SOMIOD

Service Oriented Middleware

Miguel Lopes   
Engenharia Informática  
Instituto Politécnico de LeiriaLeiria  
2222397@my.ipleiria.pt

Jorge Oliveira  
Engenharia Informática  
Instituto Politécnico de LeiriaLeiria  
2221452@my.ipleiria.pt

Frederico Oliveira  
Engenharia Informática  
Instituto Politécnico de LeiriaLeiria  
2191604@my.ipleiria.pt

Francisco Ferreira  
Engenharia Informática  
Instituto Politécnico de LeiriaLeiria  
2221410@my.ipleiria.pt

*Abstract*

O projeto SOMIOD tem como objetivo desenvolver um middleware orientado a serviços para melhorar a interoperabilidade e promover dados abertos no ecossistema da Internet das Coisas (IoT). Utilizando uma estrutura de API RESTful, o middleware garante um método padronizado para acesso, escrita e notificações de dados, independente do domínio da aplicação. A organização hierárquica dos recursos — incluindo aplicações, containers, registos e notificações — permite uma integração fluida com padrões abertos da web. Para validar o middleware, foram implementados dois cenários de teste: uma lâmpada inteligente (App A) que utiliza notificações para mudanças de estado, uma aplicação móvel (App B) que dispara essas mudanças ao criar registos, e uma dashboard (Cenário 2) desenvolvida especificamente para testar todos os endpoints da API, garantindo a funcionalidade de operações CRUD+Locate e validando a robustez do sistema.

Keywords—SOMIOD, API RESTful, CRUD, IoT

# Introdução

A Internet das Coisas (IoT) tem desempenhado um papel crucial na resolução de desafios globais, como mudanças climáticas, sustentabilidade e eficiência energética. Ao integrar a recolha de dados em diversos ambientes, a IoT permite o desenvolvimento de aplicações inovadoras que beneficiam a sociedade, empresas e governos.

No entanto, a falta de interoperabilidade entre dispositivos e sistemas, causada por protocolos proprietários e serviços fechados, dificultando o compartilhamento de dados e a colaboração entre plataformas.

Para superar esse problema, este projeto propõe o desenvolvimento do SOMIOD, um middleware orientado a serviços com API RESTful para padronizar o acesso e manipulação de dados. Com suporte a notificações via HTTP e MQTT, o SOMIOD promove a interoperabilidade e o uso de padrões abertos da web.

Este relatório está estruturado da seguinte forma: a Seção 2 aborda a arquitetura do sistema; a Seção 3 descreve a implementação e as aplicações de teste; a Seção 4 apresenta a avaliação do sistema; e a Seção 5 traz conclusões e propostas de trabalhos futuros. Apêndices incluem comandos cURL e contribuições individuais.

# Arquitetura do Sistema

O SOMIOD (Sistema de Middleware para Objetos Distribuídos) é um middleware projetado para facilitar a integração e a comunicação entre sistemas distribuídos, fornecendo funcionalidades essenciais como recursos hierárquicos, persistência de dados e um formato de transferência de dados padrão.  
A arquitetura do SOMIOD pode ser dividida em três componentes principais:

-Recursos Hierárquicos: A arquitetura SOMIOD adota uma abordagem hierárquica para gerir recursos e os seus relacionamentos, que são organizados em quatro níveis principais: Aplicação, Container, Registro e Notificação.

-Persistência de Dados em Base de Dados: No contexto do middleware SOMIOD, foi usada uma base de dados criada no Visual Studio 2022, utilizando SQL Server LocalDB, é usado para persistir dados de objetos distribuídos.

-Formato XML para Transferência de Dados: A transferência de dados entre os componentes distribuídos do sistema é realizada no formato XML (Extensible Markup Language), que é amplamente adotado devido à sua flexibilidade e interoperabilidade.

## Application

Representa uma aplicação real que interage com o middleware. É o recurso de nível superior que agrupa Containers e os seus respetivos registos ou notificações. É possível criar, modificar, listar, localizar e eliminar uma aplicação. Cada aplicação possui um identificador único, nome e a data de criação. O middleware pode suportar múltiplas aplicações, cada uma funcionando de forma independente para organizar recursos associados.

