With the contribution of the European Maritime
and Fisheries Fund of the European Union

DOCC-OFF

Project 863696 – DOCC-OFF – EMFF-BlueEconomy-2018

Scaling-up Digitalization Of Critical Components in OFFshore wind turbines

D2.4 Digital Computing Digital Architecture Specification

Use cases driven architecture approaches analysis and final solution technical and functional specification.

DELIVERABLE INFORMATION

Deliverable title: Digital Computing Digital Architecture Specification

Due date: 30/09/2020

Date of submission:

Work Package: WP 2 – Digital technologies specification and development

Dissemination level: Public

Deliverable leader: xabet digital solutions SL

ABSTRACT

The present deliverable corresponds to **D2.4 Digital Computing Digital Architecture Specification.**

This deliverable is focused on the specification of a scalable and modular Digital Architecture to contain all the developments chased during the DOCC-OFF project. It provides details of different modules to ensure a good understanding and base of all the developments that would be built during the project. Given that this architecture will manage all the data of the DOCC-OFF project it takes into account the DOCC-OFF DATA GOVERNANCE MODEL looking after Findable data, Accessible data, Interoperable data and Reusable data.

In addition, the main attributes that will be taken into account for this specification will be:

- Scalability
- Simplicity.
- Modularity.
- User experience.
- Openness or integrability to market standards adaptations, flexibility.
- Independency of third parties licenses or services (opensource).
- Sturdiness and cybersecurity.
- Maintainability.

INDEX

INDEX OF FIGURES

EXECUTIVE SUMMARY

The present deliverable corresponds to **D2.4 Digital Computing Digital Architecture Specification.**

This deliverable is focused on the specification of a scalable and modular Digital Architecture to contain all the developments chased during the DOCC-OFF project. It provides details of different modules to ensure a good understanding and base of all the developments that would be built during the project. Given that this architecture will manage all the data of the DOCC-OFF project it takes into account the DOCC-OFF DATA GOVERNANCE MODEL looking after Findable data, Accessible data, Interoperable data and Reusable data.

In addition, the main attributes that will be taken into account for this specification will be:

- Scalability: in order to grow and add new features and new assets and/or subsystems to the existing research, the architecture will be the most scalable and friendly possible.
- Simplicity: the architecture will be based in small pieces to understand very easily what does each one of them. Following this simplicity all the items which will take place into the architecture will be transparent and easy to understand.
- Modularity: thanks to the modular architecture, every main block could be replaced for a new one without affecting the other dramatically. This orientation will ensure a friendly maintainability of the architecture avoiding obsolescence in some of the elements that are being used to develop it. The different layer approach is possible thanks to this modularity.
- User experience: one of the most important attributes of the architecture will be the user experience of the front end together with the back end for developers, too, in order to grow and maintain the architecture.
- Openness or integrability to market standards adaptations, flexibility: As we are focusing in one subsystem but with the hope and mind of scale this project to other subsystems and players or stakeholders, the architecture will take into account the main standards of the renewable industry (onshore, offshore renewable energy production assets).
- Independency of third parties´ licenses or services (opensource): the architecture will be based in open source or license free libraries, modules and technologies. This way we guarantee the interoperability that is chased at the time that we developed an open environment for co-working.
- Sturdiness and cybersecurity: in one hand, the access to the digital platform will be secure and all the data will travel encrypted in both ways (reading and writing). On the other hand, the architecture will be built with a recover button strategy which means that all the data and modules will be restored just by clicking a button if a disaster or loss occurs. To ensure this sturdiness, a continuous and automatic backup strategy will be defined and implemented, too.

- Maintainability: last but not least, the architecture will be smart to allow the minimum maintenance possible in order to keep it working properly. The architecture will have some self-analysis techniques implemented to ensure that everything is right or where a problem could appear in the short future. This autonomous and continuous monitoring reduces drastically all the maintenance needs of the digital platform.

This deliverable shows with detail each module of the architecture to ensure a good basis for WP2, WP3 and WP4.

1. INTRODUCTION

This specification is designed following the experience of xabet within the energy industry and thanks to its extended experience managing data from renewable assets. In addition, the consortium partners add its experience and needs, too. This derivable answers all the requirements that are being detected for the correct achievement of the different challenges of each Work Package of the DOCC-OFF project.

This document has been prepared with four main sections. First, an overview of the digital architecture which includes the detailed scheme and all the modules with their specification and description will be presented. This architecture Second, the technological stack that will be used for the development together with the libraries and open source resources which will be included into the architecture development. Third, a technical description of each module of the architecture making emphasis into the communication between them and how the data flows. And fourth, the database's description and definition.

