Relatório Projeto IS

1st Diogo Guerra  
*dept. ESTG*  
IPLLeiria, Portugal  
2211027@my.ipleiria.pt

4th Válter Pedrosa  
2: *dept. ESTG*  
*IPL*Leiria, Portugal  
2210991@my.ipleiria.pt

2nd Lucas Remigio  
*dept. ESG*  
*IPL*Leiria, Portugal  
2211034@my.ipleiria.pt

5th Marisa Maximiano  
*dept. ESTG*  
*IPL*City, Country  
marisa.maximiano@ipleiria.pt

3rd Marco Ferreira  
*dept. ESTG*  
*IPL*Leiria, Portugal  
2210995@my.ipleiria.pt

6th Humberto Ferreira  
*dept. ESTG*  
*IPL*City, Country  
humberto.ferreira@ipleiria.pt

*Abstract*—This document blá, blá, blá

This is the abstract of the paper…..

(Remember, a good abstract is equal to Zip(Introduction) + Zip(Conclusion). So, this is the last section to be written. Delete this. Always try to keep formatting present in the document. Delete this.).

Keywords—component, formatting, style, styling, insert

# Introduction

No âmbito da disciplina de Integração de Sistemas (IS) foi nos proposto criar um Service Oriented Middleware for Interoperability and Open Data (SOMIOD) com a finalidade de construir um middleware capaz de definir uma uniformização na forma como os dados são acedidos, escritos e notificados, independentemente do domínio da aplicação para isso utilizamos 3 protocolos muito bem conhecidos e com uma vasta documentação que são o xml para o envio de informação ,o http para a comunicação entre as aplicações e o MQTT para o envio de notificações, isto para ajudar no crescente aumento de dispositivos de Internet of Things (IoT) surge assim a necessidade destes dispositivos se comunicarem entre si ou com outros dispositivos, atualmente este processo de comunicação é dificultado pois está dependente de protocolos privados e de cloud services privados que tem um impacto negativo.

O projeto está a ser desenvolvido em C# e é composto por um SOMIOD e duas aplicações de teste: a APP\_A, que simula uma lâmpada inteligente, e a APP\_B, que representa um interruptor. Adicionalmente, foram criadas três aplicações específicas, uma para cada recurso suportado pelo SOMIOD: Applications, Controllers e uma aplicação para Datas e Subscriptions. Estas aplicações de teste permitem avaliar todas as opções de CRUD do SOMIOD.

# System Architecture

Uma imagem com texto, diagrama, captura de ecrã, Esquema

Descrição gerada automaticamente

Figura 1 – Arquitetura do sistema desenvolvido

## Base de dados

A utilização de um SOMIOD cria a necessidade de guardar os recursos que são criados pelas aplicações tais como a APP\_A, APP\_B e as 3 aplicações de testes que depois podem ser lidos, atualizados e eliminados posteriormente.

Decidimos criar uma base de dados com 4 tabelas em que em cada tabela será guardada a informação de cada recurso por isso temos uma tabela: Application, Container, Subscription e Data.

A hierarquia dos recursos está de modo que uma plicação possa ter vários containers que por sua vez podem ter várias subscriptions e datas como mostra a Figura 2, decidimos por tanto que ao eliminar uma aplicação por exemplo todos os containers e por sua vez todas as datas e subscriptions seriam também eliminados sendo assim implementando uma eliminação em cascata. Esta decisão foi feita pois na nossa opinião se para eliminar uma aplicação fosse necessário eliminar cada container que por sua vez para ser eliminado tinha de se eliminar cada data e subscription iria demorar muito tempo pois a tabela data pode crescer em grandes proporções.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 2 Hierarquia dos recursos

### Application

A tabela application tem o propósito de guardar todas as applications que serão criadas para posteriormente em cada application serem criados um ou mais containers.

