SOMIOD

Integração de Sistemas

Ângelo Augusto  
Politécnico de Leiria | ESTG –

Escola Superior de Tecnologia e

Gestão  
Leiria, Portugal2201794@my.ipleiria.pt

Pedro Nascimento  
Politécnico de Leiria | ESTG – Escola Superior de Tecnologia e Gestão  
Leiria, Portugal2201774@my.ipleiria.pt

Humberto Ferreira  
Departamento de Engenharia Informática  
Politécnico de Leiria | ESTG – Escola Superior de Tecnologia e Gestão  
Leiria, Portugalhumberto.ferreira@ipleiria.pt

André Marques  
Politécnico de Leiria | ESTG –

Escola Superior de Tecnologia e

Gestão  
Leiria, Portugal2200701@my.ipleiria.pt

Marisa Maximiano  
Departamento de Engenharia

Informática  
Politécnico de Leiria | ESTG –

Escola Superior de Tecnologia e

Gestão  
Leiria, Portugalmarisa.maximiano@ipleiria.pt

*Abstract*—This document describes the development of a middleware platform that aims to keep consistency in the process of sharing data, promoting the efficiency in the communication between applications. We present the architecture used in the development and explore the challenges associated with creating a secure platform.

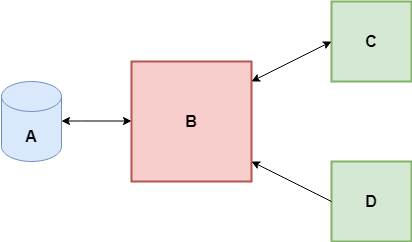
Keywords—component, formatting, style, styling, insert

# Introduction

Nowadays, to share data across different applications, most IoT solutions use private unique protocols with different formats, which has an impact on the interoperability. SOMIOD is a middleware that aims for the promotion of the consistency of how data is shared across different devices by allowing the creation, access, modification and deletion of resources to always be done in the same way and format.

# System Architecture

In the development of the SOMIOD, a controller, a database and two testing applications were used, as presented in the following architecture.



1. System architecture

## Database

This is a relational database and it’s where all the data related to the created resources is stored. The tables that exist in the database are the following.

Table I - Application

|  |  |
| --- | --- |
| Id | Identification number of the application |
| Name | Name of the application |
| Creation\_dt | Creation date of the application |

Table II - Module

|  |  |
| --- | --- |
| Id | Identification number of the module |
| Name | Name of the module |
| Parent | Identification number of the parent resource of the module |
| Creation\_dt | Creation date of the module |

Table III – Data

|  |  |
| --- | --- |
| Id | Identification number of the data |
| Content | Content of the data |
| Parent | Identification number of the parent resource of the data |
| Creation\_dt | Creation date of the data |
| Deletion\_dt | Deletion date of the data |

Table IV - Subscription

|  |  |
| --- | --- |
| Id | Identification number of the subscription |
| Name | Name of the subscription |
| Creation\_dt | Creation date of the subscription |
| Parent | Identification number of the parent resource of the subscription |
| Event | Event triggered by the subscription |
| Endpoint | Endpoint where the event will be triggered |
| Deletion\_dt | Deletion date of the subscription |

## ProjectController

This is a controller that manages the routes that allow the communication either between the database and existing client applications or between the different client applications.

## Application A

This is one of the testing applications that allows the creation of applications, modules and subscriptions; update of applications and modules and deletion of applications and modules. This application also gets the status of 2 light bulbs from the database. It is composed by the following screens.

Diagram, schematic

Description automatically generated

Fig. 2. Status of first light bulb

Diagram

Description automatically generated

Fig. 3. Status of second light bulb

Graphical user interface

Description automatically generated

Fig. 4. Screen for the CRUD operations

## Application B

This is the other testing application that allows the user to make a request to the controller that updates the status of the corresponding light bulb and then sends an Mqtt (Mosquitto) message to the application A.. It is composed by the following screens.

Graphical user interface, application

Description automatically generated

Fig. 5. Screen for changing the status of the first light bulb

Graphical user interface, application, PowerPoint

Description automatically generated

Fig. 6. Screen for changing the status of the second light bulb

Graphical user interface, application

Description automatically generated

Fig. 7. Screen for changing the status of the selected light bulb

# Evaluation

## Test bed

Our testing environment is composed by the two applications that were previously mentioned. In the application A, in terms of creation, we can create an application, a module for an application and a subscription for a module. In terms of update, we can update an application’s name and a module’s name. In terms of deletion, we can delete an application only if it doesn’t have any modules associated to it and we can delete a module even if it has subscriptions associated to it. In the application A we can also check the status of one of the light bulbs, which can be either “On” or “Off”, represented by the image of a lit light bulb and a not lit light bulb, respectively.

In the application B, we can update the status of either one of the light bulbs in the screen that corresponds to one of them or in the screen that allows the selection of the light bulb to be updated.