The digital architecture specification will allow the consortium a data analytics environment with a high performance computing in different layers (cloud, fog and edge layer) and slanted to the cloud layer after a deep analysis of the consortium needs.

2. OVERVIEW OF DIGITAL ARCHITECTURE

In this section an overview of the Digital Architecture Specification is presented. This architecture will be the basis of the digital platform of the WP of the DOCC-OFF project. To develop this Digital platform a unique and new design system will be developed to guarantee the flexibility and customization process that will be needed, avoiding third parties´ dependencies.

This architecture specification takes into account all the experience of all the consortium members and their experience both in the renewable energy sector and the software development and industry.

This architecture specification guarantees the 24x7 service, something which is needed in the offshore market and following the internationalization status of the stakeholders of this market, it results mandatory. In the same way a high performance computing environment is chased allowing a real time IoT platform and accessible to different computing layers with a single visualization point.

The main modules and architecture philosophy are shown in the figure 1, which covers the whole architecture and how the different modules are connected:

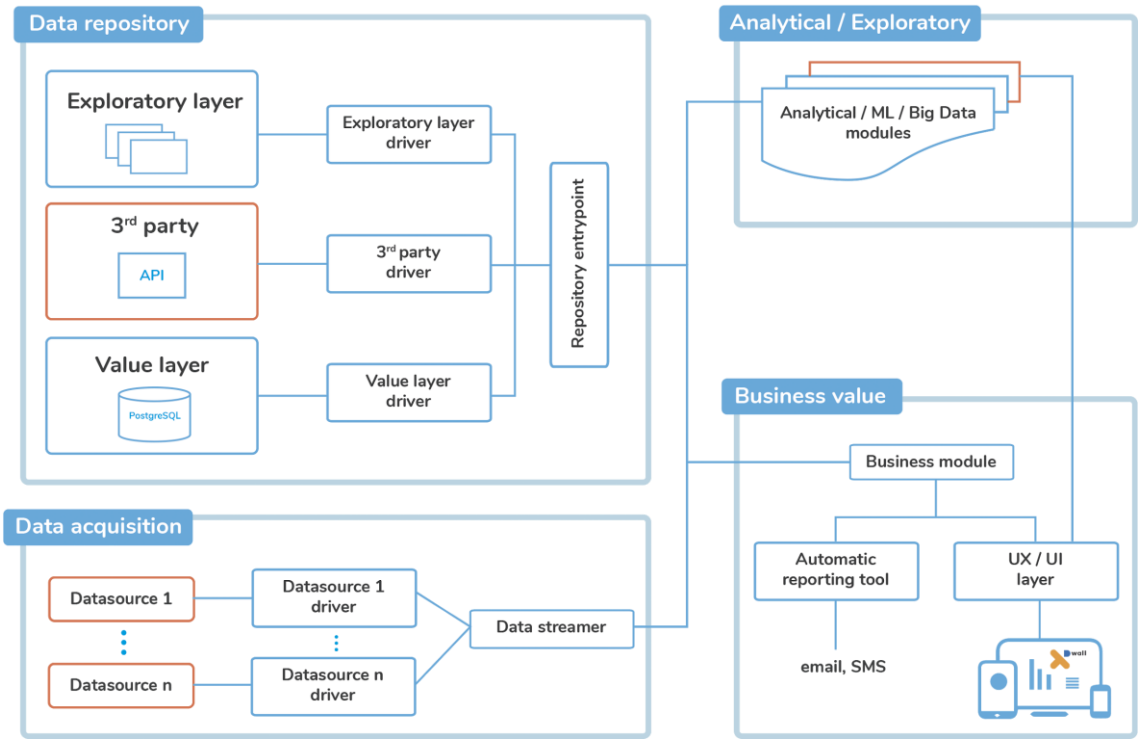


Figure 1. Logical blocks and diagram

2.1 SPECIFICATION AND CHARACTERISTICS OF EACH MODULE

2.1.1 DATA ACQUISITION

To ensure a good adaptability and interoperability of the platform with different kinds of data sources, the specification includes a way to encapsulate the data acquisition process into a software piece from third parties as data providers (DataSource driver).

The main duty of this component is to guarantee a communication flow and interactivity between the different data sources with a formatting process to ensure that all the data which is uploaded to the architecture respects the same format. The digital platform will have a single representation mode and all the data which is uploaded must be adapted to this format to ensure the interoperability and openness that it has been seeking.

Once the data has the required format to be manipulated by the platform, the Data Streamer component will be the one in charge of sending the data and uploading it to the Data Repository. In this Data Repository, all the data will be stored.