A tabela application contém apenas 3 colunas sendo estas o id que é único e vai ser a chave primária desta tabela e é criado automaticamente pelo SOMIOD quando se cria uma application, um name que é enviado para o SOMIOD por um método POST e que também tem de ser único e que vai servir para identificar posteriormente esta aplicação para a criação de containers, subscriptions e datas e por fim uma coluna chamada creation\_dt que vai ter a data e hora de quando a aplicação foi criada seguindo o formato de 2023-09-25 12:34:23.

Uma imagem com texto, Tipo de letra, número, captura de ecrã

Descrição gerada automaticamente

Figura 3 Estrutura da tabela Application

### Container

A tabela container tem como objetivo guardar as informações sobre cada container para isso a tabela conta com 4 colunas onde as primeiras 3 são iguais ás da tabela application sendo estas o id que é chave primária e criado automaticamente, o name que também é único dentro da aplicação que está e irá posteriormente identificar o container para que este possa ser atualizado eliminado ou lido e a coluna creation\_dt que guarda a data e hora de quando o container foi criado. A única coluna que o container tem a mais é a coluna application\_id que irá servir para guardar o id da aplicação em que este container foi criado.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 4 Estrutura da tabela Container

### Data

A tabela data guarda a informação de todas as datas que são criadas num container estas datas contêm a informação que deve ser passada entre aplicações, para isso a tabela data conta com 5 colunas sendo estas: o id que é o identificador único e a chave primária da coluna e é criado automaticamente, um name que também é único dentro do container em questão, um creation\_dt que guarda a data e hora em que uma data foi criada, um container\_id que identifica em que container é que esta data se encontra e um contente que é o que é enviado de uma aplicação para a outra entre aplicações este conteúdo é enviado em strings mas na base de dados foi decidido guardar o contente como um conjunto de bytes pois o contente pode ser qualquer tipo de string tal como uma string simples ou um xml por exemplo o que pode fazer com que a coluna fique com muita informação por isso foi decidido guardar apenas os bytes do content.

Uma imagem com texto, Tipo de letra, software, Software de multimédia

Descrição gerada automaticamente

Figura 5 Estrutura da tabela data

### Subscription

A tabela subscription vai guardar todas as informações sobre as subscriptions que são criadas dentro de um container para tal esta tabela conta com 6 colunas sendo estas a coluna id que guarda o identificador único de cada subscrição que é gerado automaticamente, um name que também é único dentro de cada container, um creation\_dt igual ao das tabelas anterior, um container\_id que identifica qual o container em que a subscrição foi criada, a coluna event que apenas pode conter 3 valores sendo estes 1,2 e 3 que correspondem à criação de uma data, à eliminação de uma data e a ambas as operações respetivamente e por fim a coluna endpoint que guarda o endereço para o qual o contente de uma data vai ser enviado, este endereço deve começar com mqtt:// ou http:// e de seguida o endereço para onde será enviado o content.

Todos os atributos name são unicos e case insensitive para não ocorrer conflitos com nomes em maiusculas e minusculas.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 6 Estrutura da tabela Subscription

## SOMIOD

O SOMIOD é a peça principal deste projeto sendo este responsável pelo armazenamento das applications, containers, subscriptions e data e pela sua criação, atualização e eliminação de cada recurso e pelo publish do content das datas para um broker MQTT para tal o SOMIOD usa pedidos HTTP no intuito de executar as operações de CRUD de cada recurso sendo o URI base é http://localhost/api/somiod para isso o SOMIOD foi desenvolvido no Visual Studio em C# com base em um projeto ASP.NET Web Application (.NET Framework).

Os modelos criados foram:

* Application que contem o atributo Id do tipo int, um name do tipo string e um creadiont\_dt do tipo DateTime
* Container que contem todos os atributos da application e um applicationId que é do tipo inteiro
* Data que contem todos os atribudos da application e ainda um containerId que é um int, um contente que é uma string
* Subscription que contem todos os atributos da application e ainda um containerId que é um int, um event que é um int e um endpoint que é uma string

Para estes modelos foram criados também controladores sendo estes controladores sendo estes o ApplicationController, o ContainerController e o SubscriptionAndDataController.

