2019) and the measurements to be captured are mainly temperature (T), relative humidity (RH), light (L) and air pollutants (AP). There is a possibility that the battery status and other parameters could be provided along with the rest of measures.

To cover all these requirements, a table will be created in the database with the specific characteristics described below.

DynamoDB lets us define a composite primary key to uniquely identify each measurement, with a partition key and a sort key. In our case, the partition case will be assigned to the "sensor_id" that identifies the physical device, and the sort key will be the "timestamp" when the device has sent the measurements. The sorting key is very useful to query the data efficiently, for example to get some data in a time range. Table 2 describes this composite primary key of the sensor measurements table.

Table 2. Composite primary key of sensor measurements table

FIELD	TYPE	DESCRIPTION
sensor_id (partition key)	uint32_t	Device identification number for the physical node
timestamp (sorting key)	String/Date	Timestamp of received measurement (UTC time, according to ISO_8601)

On the other hand, the table will store all the measurements provided by the sensors as attributes. DynamoDB has the property of storing only one, two, all or any combination. It is useful for us because not all the sensors capture all the attributes, as it is common practice to set sensors to gather specific attributes so as not to overload them and ensure better performance and longer battery life. Later on, the data gathered could be merged in the database. Table 3 has a description of those various attributes, where there are attributes that have to be defined but with no impact on the table schema defined.

Table 3. Attributes for sensor measurements table

FIELD	TYPE	DESCRIPTION
temperature	Float	Temperature in Celsius degree (0.01 resolution)
relative_humidity	Float	Percentage (0.1 resolution)
ambient_light	Float	Ambient light in lux (1lux

FIELD	TYPE	DESCRIPTION
		resolution)
pollutants	To be defined	To be defined
battery_status	To be defined	To be defined
battery_temperature	To be defined	To be defined
...	...	...

Table 4 provides an example of the data that could be stored, with one sensor taking temperature and relative humidity measurements and the other gathering the ambient light data:

Table 4. Sensor measurements table with example data

PRIMARY KEY		ATTRIBUTES			
sensor_id	timestamp	temperature	relative_humidity	ambient_light	...
S0001	20190101T000000	24.50	45.4		...
S0001	20190101T100000	24.63	46.0		...
S0002	20190101T110000			25	...
S0001	20190101T220000	25.05	41.2		...
S0002	20190101T230000			20	...
...	...	...	...	...	...

In Section 4.3 the relation between the sensors and the artworks will be explained.

4.2. Historical data source

The second source, important to the models, is related to the historical data that the museums will upload, which contains environmental data on the selected artwork for a range of time in the past. This information is compiled and unified in CSV files, following the standards defined in D1.2 (Rossi Doria et al., 2019).

All this information will be stored in the table created in Section 4.1 with a new sensor_id created for the historical data and linked to the artwork.

Those CSV files will be stored in the CollectionCare database with the help of the REST API services provided with the database that will permit the communication with it. More details are provided in section 5.

4.3. Artworks details

To store the information related to the artworks, another table is needed, not only for the artwork details, but also for the sensors attached over time. The composite primary key is described in Table 5:

Table 5. Composite primary key of artwork table

FIELD	TYPE	DESCRIPTION
artwork_id (partition key)	String	Artwork identification string
varies (sorting key)	String	It can be "**details**" to describe the details of the artwork, or the version control to know which sensor is measuring ("**v_0**" for actual sensor, or "**v_timestamp**" for historical sensor)

DynamoDB lets us define a sort key that allows us to store this kind of information and even a version control to know which sensor is measuring that artwork at this time, with a historical representation of the different sensors that have measured what and when. This historical information is very useful to get the measurement data from the sensor measurement table by querying a specific sensor_id in a range of dates. In sensor_attached the information will be a "v0" field to provide fast access to the current sensor measuring without querying more information not required.

As described for the sensor measurement table, the artwork table will include a variety of attributes that may appear or not, thanks to the flexibility of DynamoDB, as described in Table 6:

Table 6. Attributes for artwork table

FIELD	TYPE	DESCRIPTION
museum_id (details)	String	Museum identification string to identify which museum has the artwork
room_id (details)	String	Room identification string to identify in which room the artwork is located
... (details)	...	Other attributes needed to be stored related to the details of the artwork
sensor_id (v_X)	uint32_t	Device identification number for the physical node
start_date (v_X)	String/Date	Date when the sensor has been attached to the artwork
end_date (v_X)	String/Date	Date when the sensor has been removed from the artwork
... (v_X)	...	Other attributes needed to be stored related to the sensors attached to the artwork

To clarify this table schema and provide a better understanding by describing an example, Table 7 presents an example of different artworks that could be stored in the CollectionCare database and could be queried by the services described in Section 5.