The Data Streamer component ensures that any additional data from any new data source is viable just with a new Data Streamer component. The scalability is a very important feature and this specification takes the acquisition process into account. The Data Streamer will allow a horizontal scale process and the architecture will always have the option to grow with new data streamers if a bandwidth or server limitations occur (during the DOCC-OFF project it is not expected to reach this kind of limitations, but the specification and development are focused on a real scenario with all the subsystems and assets that an offshore operator could have). This way the architecture allows a non-dependency of the data source with a single system of data representation into the platform for all the users.

Finally, another advantage of this specification of the data acquisition is the possibility to combine the data for different kinds of duties. For example, for data dashboarding as KPIs or as data lake for third parties. The main and unique format is guaranteed not just for the digital platform but for all the data consumers, even for third parties or applications, too.

2.1.2 DATA REPOSITORY

This block is formed by three logical components giving answers to different objectives. To do that, the proposed specification includes different technological solutions for each one of them, in order to ensure a good interoperability.

A datapoint could be persisted in more than one of the logical components if the datapoint will be used for several purposes. For example, if the same datapoint of raw data is used to feed an algorithm but at the same time is used to feed a current status dashboard, this data could be stored in the exploratory layer as well as in the value layer. This approach takes into account the findable, accessible, interoperable and reusable attributes of the data governance.

The Data Repository exposes a single input point which abstracts the rest of the architecture of where the data is persisted. With this specification the growth of the digital platform is very flexible and scalable with an interoperability guarantee without any kind of affectation to other components of the digital architecture. This means that following this specification, the growth of each module or architecture component is decoupled from the others and each of them could grow autonomously without any kind of influence and modification needs in the other modules.

Finally, thanks to this approach, the Business Value Module that will be described in the next section (see Business Value section), could manage different kinds of functionalities and features at the same time (monitoring of current status behaviour or management dashboard together with deep data exploration).

Value layer

This is the persistence layer where the data related with the decision-making process is stored. This is the point where the user will understand which action should be taken for the good of the monitored asset, and the module links all the data needed for each diagnostic. All aggregated data coming from algorithms, data modelling techniques, hybrid modelling, etc will end here, when the final user understands what is going on and which kind of results offer the different models. Following this goal, a relational database is chosen to solve this problem and relate different data with assets and subsystems.

This layer will contain as well all the information and attributes which are necessary for different users role management, task programming or data meta information needs among others utilities kind characteristics.

Exploratory layer

This block will be oriented to the research work with the available data. All the data which could be used for analytical or research purposes (both time series or correlations) will be stored in this layer taking into account that calculations or model over massive data volumes should be addressed.

3rd party layer

As confidentiality is a very important aspect that has to be taken into account, the specification of the digital architecture includes a layer to guarantee a way to use some data without sharing it into the global architecture for other users or stakeholders. In the same way, this 3rd party layer will allow the connection with other local or edge layers in addition to the cloud layer to enrich the capabilities of the digital platform.

Following this specification, the digital platform could provide a connection between confidential data to run and feed models or algorithms, but without sharing the raw data and storing it in the digital platform. In this case a customized driver will be defined (in it is necessary) to connect with the third-party system or database with a cybersecurity approach guaranteeing this confidentiality. This approach could be useful as well to insert third parties components such as edge or local layers or business intelligence frameworks into the digital platform as a tool to report some of the stored data and the results of other layers computing.

2.1.3 ANALYTICAL / EXPLORATORY

The analytical modules will be worked with two perspectives depending the business case or case of use:

Analytical modules without integration into the digital platform

These modules will run third parties calculations autonomously and will expose the output through an API or data access automated system which will allow the integration as a new data source. This option runs the third party's analytical modules and allows an execution management scenario for them, but it provides just the output data to the rest of the digital platform. Again, a connection between other computing layers and the cloud and main layer is done with this module, too.

Analytical modules with integration into the digital platform

If the integration is full, the module should be interactive for the users and this interaction will be integrated into the UX/IU layer of the architecture, including at least the next features:

- To check the available features or actions for the user.
- To launch parametric actions or commands.
- To manage the current status of all the running processes.
- To operate (stop, run) these processes.
- A notification system to understand the relevant info regarding long term processes behaviour.

2.1.4 BUSINESS VALUE

This logical block will be the core module of the digital platform for the interactivity with the users and its management. The orchestration between the three layers that will be described next guarantee the correct performance of the digital platform from a user perspective. A single access point is guaranteed for the final user no matter the computing layer or data input.

Business module

This module manages all the role and permissions tasks, including scheduled tasks such as reporting, running the models in the proper moment or time, and coordinating all the data input and output towards the data repository.

As the rest of the specified modules, if the amount of users, schedules tasks and data input/output grows, this module could grow, too, with independence to the rest of the modules of the digital platform. This module has a stateless specification orientation to ensure the horizontal scalability based on the system needs and loads.