### Application Controller

O ApplicationController tem os métodos de CRUD de uma aplicação e ainda o método de create de um container dentro de uma aplicação para criar uma aplicação é necessário enviar um pedido HTTP POST para o endereço base, ou seja, o <http://localhost/api/somiod> onde o body do pedido http vai conter um xml com os dados para a criação de uma aplicação sendo estes apenas o nome da aplicação e um res\_type que vai identificar o tipo de recurso a ser criado.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 7 Exemplo de um xml para a criação de uma application

O SOMIOD vai receber um xml como o da Figura 7 e vai verificar se o atributo res\_type que está dentro do elemento request é igual a “application” caso o res\_type for diferente será enviada uma resposta com o HTTP Status Code de 400 a informar que o res\_type inserido não é o correto.

Em seguida o SOMIOD irá procurar o elemento name dentro do elemento application do xml e retirar a informação que será o nome da aplicação, mas antes de criar a aplicação verifica se o name não vem vazio e depois sim tenta inserir, caso já exista uma aplicação com o nome que está a ser inserido decidimos enviar uma mensagem ao utilizador com o código 400 e a mensagem a avisar que já existe uma aplicação com esse nome.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 8 Verificação do res\_type e se o name vem vazio

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 9 Inserção na base de dados e verificação se o nome é único

A atualização de uma aplicação é feita através de um pedido HTTP PUT para o endereço [http://localhost/api/somiod/{application}](http://localhost/api/somiod/%7bapplication%7d) onde application será o nome da aplicação que é suposto ser atualizada para tal no body do pedido de http tem de ser enviado o novo nome que vai ser dado à aplicação em xml que vai ser igual ao da Figura 7.

Ao atualizar uma application primeiro é verificado o res\_type e se o nome vem vazio tal como mostra a Figura 8, depois ao atualizar caso o novo nome já esteja a ser utilizado será enviado uma mensagem a informar tal como acontece na criação de uma aplicação.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 10 Código para o update de uma application

Para a eliminação de uma application apenas é necessário enviar um pedido HTTP com o método DELETE para o endereço <http://localhost/api/somiod/{application}> onde application será o nome da aplicação que é suposto ser eliminada.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 11 Código para a eliminação de uma application

No âmbito de descobrir quais são as aplicações existentes no SOMIOD é possível enviar um HTTP GET para o endereço <http://localhost/api/somiod/> mas com uma Header com o nome somiod-discover e com o valor “application” ao receber este pedido o SOMIOD verifica se o header está correto e enviar um documento xml com o nome de todas as aplicações que existem naquele momento.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 12 Verificação do Header e código para ir buscar todas as applications

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Figura 13 Criação da resposta que vai conter o nome de todas as applications.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 14 Exemplo do xml com o nome das applications

Para descobrir os detalhes de uma aplicação é necessário enviar um pedido HTTP GET para o endereço [http://localhost/api/somiod/{application}](http://localhost/api/somiod/%7bapplication%7d) onde application será o nome da aplicação que é suposto receber os seus dados, os dados são enviados por xml com toda a informação sobre uma application.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 15 Criação da resposta com os detalhes de uma application.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 16 Exemplo do xml com os detalhes de uma application

No mesmo método do ApplicationController que está o método de descobrir o detalhe de uma application também está o código que devolve todos os containers que estão dentro de uma aplicação pois ambos são conseguidos pelo mesmo endereço e pelo mesmo método HTTP GET sendo este o endereço [http://localhost/api/somiod/{application}](http://localhost/api/somiod/%7bapplication%7d) a única diferença é que para o pedido devolver todos os containers dentro de uma application é necessário enviar um HTTP Header com o nome “somiod-discover” e com o valor “container”

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

Figura 17 Verificação se existe o header somiod-discover e se o seu valor é container

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 18 Código para recolher todas os containers dentro de uma application

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Figura 19 Criação da resposta com o nome de todos os containers dentro de uma application

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 20 Exemplo do xml como todos os nomes dos containers dentro uma application