Table 7. Artwork table with example data

PRIMARY KEY		ATTRIBUTES			
artwork_id	varies	Attribute 1	Attribute 2	Attribute 3	…
A0001	details	*museum_id*: M0001	*room_id*: R0001		…
	v_0	sensor_id: S0003	*start_date*: 20090501T000000		…
	v_1	sensor_id: S0001	*start_date*: 20090101T000000	*end_date*: 20090301T000000	…
	v_2	sensor_id: S0002	*start_date*: 20090301T000000	*end_date*: 20090501T000000	…
	v_3	sensor_id: S0003	*start_date*: 20090501T000000		…
A0002	details	*Museum_id*: M0002	*Room_id*: R0088		…
	v_0	sensor_id: S0077	*start_date*: 20091001T000000		…
	v_1	sensor_id: S0078	*start_date*: 20090901T000000	*end_date*: 20091001T000000	…
	v_2	sensor_id: S0077	*start_date*: 20091001T000000		…
…	…	…	…	…	…

4.4. Degradation models source

The last source to take into consideration is the one related to the degradation models. It will probably be necessary to have another table to store the results of the different models to be executed in CollectionCare, to be able to show them later. All this information is being delivered under D2.1 (Bosco et al., 2019) and is being taken into account for the next deliverable of the CollectionCare database storage II D3.6.

It is not a critical path for this database, as DynamoDB let us create this new table, or even easily add this information in the same way as the sensor attached to the artwork table described in Section 4.3. This will involve developing the services for storing and gathering this information in the REST API developed in Section 5 with few modifications and in a short period of time.

4.5. Other sources

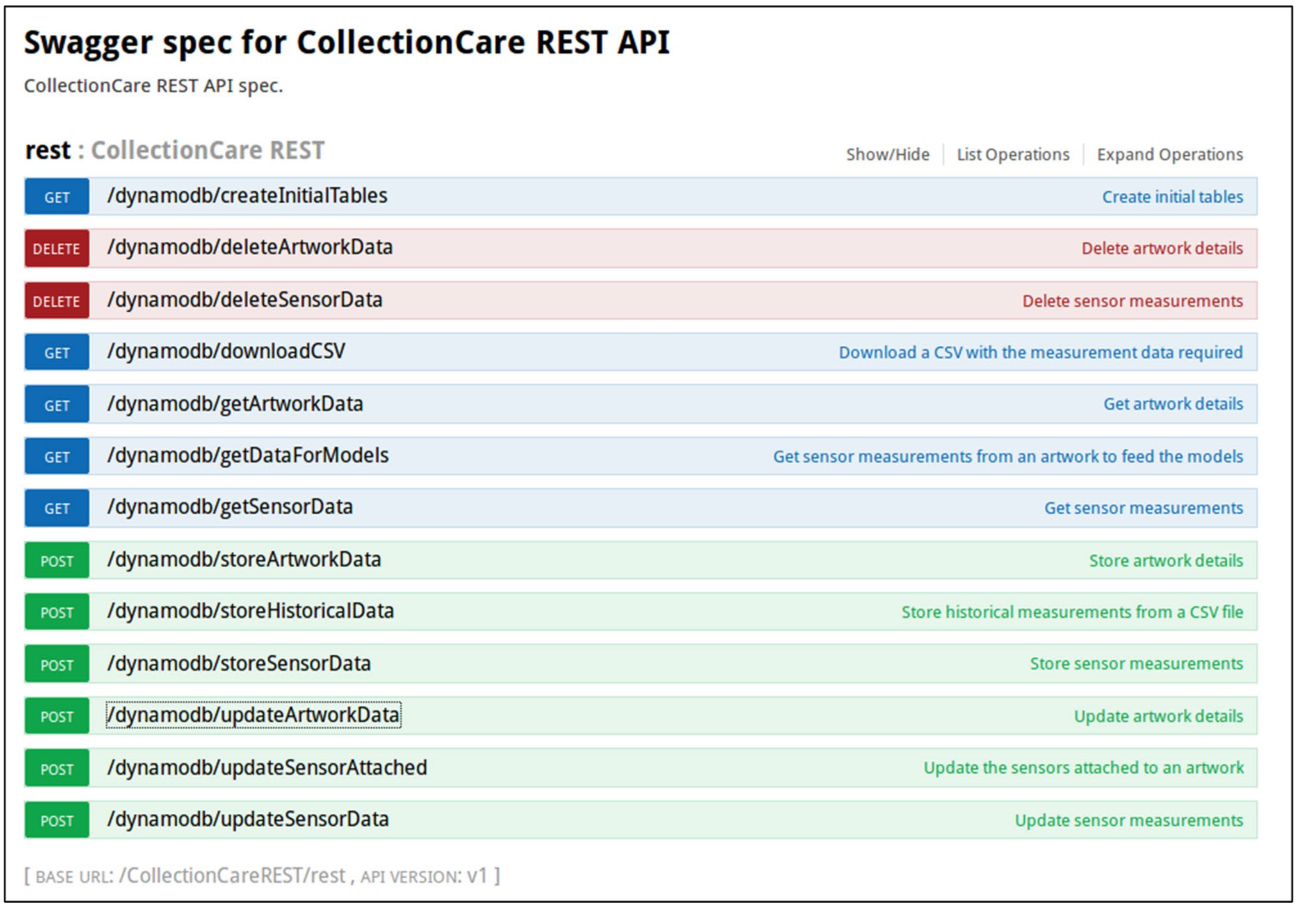
As described in this section, it is easy to update the table schema or even add new tables to the database. Because of this, new requirements that could appear during the project will be taken into consideration and could easily be added to the CollectionCare database.