## Container

Funciona como um agrupador para outros recursos, como registos (Records) e notificações (Notifications). Cada Container pertence a uma aplicação específica. Pode-se criar, modificar, listar, localizar e eliminar Containers. Organizam os dados de forma hierárquica e são associados a uma aplicação principal.

## Record

Representa cada dado registado em um Container específico, sendo tratado como um recurso associado a um Container. É possível criar, localizar e eliminar registos. Modificação não é suportada pela API. Cada registo inclui um identificador, nome, conteúdo, data de criação e o identificador do Container pai.

## Notification

Permite a configuração de notificações relacionadas a mudanças (criação ou eliminação de registos) de um Container. Cada Notification armazena informações como identificador, nome, data de criação, Container pai e o tipo de evento associado (criação ou exclusão). Também regista o endpoint para onde o middleware deve enviar a notificação via MQTT ou HTTP. As notificações devem ser disparadas sempre que um novo registo for criado ou eliminado do Container. Cada notificação define se o evento inclui criação ou exclusão.

## SOMIOD Database

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| Id | INT | Identity (1,1) | Not Null |
| Name | NVARCHAR(50) | [FK\_Application\_NomeTable] | Not Null |
| CreationDatetime | DATETIME |  | Null |

1. Table Application

Estrutura da Tabela Application:  
A tabela é associada à tabela NomeTable através da chave estrangeira (FK) na coluna Name.  
Foram criados gatilhos (triggers) na tabela Application para garantir que as operações de inserção, atualização e exclusão de dados sejam sincronizadas com a tabela NomeTable. Esses gatilhos mantêm a integridade dos dados e evitam duplicação de nomes, além de gerenciar a remoção de dados relacionados.

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| Id | INT | Identity (1,1) | Not Null |
| Name | NVARCHAR(50) | [FK\_Container\_NomeTable] | Not Null |
| CreationDatetime | DATETIME |  | Null |
| Parent | INT | [FK\_Container\_Application] | Null |

1. Table Container

Estrutura da Tabela Container:  
Está associada à tabela NomeTable através da chave estrangeira (FK) na coluna Name. Além disso, a tabela Container faz referência à tabela Application através da coluna Parent.

Foram criados gatilhos (triggers) na tabela Container para garantir que as operações de inserção, atualização e exclusão de dados sejam corretamente sincronizadas com a tabela NomeTable. Esses gatilhos garantem a integridade dos dados, evitam duplicação de nomes e gerenciam a remoção de dados relacionados, garantindo que, sempre que um container for inserido, atualizado ou excluído, as alterações sejam refletidas na tabela NomeTable de maneira consistente.

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| Id | INT | Identity (1,1) | Not Null |
| Name | NVARCHAR(50) | [FK\_Notification\_NomeTable] | Not Null |
| Endpoint | NVARCHAR(100) |  | Not Null |
| CreationDatetime | DATETIME |  | Not Null |
| Parent | INT | [FK\_Notification\_Container] | Null |
| Event | INT |  | Not Null |
| Enabled | BIT |  | Not Null |

1. Table Notification

Estrutura da Tabela Notification:

A tabela Notification armazena informações sobre notificações e está associada à tabela NomeTable através da chave estrangeira (FK) na coluna Name. Além disso, a tabela Notification também faz referência à tabela Container através da coluna Parent, que representa o container ao qual a notificação está associada.  
Foram criados gatilhos (triggers) na tabela Notification para garantir que as operações de inserção e exclusão de dados sejam corretamente sincronizadas com a tabela NomeTable. Esses gatilhos asseguram a integridade dos dados, evitam duplicação de nomes e garantem a remoção de dados relacionados, de forma que, quando uma notificação for inserida ou excluída, as alterações sejam refletidas na tabela NomeTable.

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| Id | INT | Identity (1,1) | Not Null |
| Name | NVARCHAR(50) | [FK\_Container\_NomeTable] | Not Null |
| Content | NVARCHAR(100) |  | Null |
| CreationDatetime | DATETIME |  | Null |
| Parent | INT | [FK\_Container\_Application] | Null |

1. Table Record

Estrutura da Tabela Record:

A tabela está associada à tabela NomeTable através da chave estrangeira (FK) na coluna Name. Além disso, a tabela Record também faz referência à tabela Container através da coluna Parent, que indica o container ao qual o registo pertence.

