SOMIOD

Projeto desenvolvido no âmbito da unidade curricular de Integração de Sistemas

André Silva  
Departamento de Engenharia Informática  
Instituto Politécnico de LeiriaLeiria, Portugal  
2201724@my.ipleiria.pt

Tiago Conceição   
Departamento de Engenharia Informática  
Instituto Politécnico de LeiriaLeiria, Portugal  
2201751@my.ipleiria.pt

Ricardo Tomé   
Departamento de Engenharia Informática  
Instituto Politécnico de LeiriaLeiria, Portugal  
2202318@my.ipleiria.pt

*Abstract*—O objetivo deste projeto é criar um middleware orientado a serviços que promova a interoperabilidade entre diferentes aplicações, sendo definido uma uniformização na forma como os dados são acedidos, escritos e partilhados independetemente do domínio da aplicação. Os resultados da avaliação mostraram que a aplicação atingiu os objetivos propostos e apresentou um bom desempenho em termos de velocidade e estabilidade. A integração e o desenvolvimento da aplicação incluíram a criação de uma interface gráfica de utilizador e a integração com outras aplicações através de Web Services. As conclusões e trabalhos futuros destacam a importância da interoperabilidade e propõem a criação de novas funcionalidades para a aplicação

# Introduction

Neste projeto exploramos o campo emergente de middleware orientado a serviços, focando na uniformização do acesso, escrita e distribuição de dados em diversos domínios da aplicação. A motivação para este trabalho vem da necessidade crescente da interoperabilidade em sistemas heterogêneos, onde diferentes aplicações e serviços precisam de comunicar entre si de maneira eficiente e padronizada. O objetivo principal deste projeto é desenvolver um middleware orientado a serviços que facilite a interoperabilidade, utilizando Web Services para criar uma estrutura de recursos baseada na Web e alinhada com padrões web universais. Para alcançar isso, a aplicação proposta foi desenvolvida em C# utilizando a framework .NET e um sistema de messaging MQTT, com o auxílio de templates como Windows Form App para aplicações cliente e ASP.NET Web Application para serviços.

Este documento encontra-se dividido da seguinte forma:

II - Arquitetura do Sistema

III - Evaluation

IV - Integration/App Development

V - Conclusions and Future Work

VI - Appendix

# System Architecture

O sistema é composto pelo middleware “SOMIOD”, n clients e m Mosquitto brokers.

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

Descrição gerada automaticamente

1. System Architecture (*figure caption*)

## SOMIOD

O middleware tem como responsabilidade guardar organizar todos os tipos de dados que são enviados pelos clientes, assim como publicar esses dados nos endpoints indicados nas subscrições existentes no module onde os dados foram criados.

## Mosquitto Broker

Responsável por distribuir as mensagens que são publicadas aos corretos clientes

## Client

O cliente é quem vai interagir com o sistema, comunicando com o middleware e o mosquitto broker

# Evaluation

## Software usados para teste

Postman 🡪 usado para testar as rotas Rest API.

MQTT X 🡪 usado para testar as subscrições e publicações

## Preferências tomadas neste projeto

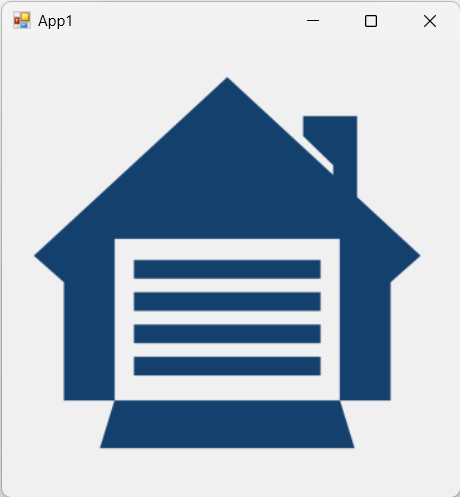
* Colocamos todas as rotas num único controlador, uma vez que estávamos a ter problemas de rotas para criá-las em controladores diferentes.
* Utilizamos regiões para dividir os pedidos HTTP no controlador, desta forma, debug e alteração de código torna-se mais simples.
* Separamos o SQL do controlador, tanto para evitar repetição de código, assim como para tornar a organização melhor e um debug mais fácil
* Na base de dados, tornamos o atributo 'Id' e ‘Name’ de todas as tabelas como unique, uma vez que achamos não fazer sentido existirem, por exemplo, 2 aplicações com o mesmo nome.
* No atributo ‘CreationDate’, caso a pessoa não insira nenhuma data de criação no body, optamos por usar DateTime.Now, porêm, acabamos por converter para String para enviar para a Base de Dados. Tomamos esta opção, uma vez que caso a pessoa deseje inserir uma data de criação no body, esta virá em String, e assim, não haverá problemas de conversão de diferentes tipos de variáveis.
* Só é possível fazer a operação DELETE ao applications se não existirem dependências com os filhos (containers), já os containers quando faz DELETE apaga também os seus (subs e datas).