O último método que existe no ApplicationController é o método que tem o propósito para criar um container dentro de uma application para isso é necessário enviar um pedido HTTP POST para o endereço [http://localhost/api/somiod/{application}](http://localhost/api/somiod/%7bapplication%7d) onde application será o nome da aplicação onde será criado o container, para a criação do container é necessário enviar no body do pedido http apenas o nome que este container irá receber e o res\_type a informar que é um container, no SOMIOD é verificado se o nome vem vazio ou se o nome já existe dentro dessa aplicação e se o res\_type é o correto e se alguma destas premissas falhar será enviada uma mensagem a informar o utilizador, também é verificado se a application que vem no URI existe.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 21 Exemplo do xml que vem no body do pedido HTTP para a criação de um container

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 22 Verificação do res\_type e se o nome não vem vazio

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 23 Verificação se a application existe com o nome enviado

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 24 Codigo para a criação e inserção do container emu ma application

### ContainerController

O ContainerController irá lidar com todos os pedidos HTTP que serão feitos para o endereço [http://localhost/api/somiod/{application}/{container}](http://localhost/api/somiod/%7bapplication%7d/%7bcontainer%7d), para a atualização de um container o método HTTP usado será o PUT para o endereço referido acima no body do pedido HTTP será enviado um xml com o novo nome do container o SOMIOD vai verificar se o pedido está certo e se o nome não vem vazio e se ainda não existe nenhum container com aquele nome dentro da application ao qual este pertence.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 25 Verificação do res\_type

Uma imagem com captura de ecrã, texto

Descrição gerada automaticamente

Figura 26 Verificação se o nome não está vazio

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 27 Código que verifica se a application existe

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 28 Código que verifica se o container existe

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 29 Código que atualiza o container

Para a eliminação de um container é necessário enviar um pedido HTTP com o método DELETE para o endereço referido no anteriormente, o SOMIOD verifica se existe o container que se está a tentar eliminar.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Figura 30 Código que verifica que a application existe

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 31 Código que elimina o container e verifica se este existe

Com o endereço referido no início também é possível a criação de Datas e Subscriptions dentro de um container para isso é necessário enviar um pedido HTTP com o método POST para diferenciar quando é para criar uma subscription ou uma data é referido no body do pedido qual é o recurso a ser criado pelo res\_type se for data é para criar uma data se for subscription é para criar uma subscription para tal é necessário verificar se a application existe e se o container e se o name não vem vazio para ambos, para a data ainda é necessário verificar que o contente não vem vazio e para a subscription é preciso verificar que o atributo event vem com um valor válido que tem de ser 1,2 ou 3 e por fim o endpoint tem de começar com mqtt:// ou http:// e que contenham um endereço válido por exemplo (mqtt://127.0.0.1).

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 32 Código que verifica se o endpoint enviado é válido

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 33 Código para a criação de uma subscription

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 34 Exemplo xml para a criação de uma subscription

Na criação de uma Data é necessário verificar se existem subscriptions dentro do container que tenham o atributo 1 ou 3 e enviar o content da data criada para o endpoint dessas subscriptions

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 35 Código que verifica que tanto o name como o content não vêm vazios

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 36 Código para o envio do content da data para os endpoints das subscriptions do container

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 37 Código para inserir a data na base de dados

Uma imagem com texto, Tipo de letra, captura de ecrã, número

Descrição gerada automaticamente

Figura 38 Exemplo xml para a criação de uma data

O último método do ContainerController é o método que será chamado quando se envia um pedido HTTP com o método GET para o endereço referenciado no início, este método retornará os detalhes de um container ou todos os nomes das datas ou subscriptions para retornar as datas é necessário enviar um Header com o nome somiod-discover e o valor “data” e para as subscriptions com o valor “subscription” e para os detalhes do container é quando não existe nenhum Header.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 39 Verificação do Header e código para recolher as datas

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 40 Código para criar o xml com os nomes das datas

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

Figura 41 Exemplo do xml enviado como resposta

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 42 Verificação do Header e código para recolher as subscriptions

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 43 Código para criar o xml com os nomes das subscriptions