Foram criados gatilhos (triggers) na tabela Record para garantir que as operações de inserção e exclusão de dados sejam corretamente sincronizadas com a tabela NomeTable. Esses gatilhos mantêm a integridade dos dados, evitam duplicação de nomes e gerenciam a remoção de dados relacionados.

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| Name | NVARCHAR(50) | [PRIMARY KEY] | Not Null |

1. Table NomeTable

Estrutura da Tabela NomeTable:

A tabela NomeTable é responsável por garantir a unicidade dos nomes utilizados em várias tabelas da base de dados. Ela armazena todos os nomes que são referenciados nas tabelas Application, Container, Notification e Record.

# Avaliação

Nesta secção, inicia-se pela descrição do ambiente de testes utilizado para validar o middleware SOMIOD, detalhando os componentes de hardware e software que permitiram recriar cenários reais de utilização e analisar o desempenho do sistema.

## Ambiente de Testes

O ambiente de testes foi configurado utilizando os seguintes componentes de hardware e software:

Hardware:

* Broker MQTT Mosquitto configurado localmente para gerir notificações.

Software:

* IDE: Visual Studio 2022, utilizado para desenvolver e testar a aplicação.
* Base de Dados: SQL Server LocalDB, para persistência de dados do middleware SOMIOD.
* Middleware: API RESTful do SOMIOD implementada em C#, utilizando tecnologias abordadas nas aulas práticas.
* Ferramentas de Teste: Dashboard desenvolvido para testar os endpoints e POSTMAN para validação manual de chamadas REST.

# Integração/Desenvolvimento da Aplicação

## Cenário 1 (App A e App B)

No primeiro cenário, foi implementado um sistema de controlo para uma lâmpada inteligente, representado pela aplicação "Lighting" (App A), e um interruptor virtual, representado pela aplicação "Switch" (App B).

A aplicação "Lighting" é responsável por criar os recursos necessários no middleware, incluindo uma aplicação e um container associado chamado "light\_bulb".

Foi configurada uma notificação no container "light\_bulb" para monitorizar eventos de criação de registos. A aplicação A mantém-se à escuta de um canal do Mosquitto, um broker MQTT, para receber notificações enviadas pelo middleware.

A aplicação "Switch" (App B) permite ao utilizador criar registos no container "light\_bulb", utilizando um pedido POST através da API RESTful. A API SOMIOD processa este pedido e encaminha a mensagem para o Mosquitto. A aplicação "Lighting" lê a mensagem no canal e interpreta a ação, alterando o estado da lâmpada (ligar/desligar).

## Cenário 2 (Dashboard)

No segundo cenário, foi implementada uma aplicação dashboard com o objetivo de validar todos os endpoints da API do SOMIOD. Esta aplicação permite ao utilizador realizar as seguintes operações:

Criar, listar, eliminar e localizar recursos (aplicações, containers, registos e notificações). A dashboard permite também editar propriedades específicas de aplicações e containers, através da operação PUT, proporcionando flexibilidade para testar a funcionalidade de atualização parcial de recursos.

Introduzir dados personalizados para testar a funcionalidade de cada endpoint.  
A aplicação dashboard utiliza formulários simples para recolher parâmetros e executar operações através de chamadas à API RESTful do middleware.

# Conclusions and Future Work

O projeto SOMIOD alcançou o seu objetivo principal de desenvolver um middleware orientado a serviços para promover a interoperabilidade entre dispositivos IoT e o acesso a dados abertos. Ao implementar uma arquitetura RESTful e suportar recursos como aplicações, containers, registos e notificações, o SOMIOD provou ser eficaz em facilitar a comunicação entre diferentes tipos de dispositivos e sistemas. As aplicações de teste, como a Lighting (App A) e o Switch (App B), validaram a funcionalidade do sistema, demonstrando a capacidade do middleware de gerir eventos de criação, leitura, atualização e exclusão de dados. O SOMIOD também se destaca por permitir o envio de notificações por diversos canais, especialmente o MQTT, garantindo uma comunicação eficiente e em tempo real entre os dispositivos.

Além disso, garante a funcionalidade dos endpoints da API, como demonstrado no Cenário 2, onde o middleware foi testado em todas as operações CRUD (Criar, Ler, Atualizar, Excluir) e localizar, com o sistema respondendo adequadamente a solicitações de registo de dados e notificações. O cenário de teste completo validou a robustez e a confiabilidade do middleware na manipulação de dados em tempo real, mostrando a sua capacidade de interagir e responder de forma adequada em diferentes condições.