5. CollectionCare Storage Communication Services

To provide a way of facilitating communication between CollectionCare and the database, a REST API will be developed jointly with the database. This implementation will be based on Swagger (SmartBear, 2019b), which is one of the best frameworks for the OpenAPI Specification (OAS). Swagger covers the whole API lifecycle, starting at the design phase and continuing with the development, but also covering documentation and test phases. The main point is that Swagger presents a REST service that can be available to others in a secure way.

For CollectionCare, this Swagger framework has been implemented in Java because it can be easily integrated in the cloud environment and Amazon Web Services provides a Java API by an SDK (Amazon, 2019g) that works with DynamoDB.

Figure 2 shows a summary of the services implemented. In the rest of this section, all those services are described and the input and output parameters explained, as well as what they do and where they work inside the database, and the permissions they could have (if they are private or public access).



Swagger spec for CollectionCare REST API

CollectionCare REST API spec.

rest : CollectionCare REST — Show/Hide | List Operations | Expand Operations

GET	/dynamodb/createInitialTables	Create initial tables
DELETE	/dynamodb/deleteArtworkData	Delete artwork details
DELETE	/dynamodb/deleteSensorData	Delete sensor measurements
GET	/dynamodb/downloadCSV	Download a CSV with the measurement data required
GET	/dynamodb/getArtworkData	Get artwork details
GET	/dynamodb/getDataForModels	Get sensor measurements from an artwork to feed the models
GET	/dynamodb/getSensorData	Get sensor measurements
POST	/dynamodb/storeArtworkData	Store artwork details
POST	/dynamodb/storeHistoricalData	Store historical measurements from a CSV file
POST	/dynamodb/storeSensorData	Store sensor measurements
POST	/dynamodb/updateArtworkData	Update artwork details
POST	/dynamodb/updateSensorAttached	Update the sensors attached to an artwork
POST	/dynamodb/updateSensorData	Update sensor measurements

[BASE URL: /CollectionCareREST/rest , API VERSION: v1]

Figure 2. Swagger specification for CollectionCare Storage REST API

At this point of the project the following services are presented, but the list could be extended with more services based on the requirements and needs that might appear in subsequent phases of CollectionCare.

- Create initial tables (Private): this service is used to initially create the tables described in Section 4 to be used in CollectionCare for a new DynamoDB instance.

- Backup (Private): there is the possibility of generating a database backup at a specific time instead of waiting for the periodical backup set by default.

- Restore (Private): restore a backup from a list of available backups.