UX/UI module

This model hosts the user interface, with a responsive design specification to allow the interaction from a web explorer from a laptop, tablet or smartphone.

3. TECHNOLOGICAL STACK

The proposed specification is focused on the architecture of a digital platform to explore, analyse and use data for several challenges through a web explorer. Thanks to the proposed architecture, the digital platform will be agnostic with a single requisite: to run over Linux CentOS 7 servers to make it available an automatic setup and future deployments. In the same direction, the suggested relational database over PostgreSQL could be run installed in the server or as a third-party service such as Amazon RDS or Google Cloud SQL.

The offered flexibility and scalability are plausible in every element of the architecture.

3.1 BACKEND

The Digital platform backend will be developed in Java using Springboot[1] as a framework.

Dependence management and the code building process will be done using Maven[2]. This ensures a well-defined and repetitive building process.

3.2 FRONTEND

JavaScript using Vue.js[3] will be the selected scenario for the front-end environment.

CSS will be written using Sass[4].

All the front-end code will be managed by Webpack[5].

Finally, xabet owns a design system which will be used for the development of the digital platform front end. These elements of xabet's design system will allow a quicker development and scalability together with high flexibility for the rest of the consortium partners in terms of front-end needs.

3.3 PERSISTENCE

Digital platform's persistence will be done using PostgreSQL[6] (9.6). The database structure change management will be orchestrated using Flyway[7] . This way we ensure a robust process with a sorted and repetitive task management. The backend code which will map the database to objects will be

[1] https://spring.io/projects/spring-boot
[2] https://maven.apache.org/
[3] https://vuejs.org/
[4] https://sass-lang.com/
[5] https://webpack.js.org/
[6] https://www.postgresql.org/
[7] https://flywaydb.org/

jOOQ[8], which generates Java code from the database structure avoiding incoherence into the source code when changes into the database occur.

3.4 INFRASTRUCTURE CONFIGURATION

All the execution environments setup of the digital platform is done using Ansible[9] (operative system, virtual machines, http server, SSL certificates, evil building process…). And the digital platform will have a monitoring system and servers resources alert system to ensure a predictive future needs to guarantee a good performance, and as a troubleshooting tool. This will be done using Munin[10].

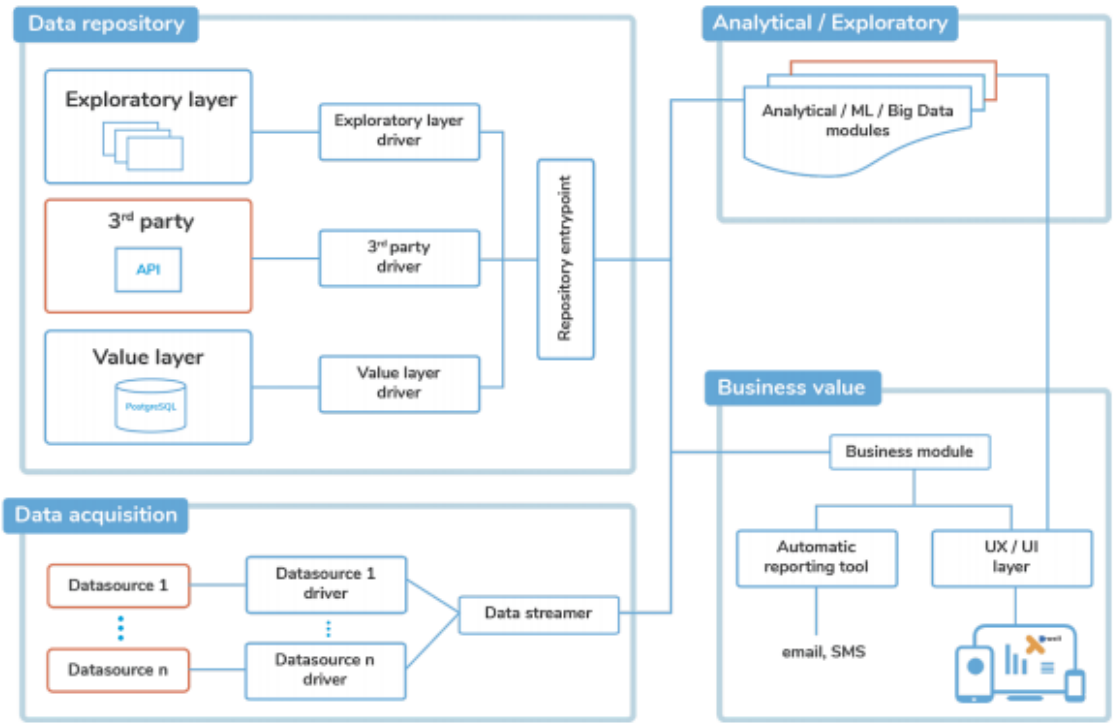
[8] https://www.jooq.org/

[9] https://www.ansible.com/

[10] http://munin-monitoring.org/

4. COMMUNICATION BETWEEN MODULES. A TECHNICAL DESCRIPTION OF DATA FLOW.

The following figure shows again the structure and architecture of the digital platform to fulfil all the needs reported by the DOCC-OFF consortium partners and their objectives.