# Integration/App Development

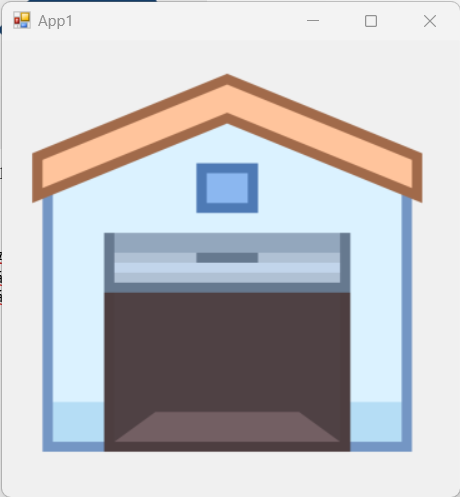
No âmbito de testar o middleware foram desenvolvidas duas aplicações do tipo Windows App Forms(.Net Framework).

## Application 1

Esta app cria uma aplicação com o nome “casa”, um container com o nome “garagem” e uma subscrição com “portão”, caso já esteja criado as seguintes funções na BaseData ambas as funções têm um if de verifição para evitar a criação de dados repetidos. A app utiliza a biblioteca M2Mqtt para se ligar a um servidor MQTT, que permite receber mensagens relacionadas com eventos da app. Através do RestSharp, efetua chamadas à API REST para criar entidades, como aplicação, container e subscrição. Antes de criar novas entidades, verifica se já existem na base de dados para evitar redundâncias. O método ”client\_MqttMsgPublishReceived” atualiza visualmente o estado do portão com base nas mensagens MQTT recebidas, alternando entre "aberto" e "fechado".



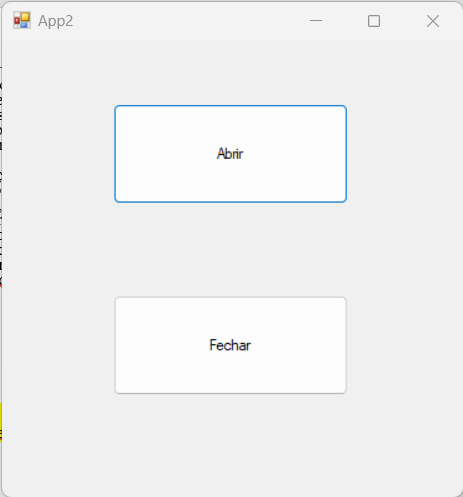
1. Application 1 interface (Fechado). (*figure caption*)



1. Application 1 interface (Aberto). (*figure caption*)

## Application 2

A app2 foi desenvolvida para interagir com o sistema da app1, focando em controlar um interruptor para abrir o “Portao” da App anterior. Ela utiliza a biblioteca RestSharp para se comunicar com a API REST, permitindo a criação de uma aplicação com o nome "interruptor" e o envio de comandos específicos ("Abrir" ou "Fechar"). A interface da app apresenta botões "Abrir" e "Fechar" para facilitar a interação do cliente, garantindo a consistência dos dados e a utilização das operações implementadas na app. Cada conjunto de dados é identificado por um nome único, baseado na data e hora atuais para evitar a redundância de dados.



1. Application 2 interface. (*figure caption*)

# Conclusions and Future Work

Concluímos este documento recapitulando o principal objetivo deste trabalho: desenvolver um middleware orientado a serviços que promova a interoperabilidade entre diferentes domínios de aplicação através da uniformização no acesso, escrita e distribuição de dados. Os objetivos específicos, que incluíam a adoção de Web Services e a implementação usando C#, .NET e MQTT, foram alcançados com sucesso. O sistema oferece uma interface padronizada e fácil de usar, reduzindo a complexidade e o tempo necessário para a integração de diferentes sistemas. Quanto ao trabalho futuro, propomos explorar a integração de tecnologias emergentes como Inteligência Artificial e Machine Learning para aprimorar a capacidade de análise e processamento de dados do middleware.

# *Appendix*

*Appendix A*

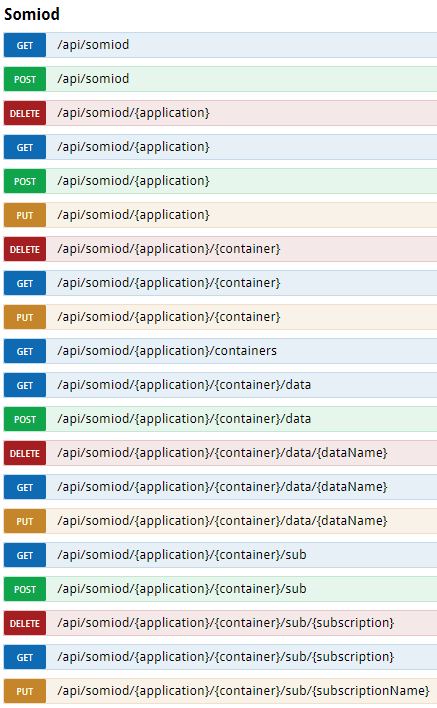


Figura 1Swagger Docummentation

CRUD do Aplicattion

GET (Lista de nomes de todas as aplicações):

curl --location 'http://localhost:57380/api

/somiod' \

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

--header 'Accept: application/xml' \

--header 'somiod-discover: application' \

GET:

curl --location 'http://localhost:57380/api

/somiod/app1' \

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

--header 'Accept: application/xml' \

POST:

curl --location 'http://localhost:57380/

api/somiod' \

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

--**data** '<Application>

<Name>app1</Name>

</Application>

'