- Search (Public/Protected):
    - Search measurements: get the data related to a sensor device.
        - Steps:
            - Get measurements of the sensor in that range of dates (Sensor Table). Call "Search measurements" service.
        - Input:
            - SensorID: the ID of the sensor device.
            - Temporal range: The time window of the requested data.
            - (Optional) Variables requested: the variables to be retrieved from the resultant registries. If no variables are defined, all the variables for each registry are returned.
        - Output:
            - An array of elements containing the variables requested for a given sensor in a given time window.

    - Search artwork: get the information of a specific artwork.
        - Steps:
            - Get the information of the artwork specified (Artwork Table).
        - Input:
            - ArtworkID: the ID of the artwork sensed.
        - Output:
            - An array of elements containing the information for a given artwork.

    - Search for models: mainly to feed the models with the data from an artwork in a range of dates.
        - Steps:
            - Step 1: Find the sensors attached to the artwork in that range of dates (Artwork Table).
            - Step 2: Get measurements of those sensors in that range of dates (Sensor Table). Call "Search measurements" service.
        - Input:
            - ArtworkID: the ID of the artwork sensed.
            - Temporal range: The time window of the requested data.
            - (Optional) Variables requested: the variables to be retrieved from the resultant registries. If no variables are defined, all the variables for each registry are returned.
        - Output:

            - An array of elements containing the variables requested for a given artwork in a given time window.

- Storage (Public/Protected): to store the measurements and artworks.
    - Artworks: store in Artwork Table the details of the artwork.
        - Input:
            - ArtworkID: the ID of the artwork to be updated
            - Data: An array of elements:
                - Variable: the name of the variable to be stored
                - Value: the value to be stored for the variable
        - Output:
            - A confirmation of the addition of the data sent.

    - Sensor data: store in Sensor Table the sensor measurement. This service will be usually be called automatically by the sensor nodes
        - Input:
            - SensorID: The ID of the sensor sending the data
            - Timestamp: The time stamp of the data collected
            - Data: An array of elements:
                - Variable: the name of the variable to be stored
                - Value: the value to be stored for the variable
        - Output:
            - A confirmation of the addition of the data sent.

    - Historical data (adaptors): store in Sensor Table the measurements of the historical data from the selected artworks provided by the museums. Sensor_id will be set as default or empty.
        - Input:
            - (Optional) SensorID: The ID of the sensor to assign the data. If not specified, a fictional default value will be given.
            - Timestamp: The time stamp of the data collected.
            - Data: An array of elements:
                - Variable: the name of the variable to be stored
                - Value: the value to be stored for the variable
        - Output:
            - A confirmation of the addition of the data sent.

- Update (Public/Protected):
    - Update_sensor_data: Update the measurement of a sensor in a period, i.e. adjust temperature (Sensor Table).
        - Input:
            - SensorID: The ID of the sensor sending the data
            - Timestamp: The time stamp of the data collected
            - Data: An array of elements:
                - Variable: the name of the variable to be updated
                - Value: the value to be updated for the variable
        - Output:
            - A confirmation of the addition of the data sent.

- Update_artwork: Update artwork details (Artwork Table)
    - Input:
        - ArtworkID: The ID of the artwork to be modified
        - Data: An array of elements:
            - Variable: the name of the variable to be updated
            - Value: the value to be updated for the variable
    - Output:
        - A confirmation of the addition of the data sent.

- Update sensors attached: Update if a sensor has change of artwork target or has been removed or added (Artwork Table).
    - Input:
        - SensorID: The ID of the sensor to be updated.
        - (Optional)PreviousArtworkID: The ID of the artwork that was sensed by that node.
        - (Optional)NextArtworkID: The ID of the artwork that will be sensed by that node from now on.
        - Timestamp: The time stamp of the sensor node change.
    - Output:
        - A confirmation of the addition of the sensor node update.
    - Note: One of the optional fields (i.e. PreviousArtworkID or NextArtworkID) has to be present

- Delete (Public/Protected):
    - Delete measurements: delete some specific measurements from a sensor. If some sensor analysed incorrect data in a specific range of dates and those measurements must be removed from Sensor Table.
        - Input:
            - SensorID: the ID of the sensor that recorded the data to be deleted.
            - Temporal range: The time window of the data to be deleted.
            - (Optional) Variables to be deleted: the variables to be deleted from the resultant registries. If no variables are defined, all the variables for each registry are deleted.
        - Output:
            - A confirmation of the deletion of the requested registries.