É importante salientar que algumas partes do projeto não estavam 100% explicitadas no relatório original, sendo muitas vezes necessárias interpretações para a implementação das funcionalidades. Dessa forma, todo o trabalho realizado foi baseado no que interpretamos do relatório, buscando sempre alinhar as expectativas e os requisitos do projeto da melhor maneira possível.

# Apêndices

*Apêndice A*

Get:

curl --location 'http://localhost:61626/api/somiod/' \

--header 'Accept: application/xml'

curl --location 'http://localhost:61626/api/somiod/Steam' \

--header 'Accept: application/xml'

curl --location 'http://localhost:61626/api/somiod/Steam/Game' \

--header 'Accept: application/xml'

curl --location 'http://localhost:61626/api/somiod/Discord/Conversa/record/Meus Dados' \

--header 'Accept: application/xml'

curl –location 'http://localhost:61626/api/somiod/Discord/Team/notif/Tópico' \

--header 'Accept: application/xml'

Get somiod-locate:

curl --location 'http://localhost:61626/api/somiod/' \

--header 'Accept: application/xml' \

--header 'somiod-locate: application'

curl --location 'http://localhost:61626/api/somiod/Steam' \

--header 'Accept: application/xml' \

--header 'somiod-locate: container'

curl --location 'http://localhost:61626/api/somiod/Steam' \

--header 'Accept: application/xml' \

--header 'somiod-locate: record'

curl --location 'http://localhost:61626/api/somiod/Steam' \

--header 'Accept: application/xml' \

--header 'somiod-locate: notification'

Post:

curl --location 'http://localhost:61626/api/somiod/' \

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

--data ' <Application>

<Name>App3</Name>

</Application>'

curl --location 'http://localhost:61626/api/somiod/App3/' \

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

--data ' <Container>

<Name>Con3</Name>

</Container>'

curl --location 'http://localhost:61626/api/somiod/Discord/Team/notif/' \

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

--data '<Notification>

<Name>NotificationNames</Name>

<Event>1</Event>

<Endpoint>http://example.com/endpoint</Endpoint>

<Enabled>true</Enabled>

</Notification>'

curl --location 'http://localhost:61626/api/somiod/Discord/Team/record/' \

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

--data '<Record>

<Name>test2</Name>

<Content>Acender Luz</Content>

</Record>'

Put:

curl --location --request PUT 'http://localhost:61626/api/somiod/App3' \

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

--data ' <Application>

<Name>App4</Name>

</Application>'

curl --location --request PUT 'http://localhost:61626/api/somiod/App4/Con3' \

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

--data ' <Container>

<Name>Con4</Name>

</Container>'

Delete:

curl --location --request DELETE 'http://localhost:61626/api/somiod/Discord' \

--data ''

curl --location --request DELETE 'http://localhost:61626/api/somiod/Discord/Conversa/' \

--data ''

curl --location --request DELETE 'http://localhost:61626/api/somiod/Discord/Team/record/test2' \

--data ''

curl --location --request DELETE 'http://localhost:61626/api/somiod/Discord/Team/notif/NotificationNames' \

--data ''

*Apêndice B*

Middleware 60% (Miguel e Frederico)

Testing applications 25% (Jorge – Cenário 2 e Francisco - Cenário 1)

Project report 15% (Miguel, Jorge e Francisco)

De forma a garantir o bom funcionamento do sistema devemos abrir e iniciar o Middleware SOMIOD primeiro, depois podemos iniciar os outros componentes (App A, App B e Dashboard), para o Cenário 1 (App A e B estão presentes dentro da solution da SOMIOD API) é preferível esperar um pouco para os programas executarem todas as suas tarefas não visíveis pelo utilizador, se nenhum erro for apresentado no ecrã as aplicações estão prontas para comunicar/funcionar. No Cenário 2 (Dashboard feita numa solution chamada Scenario2), com a API a correr podemos iniciar a aplicação e testar todos os endpoints (Get/Get somiod-locate/Post/Put/Delete) do Middleware, caso o utilizador insira dados de forma errada a aplicação mostra uma mensagem de pop-up de como fazer o respetivo pedido, caso o pedido seja bem feito a aplicação devolve a resposta do servidor