When the status of a light bulb is updated, the application makes a request that updates the status in the database, and then sends an Mqtt (Mosquitto) message to the application A, which will then proceed to make a request that gets the new status of the updated light bulb and changes the image according to that status. Data analysis

# Integration/App Development

## Application A

In this application, we can see the status of two light bulbs. When an Mqtt (Mosquitto) message from the application B is received due to the update of the status of one of the light bulbs, this application makes a request to the controller, gets the new status from the database and updates the image of the corresponding light bulb according to the status. We can also make the CRUD operations of the resources in this application.

On the CRUD operations of the resources, the deletion of an application is only successful if the selected application has no modules. The deletion of a module is always possible even if there is data associated to it.

## Application B

In this application, we can fire events that either turn on or off one of the light bulbs. This can be done in the screen that corresponds to one of the light bulbs or in the screen where it’s possible to select the light bulb. When an event is fired, a request to the controller is made, the status of the given light bulb is updated in the database and an Mqtt (Mosquitto) message is sent to the application A.

# Conclusions and Future Work

The development of this middleware gave us a broader insight into how a middleware works and how data can be shared across different platforms using a consistent format. It included well-established topics that we didn’t know much about and allowed us to conclude that the consistency in the sharing of data is very beneficial in terms of reliability and accessibility, resulting in an increase in the scalability of the middleware.

# references

1. There are no References, only the material that has been given during class.

# Appendix

*Appendix A*

The following routes are the ones provided by the controller.

**Create application:** “api/somiod/”

* **Curl:**

curl --location --request POST 'http://localhost:55645/api/somiod/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Application>

    <Res\_type>application</Res\_type>

    <Name>App1</Name>

</Application>'

**Get all applications:** “api/somiod/”

* **Curl:**

curl --location --request GET 'http://localhost:55645/api/somiod/' \

--header 'Content-Type: application/xml' \

--data-raw ''

**Edit application:** “api/soimiod/{appname}”

* **Curl:**

curl --location --request PUT 'http://localhost:55645/api/somiod/App1/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Application>

    <Res\_type>application</Res\_type>

    <Name>App1</Name>

</Application>'

**Delete application:** “api/somiod/”

* **Curl:**

curl --location --request DELETE 'http://localhost:55645/api/somiod/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Application>

    <Res\_type>application</Res\_type>

    <Name>App1</Name>

</Application>'

**Create module:** “api/somiod/{appname}”

* **Curl:**

curl --location --request POST 'http://localhost:55645/api/somiod/App1' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Module>

    <Res\_type>module</Res\_type>

    <Name>Module1</Name>

</Module>'

**Get all modules by application:** “api/somiod/{appname}”

* **Curl:**

curl --location --request GET 'http://localhost:55645/api/somiod/Module1' \

--header 'Content-Type: application/xml'

**Edit module:** “api/somiod/{appname}/{modname}”

* **Curl:**

curl --location --request PUT 'http://localhost:55645/api/somiod/App1/Module1' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Module>

    <Res\_type>module</Res\_type>

    <Name>Module1</Name>

</Module>'

**Delete module:** “api/somiod/{appname}”

* **Curl:**

curl --location --request DELETE 'http://localhost:55645/api/somiod/App1/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Module>

    <Res\_type>module</Res\_type>

    <Name>Module1</Name>

</Module>'

**Create data:** “api/somiod/{appname}/{modname}”

* **Curl:**

curl --location --request POST 'http://localhost:55645/api/somiod/App1/Module1' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<**Data**>

    <Res\_type>**data**</Res\_type>

    <Content>on</Content>

</**Data**>'

**Delete data:** “api/somiod/{appname}/{modname}”

* **Curl:**

curl --location --request DELETE 'http://localhost:55645/api/somiod/App1/Module1/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<**Data**>

    <Res\_type>**data**</Res\_type>

    <Id>430</Id>

</**Data**>'

**Create subscription:** “api/somiod/{appname}/{modname}”

* **Curl:**

curl --location --request POST 'http://localhost:55645/api/somiod/App1/Module1/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Subscription>

    <Res\_type>subscription</Res\_type>

    <Name>Subscription1</Name>

    <Endpoint>127.0.0.1</Endpoint>

</Subscription>'

**Delete subscription:** “api/somiod/{appname}/{modname}”

* **Curl:**

curl --location --request DELETE 'http://localhost:55645/api/somiod/App1/Module1/' \

--header 'Content-Type: application/xml' \

--data-raw '<?xml version="1.0" encoding="UTF-8"?>

<Subscription>

    <Res\_type>subscription</Res\_type>

    <Name>Subscription1</Name>

</Subscription>'

*Appendix B*

The project has been divided by our classmates; however, one of the four elements wasn’t present during its working progress. So, we decided to remove our fourth element.

To run the project, and use the apps that are already created, first change the *endpoint* that are in the database to an existing *mosquitto* endpoint, and in the *App A* on the *Form1* and *Form3*, change to the same endpoint. Second, change in *App A* and *App B* the *baseUrl* to the port that the project is using. Third, just press start in the *Microsoft studio*.