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 44 Exemplo do xml enviado como resposta

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 45 Código para recolher os detalhes de um container

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 46 Código para a criação de um xml com os detalhes de um container

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 47 Exemplo do xml com os detalhes do container

### DataAndSubController

Por fim existe o DataAndSubController que irá ter os métodos para leitura e eliminação tanto das datas como das subscriptions, o primeiro método do controlador é o método para a eliminação de subscriptions que é chamado quando é enviado um pedido HTTP com o método DELETE para o endereço [http://localhost/api/somiod/{application}/{container}/sub/{subscription}](http://localhost/api/somiod/%7bapplication%7d/%7bcontainer%7d/sub/%7bsubscription%7d) onde {subscription} é o nome da subscription que queremos eliminar

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 48 Código para eliminação de uma subscription

Para eliminar uma data apenas muda o endereço para o qual se envia o método HTTP sendo esse endereço o [http://localhost/api/somiod/{application}/{container}/data/{data}](http://localhost/api/somiod/%7bapplication%7d/%7bcontainer%7d/data/%7bdata%7d) onde {data} é o nome da data a ser eliminada. Ao eliminar uma data é necessário verificar se no container existem subscriptions para eliminação de datas ou seja subscriptions em que o atributo event seja 2 ou 3, ao eliminar uma data decidimos enviar para as subscriptions a informação que a data foi eliminada com sucesso.

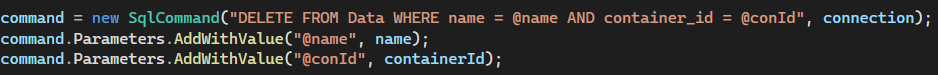


Figura 49 Código para eliminação de uma data

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 50 Código para notificação das subscriptions que uma data foi eliminada

Em seguida temos os métodos para obter os detalhes dos recursos começando pelo método para obter os detalhes de uma data que é chamado quando é feito um pedido HTTP GET para o SOMIOD com o endereço [http://localhost/api/somiod/{application}/{container}/data/{data}](http://localhost/api/somiod/%7bapplication%7d/%7bcontainer%7d/data/%7bdata%7d) onde {data} é o nome da data ao qual se quer obter os detalhes

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 51 Código para obter os detalhes de uma data

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 52 Código para a criação de um xml com os detalhes de uma data

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 53 Exemplo do xml enviado com os detalhes de uma data

Para finalizar o ultimo método do controlador é o método para obter os detalhes de uma subscriptions para isso é necessário enviar um pedido HTTP com o método GET para o seguinter endereço: [http://localhost/api/somiod/{application}/{container}/sub/{subscription}](http://localhost/api/somiod/%7bapplication%7d/%7bcontainer%7d/sub/%7bsubscription%7d) onde {subscription} é o nome da subscription que queremos obter os seus detalhes

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 54 Código para obter os detalhes de uma subscription

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Figura 55 Código para a criação de um xml com os detalhes de uma subscription

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 56 Exemplo do xml com os detalhes de uma subscription

## GARAGEM

A GARAGEM vai simular uma garagem inteligente se liga ao SOMIOD para que esta se comunique com o INTERRUPTOR que vai simular um interruptor da garagem para isso é necessário que a GARAGEM seja reconhecida pelo SOMIOD para isso ao criar a GARAGEM é enviado um pedido HTTP POST para a criação de uma application onde o seu nome será Garage

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 57 Código para a criação do xml da application Garage

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 58 Pedido HTTP para a criação da application no SOMIOD

Depois de ser criada a application é necessário criar um container dentro dessa application esse container vai ter o nome de garagem\_door que irá conter a subscription que será a própria garagem a ser ligada neste caso irá ter o nome de sub1 e o endpoint que essa subscrição vai ter vai ser o endpoint do MQTT Broker, ou seja, se quisermos adicionar mais algum recurso inteligente ao container como uma lâmpada é só adicionar mais subscriptions. Quando o container é criado também é feita a ligação com o MQTT Broker e é subscrito o tópico que contem o nome da application concatenado com o nome do container