PUT:

curl --location --request PUT 'http://localh

ost:57380/api/somiod/app1' \

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

--**data** '<Application>

<Name>AppY</Name>

</Application>'

DELETE:

curl --location --request DELETE 'http://localhost:57380/api/somiod/sim' \

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

--header 'Accept: application/xml' \

--**data** '<Application>

    <Name>nao pode</Name>

</Application>'

CRUD do Container

GET (Lista de nomes de todas os containers dentro da aplicação):

curl --location 'http://localhost:57380/api/somiod/casa/containers' \

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

--header 'Accept: application/xml' \

--header 'somiod-discover: containers' \

--**data** ''

GET:

curl --location 'http://localhost:57380/api/somiod/app1/container1' \

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

--header 'Accept: application/xml' \

--**data** ''

POST:

curl --location 'http://localhost:57380/api/somiod/app1' \

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

--**data** '<Container>

    <Name>container1</Name>

</Container>'

PUT:

curl --location --request PUT 'http://localhost:57380/api/somiod/nao/container1' \

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

--**data** '<Container>

<Name>cont1</Name>

</Container>'

DELETE:

curl --location --request DELETE 'http://localhost:57380/api/somiod/nao/cont1' \

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

--header 'Accept: application/xml' \

--**data** ''

CRUD do DATA:

GET(Lista de nomes de todos os data dentro do container)

curl --location 'http://localhost:57380/api/somiod/casa/garagem/data' \

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

--header 'Accept: application/xml' \

--header 'somiod-discover: data' \

--**data** ''

GET:

curl --location 'http://localhost:57380/api/somiod/casa/garagem/data/comando' \

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

--header 'Accept: application/xml' \

--**data** ''

POST:

curl --location 'http://localhost:57380/api/somiod/casa/garagem/comando' \

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

--**data** '<Data>

    <Name>container8</Name>

    <Content>**Exit**</Content>

</**Data**>'

PUT:

curl --location --request PUT 'http://localhost:57380/api/somiod/casa/garagem/data/comando'\

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

--**data** '<Data>

<Name>portao1</Name>

</Data>'

DELETE:

curl --location --request DELETE 'http://localhost:57380/api/somiod/casa/garagem/data/comando' \

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

--header 'Accept: application/xml' \

--**data** ''

CRUD do Subscription

GET(Lista de nomes de todos os subs dentro do container):

curl --location 'http://localhost:57380/api/somiod/casa/garagem/sub' \

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

--header 'Accept: application/xml' \

--header 'somiod-discover: sub' \

--**data** ''

GET:

curl --location 'http://localhost:57380/api/somiod/casa/garagem/sub' \

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

--header 'Accept: application/xml' \

--**data** ''

POST:

curl --location 'http://localhost:57380/api/somiod/app1/container1/sub' \

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

--**data** '<Subscription>

    <Name>sub1</Name>

    <Event>Create</Event>

    <Endpoint>127.0.0.1</Endpoint>

</Subscription>'

PUT:

curl --location --request PUT 'http://localhost:57380/api/somiod/casa/garagem/data/sub'\

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

--**data** '<Subscription>

<Name>sub1</Name>

</Subscription>'

DELETE:

curl --location --request DELETE

'http://localhost:57380/api/somiod/casa/garagem/data/sub' \

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

--header 'Accept: application/xml' \

--**data** ''

Como iniciar e correr o sistema de forma correta:

Após ter o projeto no respetivo computador pretendido, é necessário verificar se todos os ficheiros estão incluídos no projeto e a sua respetiva base de dados criada utilizando a query fornecida. Antes de fazer build à solução e por fim correr o projeto, deve iniciar o Mosquitto Broker no Services da sua máquina.

Trabalho realizado pelos membros do grupo:

Para realizar este projeto, tentamos dividir ao máximo o projeto em 4 partes iguais, de modo a que todos tivessem a mesma quantidade de trabalho, assim como dada a liberdade em trabalhar nos tópicos onde se sentissem mais à vontade.

Caso alguém tivesse dúvidas na sua parte do projeto, essa pessoa estaria à vontade de colocar questões aos restantes colegas de grupo, de modo a que todos conseguissem ajudá-lo a compreender melhor o que seria necessário de implementar.

Concluindo, todos os elementos do grupo trabalharam de forma igual, tendo sido este um grupo único e ajudando mutuamente para a conclusão deste projeto.