    - Delete artwork: All the information about one artwork (Artwork Table)
        - Input:
            - ArtworkID: the ID of the artwork whose data has to be deleted.
            - (Optional) Variables to be deleted: the variables to be deleted. If no variables are defined, all the variables are deleted.
        - Output:
            - A confirmation of the deletion of the requested registries.
        - Note: It can be followed by deleting the measurements of that artwork if needed (Sensor Table).

- Download CSV (Public/Protected): For a specific sensor, download their measurements in a range (Sensor Table).
    - Input:
        - SensorID: the ID of the sensor that recorded the data to be downloaded.
        - Temporal range: The time window of the data to be downloaded.
        - (Optional) Variables to be downloaded: the variables to be downloaded from the resulting registries. If no variables are defined, all the variables for each registry are downloaded.
    - Output:
        - A CSV file containing the measurements of the sensor.

5.1. Connectors

Regarding the task of storing the historical data, we tested other solutions that were discarded because with the approach followed using Swagger it is better to use Java libraries. In our case, we used Apache Commons CSV (Apache, 2019a) for reading the CSV files provided by the museums and to download the CSV file with a group of data required in the service described above.

The two alternatives tested are based on Amazon Lambda and Apache NiFi.

5.1.1. Amazon Lambda

Amazon Lambda (Amazon, 2019c) is a service provided by Amazon Web Services that allows the running of scripts following a serverless approach. It works by executing the scripts triggered by actions defined by the user. It has great potential, as it supports Node.js, Python, Java, Go, Ruby and C#, which makes the execution of almost any code possible. Nevertheless, it has some limitations, such as the resources to be provided for execution, as well as a time limit on the execution of each process (currently set to a max of 15 minutes).

As Amazon Lambda is a service of the AWS environment, it does not require any server or maintenance, but, in contrast, its use requires payment for every execution request, as well as for the duration of the execution (with a price adapted to the resources devoted to that function) (Amazon, 2019d).

Some tests have been carried out in the project with AWS Lambda (automating the uploading of any csv file in AWS S3 with historical data to the DynamoDB database), but at this stage of the project it has not been included in the current solution. Nevertheless, due to its inclusion in the AWS environment, the interconnection between AWS Lambda and AWS DynamoDB is quite accessible.

5.1.2. Apache NiFi

Apache NiFi (Apache, 2018) is a well-known framework for processing and distributing data. It allows us to create specific processors to transform the data coming from the historical CSV files to the table schema created for CollectionCare. Through this web-based user interface it is possible to easily create a flow with highly configurable properties, in our case to be able to join the CSV management and connect it to DynamoDB. Testing and development phases are very fast, with continuous monitoring of the dataflow throughout the process.

This Apache NiFi flow should be executed in the environment described in D3.1 (Juan, 2020) by having a call that launches the process in the REST API. As mentioned before, this solution has been ruled out because it involves another piece to match in the cloud infrastructure and another added management task to control whether it is executed correctly and efficiently.

Just in case this solution could be adopted in the future or in another situation, Apache NiFi provides different components to communicate with DynamoDB. The main one for this connector is “PutDynamoDB” (Apache, 2019b), allowing the user to connect to a table created in DynamoDB and store the required data. The data to be stored must be in JSON format. In (Apache, 2019c), not only the components related to DynamoDB are described, but also all the processors for working with AWS.

6. Security measures and implementation

An important issue when working with databases is security, to avoid problems with fraudulent actions, forbidden accesses or other events that could result in inconsistent data or erase important records for the project, and denial of services.

It is not only about the data or the usage, but also the monetisation when you are working under the AWS umbrella. It is important to be very careful and keep everything under control to spend only the costs required. However, AWS provides security mechanisms for all their applications, and in this case for DynamoDB (Amazon, 2019b).

As we have another component, the REST API, added security measures are required. For this deliverable, all those security actions discovered are described, and they will be implemented for the next deliverable D3.6, with more actions that may be needed to ensure the successful results of CollectionCare.

The first action is related to who can access to the REST API services, as an authentication and authorisation process is required (SmartBear, 2019a). To get to this point, Swagger follows the security scheme used in OpenAPI, by defining a set of headers, or by using cookies, to read the API keys and check if they are allowed to access the services. In the case of controlling different actions requested by the users, as reads or writes, an OAuth 2.0 protocol could be added. With this authorisation it is possible to set specific rights to determined services based on the role of the user leveraging the API.