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 59 Código que gera o xml para a criação de um container com o nome garagem\_door

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 60 Pedido HTTP para a criação do container garage\_door na application Garage

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 61 Subscrição no tópico garagegarage\_door



Figura 62 Código que gera o xml da subscription sub1 que vai ser enviada

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 63 Pedido HTTP para a criação da subscription sub1 dentro do container garage\_door dentro da application Garage

Depois de criados todos os componentes necessário para a interação entre a lâmpada e o interruptor APP\_A e APP\_B respetivamente agora a APP\_A vai estar apenas á escuta de mensagens enviadas para o tópico subscrito no MQTT.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 64 Função que fica a escuta de mensagens publicadas no tópico garagegarage\_door

## INTERRUPTOR

O INTERRUPTOR irá simular o interruptor para abrir e fechar a garagem para tal primeiro é criada a application Switch dentro do SOMIOD.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 65 Código para a criação do xml para criar a application Switch

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 66 Pedido HTTP para a criação da application SWITCH

Em seguida é apenas configurar os botões do “interruptor” para enviarem o recurso data para o servidor na application Garage no container garage\_door para o envio de uma data foi acordado que a app GARAGEM está à espera de receber um xml com o element status e o innerText abrir para abrir a garagem e fechar para fechar a garagem. A data criada no INTERRUPTOR para não ter o mesmo nome vai concatenar o nome com o timestamp em que a data é criada.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 67 Código para a criação de uma data para fechar/abrir a garagem

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 68 Pedido HTTP para a criação da data

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 69 Código que é chamado quando o utilizador clica no botão abrir do interruptor

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 70 XML enviado para o SOMIOD quando se clica no botão abrir

Uma imagem com Retângulo, símbolo, design

Descrição gerada automaticamente

Figura 71 A imagem da app GARAGEM muda para uma garagem aberta

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 72 Código chamado quando se clica no botão fechar do interruptor

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

Figura 73 Xml enviado para o SOMIOD quando se clica no botão fechar do interruptor

Uma imagem com Retângulo, símbolo, design, moldura

Descrição gerada automaticamente

Figura 74 Imagem da app GARAGEM muda para uma garagem fechada

No exemplo da comunicção entre a GARAGEM e o INTERRUPTOR o INTERRUPTOR no content envia um xml que é o esperado pela GARAGEM mas o content pode ser qualquer tipo de string.

## Component E

This is example text. This is example text. This is example text.

#### This is example text. xcvzxcvcxvzxcvzcxvzx

#### This is example text. Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

This is example text. Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

# Evaluation

This is example text. Blá, blá, blá introducing this main section...

## Test bed

Start by presenting the test bed (the HW and SW environment you used to test and evaluate your solution). So the first sub section can be called Test bed.

## Data analysis

For instance, you can present charts and data tables that describe the temperature and humidity behavior by measuring night and days...

## Data access

For instance, present here the min, max, avg and std deviation of the time taken to access story data for one month ago, 2, 10, 100 months, etc. as historical data is crucial for building monitoring applications. Again, present charts for that and describe them.

## Other1

Present other kind of metrics that characterize the behavior of your system in load states...

## Other n

Present other kind of metrics that characterize the behavior of your system in load states...

# Integration/App Development

This section exists only if any other applications were developed that used the exposed public API by this system. If so, presented the applications and the performance, capabilities, advantages to the community, etc.

## Application X

Blá, blá, Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,.

## Application Y

Blá, blá, Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,Blá, blá,.

# Conclusions and Future Work

Present conclusions here by recalling the main aim of this work and specific objectives and say if those were acomplished or not and the advantages of the system for the users and/or community. Then try to propose with brief detail some future work in order to improve and enhance the solution presented here.

# references

(every citation present in the text must be described here. Delete this)

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

# Appendix

*Appendix A*

Include the cRUD commands for crud operations for all supported resources.

*Appendix B*

Use this section to mention the work of each group member, required passwords, way of starting and runing the system, etc.