Following the description defined previously, the data acquisition module will manage data which occurs in a single and defined moment with the next attributes:

- Variable identifier
- Timestamp of the datapoint
- Value of the datapoint

During this section, the specification of the communication between modules and the way to manage add-ons, modules and features will be described. There is a part of the architecture called CORE which will contain the main modules and features of the digital platform. These main modules will be those that are transversal for any kind of company/partner. And there is another site of code for customization or CUSTOM features which will offer specific solution or tool to specific problems or scenarios, given flexibility and openness with a way to connect this architecture easily with any kind of software piece.

Next each module of the architecture will be presented from a perspective of what is CORE and what is CUSTOM to understand how the architecture is developing from a technical and communication perspective.

4.1 DATA ACQUISITION

4.1.1 CORE

To send data:

The libraries in charge to send data through http will be in the CORE part. Currently Java and Python libraries will be defined as the main libraries that will be used during the DOCC-OFF project but other libraries could be added if the consortium and the project need it, with custom data connectors.

Modules:

`dwall-core / dwall-http-client-library (Java)`

`dwall-core / dwall-http-client-library-python (Python)`

To receive data:

The CORE will be prepared for single-delivery (one data point in one timestamp) and multi-delivery (multiple data points in multiple timestamps) of data. The code will be hosted as follow:

`dwall-core / dwall-app-boot / .... / controller / ingest / IngestVariableController`

4.1.2 CUSTOM

To send data:

Whenever a new delivery of data appears, the architecture will host as it via driver as is described in the section **OVERVIEW OF THE DIGITAL ARCHITECTURE**

Each time new data is included into the research or the digital platform an analysis of each datapoint source typology is done and at the time to send the data all the needed implementation is done through the mentioned libraries using http and based on the needs of the data.

This autonomous pieces of software with the duty of read and send datapoints into the Digital Platform are DRIVERS. These drivers could be developed in any kind of programming language if they follow the defined API (Application Programming Interface).

As the customization will be done by partner, customer or brand, and understanding that they will be very specific for each company or brand, a specific place to include all this information will be prepared as follows:

`dwall-brand / brand-name / name-module-driver`

The “driver” suffix is added to these modules to make it easier the understanding of the different pieces of customization that could be developed for each brand.

To receive data:

If the data source offers the data through an API REST (Representational State Transfer) and the delivery mode does not maintain the standard of the digital platform, this architecture will have the capability to define endpoints for the API REST to adjust to the data contract of the sender.

These modules should be written in Java and this will be the place to host them:

```
dwall-brand / name-brand / dwall-app / .... / controller / NombreDelController
```

4.2 DATA REPOSITORY

4.2.1 CORE

As it is described in **OVERVIEW OF THE DIGITAL ARCHITECTURE**, the main objective of this module is the abstraction of the data source. Currently the abstraction for the main DATABASE (PostgreSQL) is adopted in the projected specification but other systems or data sources time series could be abstracted, too, using this module, to ensure the communication of the data between the different modules.

All the code related to this concept will be at:

```
dwall-core / dwall-persistence
```

All the API requirements to be connected with the rest of the modules will be at:

```
dwall-core / dwall-persistence / … / api /
```

This way, as it is described in **OVERVIEW OF THE DIGITAL ARCHITECTURE,** the implementation of the data persistence can be adjusted to each case without any kind of effects or influence to the business logic and layer (Business value). These implementations will be done here:

```
dwall-core / dwall-persistence / ... / repository /
```

Flyway will be used to manage all the changes into the database structure and the migration will be saved in the next route of the architecture code:

```
dwall-core / dwall-persistence / resources / db / migration
```

Following a scheme of consecutives numbers with prefixes such as V001_…, V002_…, Flyway executes all the migrations that are needed at the application launching time (see Flyway[11] for more technical information).

4.2.2 CUSTOM

If a specific partner or brand needs a customization of the persistence module with a new data source or with an addition of tables to the main database, all these changes will be:

[11] https://flywaydb.org/documentation/

`dwall-brand / name-brand / dwall-persistence /`

As it is planned for the CORE area, all the customization over the database will be managed by Flyway and will be hosted here:

`dwall-brand / name-brand / dwall-persistence / resources / db / migration`

To avoid conflict with the persistence of CORE modules, all the customizations will be named with the prefix: `brand_`.