All these security measures will let us control, or at least minimise, attacks by hackers or users with fraudulent objectives.

The second action should cover the ownership of the data, for letting the CollectionCare users have access to the data that they own or to which they have permission. This is the scenario where a museum cannot access the information of another museum. This action is intended to be mitigated by having roles in the project, to describe who is asking for information and which data is assigned to them. Specifically, a solution consists of adding a process at the output of the services to filter the data and only return the one that fits.

More actions or a refinement of the same will be taken into consideration to cover all the needs of each museum and CollectionCare itself.

7. Conclusions and next steps

In this document, a description of the initial version of the storage solution has been provided. As we have seen, the different approaches proposed at the beginning of the project have been evaluated, taking into account the technical requirements identified in D1.9 of the project. An initial version of the storage solution has been developed and is being tested, in order to cover all the required functionalities of the project. The initial approach proposed seems to adapt to the initial requirements identified and has been designed with some characteristics such as flexibility in mind to ease future adaptations to possible changes that could occur in the different components that interact with the storage layer in the project.

Nevertheless, although this version of the storage solution is ready for data accommodation, the development of the storage solution is still live, as the results of the tests carried out and the identification of potential new issues have to be taken into account in future releases of the storage solution (D3.6), with a more advanced and integrated development and the historical data uploaded.

Bibliography

Amazon. (2019a). Amazon DynamoDB. Retrieved from https://aws.amazon.com/dynamodb/ in December 13th 2019.

Amazon. (2019b). Security in Amazon DynamoDB. Retrieved from https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/security.html in December 05th 2019.

Amazon. (2019c). Amazon Lambda. Retrieved from https://aws.amazon.com/lambda/ in December 13th 2019.

Amazon. (2019d). Amazon Lambda Pricing. Retrieved from https://aws.amazon.com/lambda/pricing/ in December 13th 2019.

Amazon. (2019e). Amazon Relational Database Service. Retrieved from https://aws.amazon.com/rds/ in December 13th 2019.

Amazon. (2019f). Amazon Simple Storage Service (S3). Retrieved from https://aws.amazon.com/s3/ in December 13th 2019.

Amazon. (2019g). AWS SDK for Java 2.0 Developer Guide. Retrieved from https://docs.aws.amazon.com/sdk-for-java/v2/developer-guide/welcome.html in December 05th 2019.

Apache. (2018). Apache NiFi. Retrieved from https://nifi.apache.org/ in December 05th 2019.

Apache. (2019a). Apache Commons CSV. Retrieved form https://commons.apache.org/proper/commons-csv/ in December 05th 2019.

Apache. (2019b). Apache NiFi – PutDynamoDB processor. Retrieved from https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-aws-nar/1.10.0/org.apache.nifi.processors.aws.dynamodb.PutDynamoDB/ in December 05th 2019.

Apache. (2019c) Apache NiFi Overview. Retrieved from https://nifi.apache.org/docs/nifi-docs/ in December 05th 2019.

Bosco, E., Parsa Sadr, A., Krarup Anderson, C., Fuster López, L., Bratasz, L., Andersons, B., Siani, A., Frasca, F., Sang-Hoon Lee, D., Kim, N., Kępa, L., Wagner, B., Gąsiorowska, I., Zschech, E., Kozłowki, R. & Kutorasiński, K. (2019). *Progress report I. Monitoring the progress in the tailoring of the degradation models for canvas paintings, wooden objects, paper art-objects and metal art objects*. CollectionCare Project, Deliverable D2.1.

Juan, A (2020). *Design and implementation of CollectionCare cloud computing architecture.* CollectionCare Project, Deliverable D3.1.

Perles, A., Peiró Vitoria, A & Chazal, S. (2019). *Definition of technical requirements for wireless communication.* CollectionCare Project, Deliverable D1.4.

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.

Sánchez. Á., Salmerón, S., Montero, J., Pariente, T. & Juan. A. (2019). *Definition of technical requirements for cloud computing*. CollectionCare Project, Deliverable D1.9.

SmartBear. (2019a). Swagger - Authentication and Authorization. Retrieved from https://swagger.io/docs/specification/authentication/ in December 05th 2019.

SmartBear. (2019b). Swagger Open Source. Retrieved from https://swagger.io/ in December 05th 2019.