4.3 BUSINESS VALUE

4.3.1 CORE

Business module:

All the issues regarding digital platform and workspace management will be hosted here:

`dwall-core / dwall-app-boot`

4.3.1.1 WEB INTERFACE:

The front-end code will be written following webpack and Vue.js standards and will be hosted here:

`dwall-web`

More detail in Webpack[12], Vue.js[13] y Sass[14]

4.3.1.2 NOTIFICATION SYSTEM

Sending emails is one of the features that the digital platform will have. To do this task a commercial service will be adopted: Amazon SES[15] . All the code related to these implementations and the governance of emails and personal data will be hosted here:

`dwall-core / service / batch / EventEmailNotificator`

Other email delivery services could be used if wanted. As the rest of the specified architecture, flexibility and modularity are present. For example, other kinds of service that could be adapted or implemented are SMS, Telegram, Slack, monitoring systems, etc. The service will be added to this module to launch it.

Scheduled tasks

There are two types of scheduled tasks:

- Execution programmer

[12] https://webpack.js.org/guides/getting-started/
[13] https://vuejs.org/v2/guide/
[14] https://sass-lang.com/documentation
[15] https://aws.amazon.com/ses/

- To do task

The job and task scheduler are hosted here:

`dwall-core / dwall-app-boot / … / JobsScheduler`

As it is described in section 2, Springboot will be used for this task, and it will be prepared as well to use a queuing system as a requirement if it is needed.

4.3.2 CUSTOM

4.3.2.1 BUSINESS MODEL

If an ad-hoc feature is needed, it will be hosted as a specific element of the brand here:

`dwall-brand / brand / dwall-app`

4.3.2.2 WEB INTERFACE

Customization for specific dashboard or ad-hoc screens for each brand will reproduce the main structure of the CORE project mixing it into a single contract. It will be hosted here:

`dwall-web / src / brand / brand-name`

Scheduled task

This could make sense in the case of a specific needs of a brand to schedule specifics task. Following the flexibility and modularity of the entire architecture, if a specific task scheduler or manager is needed it will be hosted here and it will be specific for the brand:

`dwall-brand / dwall-app / … / BrandJobsScheduler`

4.4 ANALYTICAL/EXPLORATORY

4.4.1 CORE

All data entry into the system, even when the data is generated into the digital platform will be managed as it is described in the Data Acquisition module. The approach is always the same along the entire architecture to send and receive data.

4.4.2 CUSTOM

For those cases where a customization is needed due to the presence of a third party which does not follow Digital Platform's API REST, a customization will be addressed as it is shown previously in Data Acquisition and Data Repository.

5. STRUCTURE OF THE DATABASE

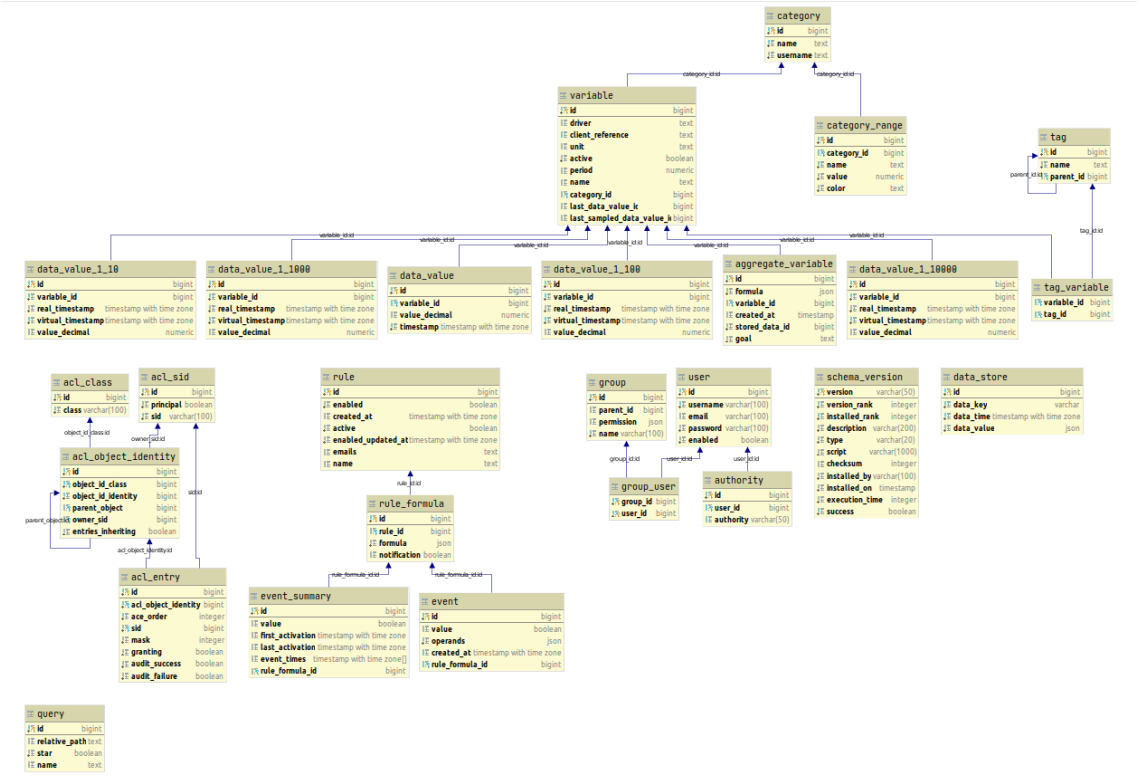


Figure 2. Database structure

The digital platform is built over a PostgreSQL Database. The database structure evolves constantly to adapt new CORE demands and it allows customization as well as it is described in the previous section.

5.1 TABLES STRUCTURE

For further information about the used Spring Security for privileges and permissions management, the official documentation of the service could be checked here: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/domain-acls.html.

acl_class: register of the names for the domain objects class

Fields:

-id: self-increasing identifier.

-class: secured domain's object.

acl_sid: role and user registration

Fields:

-id: self-increasing identifier.

-sid: user name or role.

-principal: boolean which says if it is a role or a user.

acl_object_identity: information registration of each domain unique object.

Fields:

-id: self-increasing identifier.

-object_id_class: domain object class identification (links with acl_class).

-object_id_identity: Unique object identifier within the domain of the rest of the database.

-parent_object: parent specification towards the same table.

-owner_sid: object property identification (links with acl_sid).

-entries_inheriting: indicates if ACL entries (tabla acl_entry)inherit from parent.

acl_entry: user permissions storage for each SID related to the Object Identity.

Field:

-id: self-increasing identifier.

-acl_object_identity: object identity identification (acl_object_identity table).

-ace_order: sorted by Object identity into the ACL entries.

-sid: sid which applies (links with table acl_sid).

-mask: mask of permissions to be associated with.

-granting: granted(1) o denied (0).

-audit_success: auditory registration.

-audit_failure: auditory registration.

5.1.1 TABLES RELATED WITH INTERNAL PERMISSIONS AND PRIVILEGES

group: user groups

Fields:

-id: self-increasing identifier

-parent_id: parent group identifier

-permission: json with the enable features for the group

-name: name of the group.

user: users of the digital platform

Field:

-id: self-increasing identifier

-username: user name

-email: user email

-password: user password

-enabled: status of the user account

group_user: user-group relationship

Fields:

-group_id: group identifier

-user_id: user identifier

authority: permissions granted by the user

Fields:

-id: self-increasing identifier

-user_id: user identifier

-authority: roles and functions granted for the user

5.1.2 TABLES RELATED WITH THE RULES AND EVENT MANAGEMENT

rule: rules created by the user

Fields:

-id: identificador self-increasing

-enabled: rule enable

-created_at: date of rule creation

-active: rule status

-enabled_updated_at: last enable status by the user

-emails: receptors of emails generated by the rules

-name: name of the rule

rule_formula: mathematical rule definition

Fields:

-id: self-increasing identifier

-rule_id: rule associated with the formula

-formula: mathematical formula to apply

-notification: in case of notification with each activation

event: rules activation registration

Fields:

-id: id self-increasing

-value: enable/disable

-operands: json with the operator values with presence in the formula

-created_at: date of the event (rule fulfillment)

-rule_formula_id: id of the activated formula

event_summary: list of event activations and deactivations

Fields:

-id: self-increasing identifier

-value: true/false

-first_activation: date of the first activation

-last_activation: date of the last activation

-event_times: number of repetitions of the activation for the same event

-rule_formula_id: id of the activated formula

5.1.3 TABLES RELATED WITH VARIABLES MANAGEMENT AND THEIR DATA

variable: variable definition

Fields:

-id: self-increasing identifier

-driver: identifier text for the data source

-client_reference: unique identifier for the brand or client

-unit: variable measurement unit

-active: variable status for visualization

-period: signal time lapse in seconds

-name: nickname in the digital platform for the variable

-category_id: link with the category which includes the variable (if it applies for the variable)

-last_data_value_id: last value for the ID ofdata_value table for the last inserted datapoint.

-last_sampled_data_value_id: last sampled value for the ID ofdata_value table for the last inserted datapoint.

category: categorical variables definition

Fields:

-id: self-increasing identifier

-name: name of the categorical variable

-username: user who creates the categorical variable

category_range: values of each categorical variable

Fields:

-id: self-increasing identifier

-category_id: relation with category

-name: text to be linked to the value

-value: value

-color: the color which will be used for visual representation of the categorical variable.

aggregate_variable: aggregate variable definition

Fields:

-id: self-increasing identifier

-formula: json with the mathematical formula for the aggregate variable

-variable_id: id of the aggregate variable in the variable table

-created_at: creation date

-stored_data_id: last id of the first operator of the aggregate variable in the table data_value

-goal: type of aggregated variable, "operated" for calculations between variables and constant and "gradual" for calculations between the same variable in different timestamps.

tag: labels or tags to be applied to variables

Fields:

-id: self-increasing identifier.

-name:tag name.

-parent_id: tags are defined following a tree structure. This filed represents the id of the parent tag, if it si null meas that it is the parent label.

tag_variable: relation variable-tags

Fields:

-variable_id: variable identifier.

-tag_id: tag identifier.

data_value: reads of the variables values

Fields:

-id: self-increasing identifier.

-variable_id: variable identifier.

-value_decimal: reading value.

-timestamp: reading timestamp.

data_value_1_10: abbreviation table for data sampling (1 of ten data points)

Fields:

-id: self-increasing identifier.

-variable_id: variable identifier.

-real_timestamp: value timestamp.

-virtual_timestamp: virtual timestamp.

-value_decimal: reading value.

data_value_1_100: abbreviation table for data sampling (1 of 100 data points)

Fields:

-id: self-increasing identifier.

-variable_id: variable identifier.

-real_timestamp: value timestamp.

-virtual_timestamp: virtual timestamp.

-value_decimal: reading value.

data_value_1_1000: abbreviation table for data sampling (1 of 1000 data points)

Fields:

-id: self-increasing identifier.

-variable_id: variable identifier.

-real_timestamp: value timestamp.

-virtual_timestamp: virtual timestamp.

-value_decimal: reading value.

data_value_1_10000: abbreviation table for data sampling (1 of 10.000 data points)

Fields:

-id: self-increasing identifier.

-variable_id: variable identifier.

-real_timestamp: value timestamp.

-virtual_timestamp: virtual timestamp.

-value_decimal: reading value.

data_value_[aaaammdd] (it will apply just if partition of the table data_value is needed)

5.1.4 TABLE FOR GENERIC INFORMATION STORAGE

data_store: non structured information storage. It will be used when a relational database is not the best option or if it does not apply to the problem to be solved.

Fields:

-id: self-increasing identifier.

-data_key: key (defined by the developer) to identify the storaged readings.

-data_time: timestamp associated with the data point.

-data_value: json with the associated information to the timestamp.

5.1.5 TABLE FOR QUERIES STORAGE:

query: queries that the user generates into the exploratory tool available in the digital platform.

Fields:

-id: self-increasing identifier.

-relative_path: url of the query.

-star: if the query is marked as favorite.

-name: name of the query by the user.

5.1.6 REQUIRED TABLE OF FLYWAY FOR MIGRATIONS

schema_version: this table belongs to Fliway. More info at: https://flywaydb.org/getstarted/how.

5.2 CHANGE MANAGEMENT

The change management will be done with Flyway.

5.3 SCALABILITY

Two different approaches will be followed to guarantee a wide scalability of the database structure.

Datapoint reads are stored in the table data_value, and from this point the performance will be optimised using two methods: downsampling and partition of tables.

5.3.1 DOWNSAMPLING

The tables data_value_1_10, data_value_1_100, data_value_1_1000, data_value_1_10000 are tables with sampled data of the table data_value with different resolutions specified by the suffix (10, 100, 1000…. which means 1 data sampling from 10, from 100, or 1000 for example). This resampling will be launched automatically by the digital platform once new data arrives to Data acquisition module.

For time series the resampling takes into account the higher and lower data value.

For correlations two fixed points for the max and min value are selected, to guarantee that the system has data from different variables at the same timestamp.

5.3.2 TABLE PARTITION

When the performance of any of the tables data_value* is affected negatively by the storage data volume, the table is partitioned. This partition is done in a transparent way, activating a flag in the repository package setup and both reading and writing processes are optimised.

5.4 THIRD PARTIES INTEGRATION

As it is described in the overview of the architecture of the digital platform, an API REST is defined to allow third parties integrations.

For the DOCC-OFF project, the next two libraries are defined: Java and Python:

dwall-core / dwall-http-client-library (Java)

dwall-core / dwall-http-client-library-python (Python)

If the data integration requires different libraries for the data acquisition of a third party, new endpoints will be defined, or even a specified driver could be developed. In this case, these developments will be hosted here:

```
dwall-brand / name-brand/ dwall-app / … / controller / nombre_controller.java
```